Scaling Conditional Random Fields for Natural Language Processing

Trevor A. Cohn

Submitted in total fulfilment of the requirements of the degree of Doctor of Philosophy

January, 2007

Department of Computer Science and Software Engineering
Faculty of Engineering
University of Melbourne
Abstract

This thesis deals with the use of Conditional Random Fields (CRFs; Lafferty et al. (2001)) for Natural Language Processing (NLP). CRFs are probabilistic models for sequence labelling which are particularly well suited to NLP. They have many compelling advantages over other popular models such as Hidden Markov Models and Maximum Entropy Markov Models (Rabiner, 1990; McCallum et al., 2001), and have been applied to a number of NLP tasks with considerable success (e.g., Sha and Pereira (2003) and Smith et al. (2005)). Despite their apparent success, CRFs suffer from two main failings. Firstly, they often over-fit the training sample. This is a consequence of their considerable expressive power, and can be limited by a prior over the model parameters (Sha and Pereira, 2003; Peng and McCallum, 2004). Their second failing is that the standard methods for CRF training are often very slow, sometimes requiring weeks of processing time. This efficiency problem is largely ignored in current literature, although in practise the cost of training prevents the application of CRFs to many new more complex tasks, and also prevents the use of densely connected graphs, which would allow for much richer feature sets.

This thesis addresses the issue of training efficiency. Firstly, we demonstrate that the asymptotic time complexity of standard training for a linear chain CRF is quadratic in the size of the label set, linear in the number of features and almost quadratic in the size of the training sample. The cost of inference in cyclic graphs, such as lattice structured Dynamic CRFs (Sutton et al., 2004), is even greater. The complexity of training limits the application of CRFs to large and complex tasks. We compare the accuracy of a number of popular approximate training techniques, which can greatly reduce the training cost. However, for most tasks this saving is coupled with a substantial loss in accuracy. For this reason we propose two novel training methods, which both reduce the resource requirements and improve the scalability of training, such that CRFs can be applied to substantially larger tasks.

The first method uses error-correcting output coding, a method originally devised for classifier combination (Freund and Schapire, 1996). This method decomposes the task of training a multiclass CRF (modelling a problem with three or more labels) into a series of simpler binary labelling tasks. Each of these sub-tasks are modelled with a CRF and are trained in the standard manner. Overall, these constituent models are considerably faster to train than a full multiclass CRF; critically, this can also reduce the complexity of training. Once trained these constituent models can be combined to decode unlabelled test instances and this results in similar accuracy to standard training.

We introduce a second alternative training method which uses feature constraints to improve the time cost of inference. These constraints tie groups of features in the model which are exploited in sum-product and max-product belief propagation (Pearl, 1988). This leads to faster training and decoding. Overall, even simple tying strategies
can lead to large reductions in training time, and for only a very small accuracy sacri-
fice. Smarter tying strategies, which integrate domain knowledge, have the potential to improve accuracy while also reducing the training time.

We provide empirical evidence showing that these methods allow CRFs to be scaled to previously intractable tasks, and allow for the use of more densely connected graphs. This is particularly important for NLP, where many tasks have large label sets, and cannot be modelled accurately using only local features. Both these methods are quite flexible, allowing the user to trade a small accuracy loss for a large reduction in the training time.
Declaration

This is to certify that

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface,
(ii) due acknowledgment has been made in the text to all other material used,
(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

(Trevor A. Cohn)
Preface

Some of the material presented in this thesis has been published. This applies to
(i) Chapter 6, parts of which were published in Cohn et al. (2005),
(ii) Chapter 7, parts of which were published in Cohn (2006)
The joint work (Cohn et al., 2005) was primarily my own work with only peripheral
ccontributions from my co-authors.

The feature sets used for the named entity recognition and shallow parsing experi-
ments in Chapters 5, 6 and 7 were based on work by, and in collaboration with Andrew
Smith (Smith et al., 2005). The semantic role labelling experiments in Chapter 7 borrowed
from the feature set and scripts developed in collaboration with Phil Blunsom (Cohn and
Blunsom, 2005).
Acknowledgements

I am extremely grateful to my supervisors, Steven Bird and Miles Osborne, who both provided invaluable guidance and feedback over the course of my studies. This thesis benefited from Steven’s incisive criticism and keen eye for detail. I am much indebted to Miles, whose enthusiasm for research was both infectious and often overwhelming. Without his brilliant guidance and penetrating criticism, I am sure the thesis would have been considerably poorer. Thanks also to my examiners, who both provided thorough and constructive comments on the thesis draft. I would like to thank my colleagues at the University of Melbourne and at the University of Edinburgh, particularly Phil Blunsom and Andrew Smith, who both provided stimulating discussions and invaluable feedback. I would like to acknowledge the financial assistance from a Melbourne Research Scholarship and a Postgraduate Overseas Research Experience Scholarship, which allowed me to broaden both my knowledge and my contacts by visiting Edinburgh for a year of my candidature. Lastly, I am extremely grateful to my partner, Charlotte Wilson, who was supportive of my return to studies, provided much-needed feedback and uprooted herself to accompany me in Edinburgh.
Table of Contents

1 Introduction 1
   1.1 Aim and Scope .................................................. 5
   1.2 Thesis Structure ............................................... 5

2 Graphical Models 7
   2.1 Directed Models ............................................... 8
   2.2 Undirected Models ............................................. 10
   2.3 Inference ...................................................... 13
      2.3.1 The Sum-Product Algorithm ............................. 14
      2.3.2 The Max-Product Algorithm ........................... 20
      2.3.3 Partially Observed Data ................................ 21
      2.3.4 Inference in Cyclic Graphs ............................ 22
   2.4 Conclusion .................................................... 26

3 Conditional Random Fields 29
   3.1 Hidden Markov Models ......................................... 31
   3.2 Conditional Markov Models ................................... 34
   3.3 Conditional Random Fields .................................. 38
      3.3.1 Graphical Structure and Factorisation ............... 39
      3.3.2 Features and the Maximum Entropy Principle ....... 40
      3.3.3 Potentials ............................................... 42
   3.4 Alternative Graphical Structures ............................ 43
   3.5 Inference in Conditional Random Fields .................... 45
      3.5.1 Marginalisation: The Forward-Backward Algorithm ... 45
      3.5.2 Decoding: The Viterbi Algorithm ..................... 46
   3.6 Estimation of Conditional Random Fields .................. 46
      3.6.1 Maximum Likelihood .................................... 47
      3.6.2 Maximum \textit{a Posteriori} ............................ 49
   3.7 Approximate Estimation Techniques ........................ 50
      3.7.1 Loopy belief propagation ............................... 51
      3.7.2 Pseudolikelihood ....................................... 51
List of Tables

2.1 Notational conventions used in this chapter. .......................... 8
4.1 Part of speech corpus sample statistics ................................. 62
4.2 Feature templates used for part-of-speech tagging ...................... 63
4.3 Chunking corpus sample statistics .................................. 65
4.4 Feature templates for joint part-of-speech tagging and chunking ....... 66
4.5 Named entity corpus sample statistics ................................. 66
4.6 Features templates used in the named entity recognition task. ......... 68
5.1 Timing and accuracy results for part-of-speech tagging ............... 84
5.2 Baseline timing and $F_1$ results for named entity recognition ......... 86
5.3 Results from seeding standard training with perceptron weights and feature sets .................................................. 87
5.4 Timing and accuracy results for joint part-of-speech tagging and noun phrase chunking .................................................. 90
6.1 Results for named-entity recognition ................................. 110
6.2 Part-of-speech tagging results on the small training sample .......... 112
6.3 Part-of-speech tagging results on the full training sample .......... 114
6.4 Results for joint part-of-speech tagging and noun phrase chunking .... 115
6.5 Timing and accuracy results using alternative codes ................. 123
6.6 Expert weights learnt from logarithmic-opinion pool training .......... 131
7.1 Part-of-speech tagging results on the small training sample .......... 150
7.2 Part-of-speech tagging results on the full training sample .......... 152
7.3 Named-entity recognition results .................................. 153
7.4 Results for joint part-of-speech tagging and noun-phrase chunking ... 154
7.5 A random sample of the POS transition clusters from the 300 cluster run. 155
7.6 Feature templates used for semantic role labelling ..................... 160
7.7 Semantic role labelling development results. ........................ 161
# List of Figures

2.1 Example directed graphical model .................................................. 9  
2.2 Example undirected model ............................................................. 11  
2.3 Cliques of three variables and their two possible factor graphs .......... 15  
2.4 Example factor graph ................................................................... 16  
2.5 Readiness in belief propagation ....................................................... 17  
2.6 Message passing schedule ............................................................... 18  
2.7 Transformation to remove evidence nodes ...................................... 22  
2.8 Messages passed around a loop can continue for ever. .................... 23  
2.9 Clustering and stretching graph transformations ............................... 25  
3.1 First order Hidden Markov Model (HMM) ....................................... 32  
3.2 First order Conditional Markov Model (CMM) ................................. 35  
3.3 Finite-state acceptor for POS tagging two sentences ........................ 38  
3.4 First order Conditional Random Field (CRF) ................................. 39  
3.5 Relational Markov Network (RMN) for web-page classification .......... 44  
3.6 Factorial dynamic Conditional Random Field (DCRF) ..................... 44  
4.1 Algorithm for calculating the matched-pairs statistic ....................... 70  
4.2 Architecture flow diagram, showing the files and processes involved. 71  
4.3 Sample XML snippet from a feature templates file. .......................... 73  
4.4 Multi-processor parallelisation used for training. ............................ 75  
5.1 Training time for part-of-speech tagging with varying sized training samples 80  
5.2 Feature counts for part-of-speech tagging with varying sized training samples 80  
5.3 Training evaluations for part-of-speech tagging with varying sized training samples ............................................................... 81  
5.4 Accuracy results for POS tagging trained on increasing amounts of data. 82  
5.5 Decoding time for part-of-speech tagging where the models are trained on varying sized training samples ................................. 83  
5.6 Averaged perceptron training curve for part-of-speech tagging. ........ 85  
5.7 Averaged perceptron training curve for named-entity recognition. .... 87
5.8 Averaged perceptron training curve for joint POS tagging and noun phrase chunking ................................................................. 90
5.9 Error distribution comparing the clumping of errors under the pseudolikelihood model and the standard model ........................................ 92
6.1 The Hamming $H(7,4)$ error-correcting code ................................. 100
6.2 Example error-correcting output codes for a four-class problem ........ 102
6.3 Decision boundaries of a simple error-correcting code .................. 102
6.4 Example error-correcting output coding decoding using the standalone and marginals methods ..................................................... 106
6.5 Learning curves for error-correcting CRFs .................................... 117
6.6 Two example error-correcting codes with different error-correcting capacities 118
6.7 Comparing column density and the number of decision boundaries in an error-correcting code ............................................ 120
6.8 Column density is correlated with the constituent model training time and error rate ................................................................. 121
6.9 Error correlations between constituent models in an ECOC ensemble ... 122
6.10 Accuracy vs. time trade-offs for alternative error-correcting codes .... 124
6.11 Sensitivity of accuracy to reordering of rows in an error-correcting code . 126
6.12 Accuracy curves for random codes of increasing size for named-entity recognition ............................................................... 128
6.13 Accuracy curves for random codes of increasing size for part-of-speech tagging ............................................................... 128
7.1 Venn diagram showing the set of labellings .................................... 138
7.2 Frequency of attested named-entity label bigrams ............................ 141
7.3 Example trie encoding the selected labellings .................................. 145
7.4 Pseudo-code to calculate the max-product messages .......................... 146
7.5 Alternative tries for sum-product inference .................................... 148
7.6 Part-of-speech tagging histogram over transitions ............................ 149
7.7 Part-of-speech tagging times ....................................................... 151
7.8 Part-of-speech tagging accuracy versus the number of selected transitions 151
7.9 Accuracy for feature tying using transition clusters on the small POS tagging task ............................................................... 156
7.10 Syntax tree labelling with semantic roles ..................................... 157
7.11 Proposition-bank entry for the verb ‘to sell’ ................................... 158
7.12 Factor graph used for semantic role labelling ................................ 159
Chapter 1

Introduction

This thesis deals with Conditional Random Fields (CRFs; Lafferty et al. (2001)), a new and powerful model for predicting structured labellings. CRFs are increasingly popular primarily in natural language processing (NLP) and also in other domains such as computational biology, where structured labelling tasks are commonplace. NLP tasks ranging from document classification to part-of-speech (POS) tagging to syntactic parsing can all be considered as structured labelling tasks – where the labelling is comprised of the topic label, a sequence of POS tags or a syntactic parse tree respectively. All of these tasks can be modelled using a CRF, often with state-of-the-art results (Berger et al., 1996; Lafferty et al., 2001; Clark and Curran, 2004).

CRFs are particularly applicable to natural language processing for a number of reasons, which are described below:

1. CRFs allow the joint modelling of complex structured labelling tasks which should not be decomposed into a number of smaller independent tasks. For example, the construction of a parse tree should not be decomposed into a series of independent decisions; rather a local decision to label a span of words as a constituent of a given type will affect the decisions for both its parent and child constituents, which recursively affect their children and parents. Therefore the prediction process must consider the complex interdependence between these decisions.

2. CRFs provide a rich and largely unconstrained feature representation. For example, a feature might detect that a word ending in *ing* is labelled as a gerund verb (tag *VBG*), or that the word *bling* is labelled as a noun (*NN*). These features may reference overlapping and non-independent aspects of the data, as shown in the previous example. Associated with each feature is a model parameter which assigns a weight to the feature. CRFs can use very large feature sets, often numbering in the millions, which allow for flexible and expressive modelling power.

3. CRFs are probabilistic models which describe a probability distribution over labellings. Therefore they inherently represent ambiguity in their predictions; this
can be used to better inform other algorithms which use the model’s predictions as input to perform further processing. Complex NLP applications commonly employ the cascade architecture where many prediction tasks are performed in series with the results of each task informing all subsequent tasks. In this case prediction errors in early tasks lead to many more errors in downstream tasks (Sutton et al. (2004) demonstrate the effect of cascading errors in POS tagging followed by NP chunking). For this reason, probabilistic models are ideal, in that they can preserve ambiguity in their predictions, thereby limiting one source of errors in the cascade.\footnote{Despite this benefit, most NLP cascades use 1-best predictions rather than preserving ambiguity. This is often a pragmatic choice, as it allows many models to be integrated with the minimum effort.}

Moreover CRFs can be used to jointly model many tasks. A single model can replace the cascade, where this model makes a structured prediction over the joint space of labellings at all levels (Taskar et al., 2002; Sutton et al., 2004). The joint model allows the decisions at each layer to inform the decisions at other layers and therefore the model can find the mutually optimal prediction. This contrasts with the traditional cascade which assumes a fixed order of tasks with no mechanism for feedback.

4. The model can be fitted to a training sample using a maximum likelihood estimate. This finds the parameter values which maximise the likelihood of the training data under the model. Moreover, the model may be regularised with a prior over the parameter values. This provides a practical and elegant way of smoothing the model and thereby reducing overfitting of the training sample. Overfitting is a big problem for CRFs, which are often extremely expressive with thousands or millions of features, and therefore parameters. However, overfitting can be limited by using a simple Gaussian prior (or other distribution) which discourages over-reliance on any single feature.

5. The model is discriminative, i.e., it predicts the labelling conditioned on some observations. The observations are typically sequences of words, or other contextual data. This setup exactly matches the testing configuration, where the model is supplied with the observations and must predict a labelling. The conditioning allows the features of the model to reference arbitrary dependent aspects of the observations, and also allows features over larger groups of observations, such as word \textit{n}-grams or phrases. These features are often very difficult to include in other similar models (e.g., Hidden Markov Models).

6. Inference operations can efficiently marginalise or maximise the distribution: for example, the belief propagation algorithm can find the maximum probability labelling or find the probability for a full or partial labelling (Pearl, 1988). These operations use dynamic programming to avoid enumerating the full space of labellings which is often exponentially large in the number of labels. These inference operations
instead have a polynomial complexity for most types of task. Inference allows for both prediction and supervised training – finding the parameters which best model a training sample.

For these reasons CRFs have been widely adopted by the NLP community. Many of the compelling reasons to use CRFs also apply to other models, although CRFs are one of the few models which combines all these benefits in one model. CRFs are a type of graphical model; this umbrella category encompasses probabilistic models over structured labelling spaces and includes other types of model such as Bayesian networks and Markov Random Fields. General graphical models share the benefits 1, 3, 4 and 6 listed above. CRFs combine these strong benefits in a discriminative framework. Discriminative models describe a conditional distribution over labellings given some observations, while their counterparts, generative models, model the joint distribution over the labelling and observations. Both types of model can be used for prediction and other forms of inference, however discriminative models allow for a more flexible feature representation, as they do not need to model the interdependence between the observations or assume-away the dependence. Coupled with the CRF’s maximum entropy parameterisation, this leads to the benefits 2 and 5 listed above.

While CRFs are extremely well suited to language tasks, all their benefits do not come for free. The model has two main failings. Firstly, as a consequence of their flexible feature representation, the models are often used with an extremely large feature set and therefore have an equally large parameter space. This allows for a very expressive model which can fit the training sample very precisely, without sufficient inductive bias to ensure strong accuracy. This overfitting can be countered to some extent by smoothing the model – most commonly by including a prior over the parameter values. However, there is compelling evidence that simple priors do not provide sufficient regularisation; i.e., the model still could benefit from further smoothing, as demonstrated in Smith and Osborne (2005) and Sutton and McCallum (2005).

Secondly, CRFs often take a very long time to train especially when compared to similar generative directed models (e.g., Hidden Markov Models). This is because there is no closed form solution for maximising the data likelihood in a CRF, even for fully observed (supervised) training data. For this reason the likelihood is usually optimised by an iterative process, which is guaranteed to find the optimal values, but often requires many hundreds of iterations to do so. Supervised training in generative directed models is often trivial, while for CRFs it can take weeks on a large cluster of machines.

The thesis addresses this second failing: the high cost of training (estimating) a CRF on a fully observed training sample. This is a significant issue as it prevents CRFs being applied to very complex or large tasks. Current estimation methods use a numerical optimiser to maximise the data likelihood. This requires repeated inference over the training sample, which leads not only to an expensive training stage, but also one that
Chapter 1. Introduction

scales poorly. The cost of training varies with the amount of training data, the number of labels (raised to a power) and the number of features. Therefore it is impractical to apply the model to tasks with large label sets, large training sets, or extremely rich feature representations.

Training is already quite slow for tasks with modest sized label sets. For example, it takes almost a month to train a linear chain CRF on a one million word training sample for part-of-speech tagging. This task has 45 labels, which explains the slow training time: the training complexity for a chain CRF is quadratic in the size of the label set. This does not augur well for other sequence labelling tasks with larger label sets, for example super-tagging which can have many hundreds or thousands of labels (Bangalore and Joshi, 1999). For these tasks the quadratic complexity means training is infeasible.

The richness of the underlying graphical structure also has a strong impact on the cost of training. While a typical chain-structured CRF has a training cost quadratic in the number of labels, in more densely connected graphs (those with larger maximal cliques) the power increases to a higher polynomial order. Furthermore, when the graph contains one or more cycles, inference is even more costly: exact inference is rendered intractable, and approximate methods must then be used, which themselves are often very expensive. These densely connected graphs allow for richer features over the labelling than would otherwise be permitted, and therefore allow the CRF to better model complex tasks. Clearly the cost of inference is a limiting factor which discourages the use of richly connected graphical structures, and severely limits the application of CRFs to tasks with very large label sets.

To date, these efficiency and scalability problems with standard CRF training have been addressed by various approximation methods. Structural approximations, such as pseudolikelihood (Besag, 1974, 1975) or the piecewise estimator (Sutton and McCallum, 2005), optimise a much simpler formulation than the likelihood by making strong independence assumptions. Other techniques include perceptron training (Roark et al., 2004), which optimises a different loss function instead of the data likelihood. On standard datasets the accuracy of these methods is often substantially lower than that achieved by regular likelihood maximisation, although the training time is often substantially reduced. However, very few of these methods provide a scalability improvement – that is, a reduction in the asymptotic complexity of training. Pseudolikelihood is the exception here, as it removes the polynomial dependence on the label set, although it is the most coarse of the approximations. Feature induction has also been suggested as a technique for reducing training cost (McCallum, 2003). This limits the number of unnecessary features cluttering the model and slowing the estimation process, while also limiting the model’s ability to over-fit the training data. However, the process of feature induction itself is very time consuming, and only minimises one of the factors in the training complexity.

---

\(^2\)The Penn Treebank WSJ section (Marcus et al., 1994). The time was measured on a large cluster of Intel Xeon 3Ghz machines.
1.1 Aim and Scope

The primary aim of this thesis is to scale CRF training to larger tasks in natural language processing. Task size refers to the amount of training data, the size of the feature set, the size of the label set and the structure and connectedness of the graph. Inference in undirected trees (which subsume linear chains) is most simple, while inference in cyclic graphs is considerably more difficult, as is inference in graphs with large cliques even if the graph forms an undirected tree.

The outcomes of the thesis are alternative training methods which are faster than standard maximum likelihood training, as demonstrated on tasks in which current methods are practical. In addition, the complexity of these methods is lower than that of standard training, such that the novel methods have a slower growth in the cost of training as the task is made more complex. The last of these aims contrasts with previous alternative training methods, which typically do not reduce complexity, and therefore do not provide better scaling characteristics. In addition these new methods result in strong accuracy, and provide a high fidelity approximation of standard training.

While the primary focus is on training efficiency, minimising the decoding cost is a secondary aim. Decoding involves finding the best labelling given some observations. Typically decoding time is small in comparison to training time, however, it has an identical complexity to training and thus suffers from the same scaling issues. The cost of decoding can present problems in deployment scenarios, where the algorithm must be run a user’s basic computer workstation. As such, faster decoding will make CRFs more applicable for a wider range of tasks.

Methods which satisfy these aims will allow CRFs to be used more efficiently (both training and decoding) on currently practical tasks, but more importantly, these methods will allow CRFs to be applied to new ‘large’ tasks, particularly those with large label sets or complex graphical structures. This would allow the many compelling benefits of CRFs to be realised for many new tasks, potentially leading to many improvements in accuracy over current models.

The thesis will consider training in the conventional manner (Lafferty et al., 2001): using a supervised (fully observed) training sample and optimising a point estimate of the model parameters (e.g., the maximum likelihood (ML) or maximum a posteriori (MAP) parameter estimates). This specifically excludes Bayesian approaches, which perform inference under all possible parameter configurations (Qi et al., 2005), and semi-supervised or unsupervised training, where the training data is not fully observed.

1.2 Thesis Structure

The first two chapters of the thesis examine graphical modelling literature. Chapter 2 reviews graphical models, contrasting directed and undirected models. It also presents
Chapter 1. Introduction

Belief propagation (Pearl, 1988), a generic inference algorithm for graphical models, which can be used to marginalise or maximise the distribution described by the model. Chapter 3 describes CRFs following Lafferty et al. (2001), showing how belief propagation can be using for training and decoding CRFs. This chapter also covers existing methods for estimating CRFs, including a survey of current approximations which are used to improve inference time and, to some extent, scalability.

The experimental setup is described in Chapter 4. This describes the NLP tasks to which we have applied the model and the modelling configuration used for each task. The tasks are named entity recognition (NER), part-of-speech (POS) tagging and combined POS tagging and noun-phrase chunking (NPC). These tasks are of increasing difficulty in terms of training efficiency: NER has a small label set, while the POS label set is considerably larger; the first two tasks are modelled with a linear chain, while the last uses a cyclic lattice. The chapter also describes the evaluation method used to collect and compare timings and accuracy results. Finally, the chapter presents an overview of the software developed for the thesis.

Chapter 5 presents results on the three tasks using standard training, which maximises the likelihood of the training sample. This provides a clear view as to how characteristics of the task affect training and decoding times and the model’s accuracy. In addition, some popular alternative training methods are also evaluated on these tasks. These timing and accuracy results motivate the search for better alternative training methods, and also serve as a baseline for subsequent approximation methods presented in the thesis.

Chapter 6 proposes an alternative method of training which addresses the cost of using large label sets. This method uses an error-correcting output coding framework for model combination. This allows a number of very simple models to be trained separately which are then later combined together to perform decoding. This results in a considerably lower training cost than a standard CRF. Moreover, the technique results in similar accuracy to standard training, and in one case with significantly better accuracy.

Chapter 7 presents a second, more direct method to optimise exact inference in CRFs. Faster inference leads to faster training and decoding. This method exploits regularities in the feature set to optimise expensive sums and maximisations during inference. These regularities may be enforced by feature constraints, therefore approximating the full model with a constrained model. Ideally these constraints are engineered using domain knowledge, in which case the method can provide an accuracy benefit as well as an efficiency gain. Even naive methods of automatically inducing these constraints can lead to substantial decreases in training and decoding times.

Finally, Chapter 8 summarises the findings of the thesis and presents avenues for future work.
Chapter 2

Graphical Models

This thesis investigates Conditional Random Fields (CRFs; Lafferty et al. (2001)), probabilistic models for labelling sequence data. CRFs are typically used for sequence tagging tasks where a sequence of words must be annotated with a sequence of labels, one per word. In this scenario they model a joint probability distribution over label sequences conditioned on the words in the sentence. Inference operations allow this distribution to be fitted to a data sample or maximised to predict the most probable label sequence, for instance. CRFs were designed to address problems in Conditional Markov Models (CMMs; McCallum et al. (2001)), which were designed to be more flexible versions of Hidden Markov Models (HMMs; Rabiner (1990)). All three models are types of graphical models, which differ in their conditioning, independence assumptions and their factorisation. The inference procedures for the three models can be understood as specialisations of belief propagation, a general inference technique for graphical models. This chapter describes the two types of graphical models – directed and undirected – and the belief propagation algorithm. This provides the necessary background for the reader to understand the following chapters which build upon this foundation.

Graphical models are probabilistic systems used to model joint probability distributions over sets of random variables. Complex joint distributions are commonly used in natural language processing (NLP) to model phenomena such as the distribution of words in a passage (language modelling), the grammatical structure of these words (parsing), the translation process between passages of text in different languages (machine translation), word based tagging (part-of-speech tagging, named-entity recognition, etc.), and a plethora of other tasks. Graphical models allow such complex distributions to be factorised into the product of many simpler functions, each ranging over a smaller event space than the full joint distribution. The graph encodes independence assumptions between random variables, which yields the factorisation of the distribution. Inference methods allow us to efficiently query the joint distribution, using the graphical structure to avoid redundant computation. Possible inference operations include (a) finding the probability of a reali-
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_i$</td>
<td>a random variable, or the node for $X_i$ in a graph</td>
</tr>
<tr>
<td>$X$</td>
<td>a set of random variables, $X_1, X_2, \ldots, X_N$</td>
</tr>
<tr>
<td>$x_i, x'_i$</td>
<td>a realisation of random variable $X_i$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>a realisation of random variables $X = (X_1, X_2, \ldots, X_N)$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>a realisation of random variables $X$ after excluding $X_i$, i.e., $X \backslash X_i$</td>
</tr>
<tr>
<td>$\pi_i$</td>
<td>the parent indices of node $i$ in a directed graph</td>
</tr>
<tr>
<td>$C$</td>
<td>the set of cliques in the graph</td>
</tr>
<tr>
<td>$N(i)$</td>
<td>the neighbour indices of node $i$ in a graph</td>
</tr>
<tr>
<td>$f_i$</td>
<td>factor $i$ from a factor graph</td>
</tr>
<tr>
<td>$\psi_{f_i}$</td>
<td>a potential function over factor $f_i$</td>
</tr>
<tr>
<td>$\phi_{f_i}$</td>
<td>a log-potential function over factor $f_i$</td>
</tr>
<tr>
<td>$Z$</td>
<td>the normalisation constant, or partition function</td>
</tr>
</tbody>
</table>

Table 2.1. Notational conventions used in this chapter.

Inference operations allow the fitting of the model to a data set (estimation, or training) and the application of the resulting distribution to unlabelled test data (decoding), for example. This chapter presents both directed graphical models (Bayesian Networks) and undirected models (Markov Random Fields, or Markov networks), highlighting their differences and similarities. The chapter reviews the belief propagation algorithm (Pearl, 1988), a general inference method for undirected trees, which can be applied to both directed and undirected models. The chapter also reviews some methods for inference in cyclic structures: both loopy belief propagation (Pearl, 1988), an approximate inference algorithm, as well as graph transformations to convert cyclic graphs into undirected trees such that belief propagation may then be used for exact inference.

## 2.1 Directed Models

Directed graphical models (or Bayesian or belief networks) are described by a directed graph $G = (X, E)$, where $X = \{X_1, X_2, \ldots, X_N\}$ are the nodes, and $E = \{(X_i, X_j)\}$ are the edges (see Jordan, 2002). This graph has no directed cycles. Each node represents a random variable (for simplicity we refer to a variable and its node interchangeably as $X_i$), while edges indicate possible dependencies between these random variables. We use the common notation in which upper case letters denote random variables (and nodes) while lower case letters denote realisations. These conventions are summarised in Table 2.1.

---

1 A realisation of a random variable is a value taken by the variable. For example if the random variable $X$ describes the result of a coin toss, $X = \text{heads}$ and $X = \text{tails}$ are the two possible realisations.
2.1. Directed Models

Directed graphical models describe a family of probability distributions:

\[ p(x_1, \ldots, x_N) = \prod_{i=1}^{N} p(x_i | \pi_i) \]  

(2.1)

where the relation \( \pi_i \) indexes the parent nodes of \( X_i \) (the sources of incoming edges to \( X_i \)), which may be the empty set. Each function on the right hand side of (2.1) is a conditional distribution over a subset of the variables in \( X \); each function must return positive scalars which are appropriately normalised.

An example directed model describing a set of five random variables is shown in Figure 2.1. This graphical structure implies the parent relationships \( \pi_1 = \emptyset \), \( \pi_2 = \{1\} \), \( \pi_3 = \pi_4 = \{2\} \) and \( \pi_5 = \{3,4\} \). Using (2.1) this yields the following factorisation:

\[ p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2)p(x_5|x_2, x_3) \]  

(2.2)

To interpret the meaning of (2.2), we first compare the factorisation to that produced using the standard chain rule of probabilities. The chain rule allows us to factorise a joint distribution into a product of distributions. There are many possible factorisations; one such expansion is:

\[ p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_1, x_2, x_3)p(x_5|x_1, x_2, x_3, x_4) \]  

(2.3)

This represents the joint distribution over \( X \) as the product of five distributions. When we compare (2.3) with (2.2), we notice that a number of conditioning variables are omitted in (2.2). For example, the third term \( p(x_3|x_2) \) is missing \( x_1 \) from its conditioning context. This omission represents a conditional independence relation, encoding our knowledge about the lack of inter-relatedness between the variables, and thus simplifies the joint probability distribution. Many more independence and conditional independence relations between sets of variables are encoded by the graph. The set of independence relations can be found using the Bayes’ Ball algorithm (Jordan, 2002), which analyses paths connecting these sets of variables for possible sources of dependence.

Formally, two sets of random variables \( X \) and \( Y \) are independent iff

\[ p(x, y) = p(x)p(y) \]
which we denote $X \perp Y$. Recall that lower case letters denote a realisation of the upper case variable. Conditional independence is defined as follows: $Z$ is conditionally independent of $X$ given $Y$ iff

$$p(z, x | y) = p(z | y)p(x | y)$$

which is denoted $Z \perp X | Y$.

Graphical models are commonly used in NLP to describe evolving phenomena over time, such as the production of words to make a sentence. These models typically make a Markov assumption over the sequence, i.e., the current state can be summarised by a fixed sized history over previous items. For example, a $k^{th}$ order language model assumes that $X_t \perp X_{t-k-1}^t | X_{t-k}^{t-1}$. Markov assumptions preclude the modelling of common long distance effects such as noun verb agreement, case marking etc. While modelling the full joint distribution would allow the model to be more expressive and accurately model these long-distance effects, inference over the joint distribution is rarely tractable. In addition, the sheer number of parameters of the full joint distribution would make it particularly hard to estimate from small training samples. Instead, using a Markov assumption to decompose the joint distribution into a series of small distributions allows for tractable inference while still modelling the core of the distribution.

### 2.2 Undirected Models

Now that we have briefly reviewed directed models, we turn to undirected models, contrasting the properties of the two. Undirected models (also known as Markov Random Fields or Markov networks) are another class of graphical model, which represent a different factorisation of the joint distribution to that of directed models, and with different conditional independence semantics. The following review is based on Jordan (2002).

An undirected model is a graph $G = (X, E)$, where $X = \{X_1, X_2, \ldots, X_N\}$ are the nodes, and $E = \{(X_i, X_j) : i \neq j\}$ are the undirected edges. This graphical structure describes a family of probability distributions:

$$p(x_1, \ldots, x_N) = \frac{1}{Z} \prod_{c \in C} \psi_{X_c}(x_c)$$

(2.4)

such that $\psi_{X_c}(x_c) > 0$, $\forall c, x_c$.

where $C$ are the set of cliques in the graph (subsets of variables in $X$), $Z$ is the normalising partition function and $\psi$ are strictly positive real-valued potential functions.

The term clique describes a subset of nodes that are fully connected: every pair of nodes in the subset are connected by an edge. A maximal-clique is a clique that cannot be enlarged with additional nodes while still remaining fully connected. For example, the graph in Figure 2.2 contains the following cliques: $\{X_1\}, \{X_2\}, \{X_3\}, \{X_4\}, \{X_5\}, \{X_1, X_2\}, \{X_2, X_3\}, \{X_2, X_4\}, \{X_3, X_4\}, \{X_5\}, \{X_2, X_3, X_4\}, \{X_2, X_3, X_5\}, \{X_3, X_4, X_5\}$. Of these, only three are maximal cliques; $\{X_1, X_2\}, \{X_2, X_3, X_4\}$, and $\{X_3, X_4, X_5\}$, in that these three cliques
subsume all the remaining cliques in the graph. We can limit the product in (2.4) to only maximal cliques without any loss of generality. This requires the potential functions, $\psi$, for each non-maximal clique to be incorporated into the potential function of exactly one of its subsuming maximal cliques.

![Figure 2.2. Example undirected graphical model depicting a system of five random variables.](image)

The partition function, $Z$, in (2.4) ensures that $p$ is a valid probability distribution, i.e., $\sum_{x} p(x) = 1$. This is achieved by summing out the numerator in (2.4) for every possible realisation:

$$Z \triangleq \sum_{x_1} \sum_{x_2} \cdots \sum_{x_N} \prod_{c \in C} \psi_{X_c}(x_c)$$

(2.5)

where each clique $c$ indexes a subset of the variables in $X$, as defined by its edges in the graph. This large summation arises as a direct consequence of the mostly unconstrained potential functions, $\psi$, in (2.4). Directed graphical models use conditional probability distributions as their potential functions and thus they need no normalisation, i.e., $Z = 1$. However, this constant is not necessarily one for undirected models, and often proves intractable to calculate. This is because it requires summing over a number of realisations which is exponential in the number of random variables. For sparsely connected graphs, this summation can be made more efficient by dynamic programming. Fortunately in many NLP applications sparse graphs are commonly used to model complex language phenomena. For example, trigram language models and Hidden Markov Models both use strong Markov assumptions to reduce the number of edges in the graphs. Efficient summation is performed by the belief propagation algorithm, an inference method for both directed and undirected graphical models, which is described in Section 2.3.

An alternative representation avoids the positivity constraints in (2.4) by defining each potential function as an exponential:

$$\psi_{X_c}(x_c) = \exp \phi_{X_c}(x_c)$$

(2.6)

This allows (2.4) to be re-parameterised over only the unconstrained functions, $\phi$, which
we call the log-potentials:

\[
p(x_1, \ldots, x_N) = \frac{1}{Z} \prod_{c \in C} \exp \phi_{X_c}(x_c) \tag{2.7}
\]

\[
= \frac{1}{Z} \exp \sum_{c \in C} \phi_{X_c}(x_c) \tag{2.8}
\]

with the benefit that the exponential can be moved outside the product, to yield (2.8). The exponent of (2.8) is often referred to as the negative energy.\(^2\)

Equation (2.8) allows the global probability to be characterised with local consistency functions over small neighbourhoods, i.e., the log-potential functions defined over maximal cliques. Unlike the directed model, these consistency functions need not satisfy the laws of conditional probability, being completely unconstrained. This property allows considerable freedom in designing the potential functions, which itself is a compelling reason for using undirected, rather than directed, graphical models. For example Conditional Random Fields (CRFs) are undirected models which define the log-potential functions as linear functions over a large set of features, where each feature detects aspects of the realisation. The same feature representation can also be used in a directed model, which results in the Conditional Markov Model (CMM). The similarities and differences between these models are described in Chapter 3.

**Conditional Independence** The conditional independence semantics for undirected graphs differ from those of directed graphs. In undirected graphs, we state that \(X_A \perp X_B \mid X_C\) when the nodes of \(C\) separate the two sets \(A\) and \(B\) in the graph. That is, all paths connecting a node in \(A\) to a node in \(B\) must pass through a node in \(C\). This is considerably simpler than the semantics for directed graphs, described in Section 2.1. Directed and undirected graphical models are capable of representing different types of conditional independence relations which can not be represented in the other model (Jordan, 2002). However, for the very simple graphs commonly used in NLP tasks (i.e., chains and trees), the difference between the independence semantics of the two models is often very subtle.

The Hammersley-Clifford theorem proves the equivalence between the family of distribution satisfying all the conditional independence relations thus defined from a graph and the family of distribution described by (2.8) on the same graph, under all possible potential functions (Hammersley and Clifford, 1971; Clifford, 1990).

The conditional independence semantics for undirected graphs also describe a Markov condition between variables. The marginal probability over a single node, \(X_i\), when given a realisation of the remaining nodes can be expressed as \(p(x_i \mid x_{\bar{i}}) = p(x_i \mid x_{N(i)})\), where \(x_{\bar{i}}\) is the realisation over all nodes except \(X_i\) and \(N(i)\) are the indices of the nodes in

\(^2\)The term energy comes from statistical physics, where undirected models are used to model the behavior of a system of particles in a lattice configuration (Yedidia et al., 2003). Here the negative exponent can be interpreted as the total energy of the system, which is affected by the interactions between the states of neighbouring particles in the lattice.
neighbourhood of $X_i$ (i.e., those nodes which are connected to $X_i$ by an edge). We can
derive this relation as follows:

$$p(x_i | x_i) = \frac{p(x_i, x_i) \sum_{x'_i} p(x'_i, x_i)}{\sum_{x'_i} p(x'_i, x_i)}$$ (2.9)

$$= \frac{1}{Z} \exp \sum_{c \in C} \phi_{c}(x_i)$$ (2.10)

$$= \frac{1}{Z} \exp \sum_{c \in C : i \in c} \phi_{c}(x_i)$$ (2.11)

$$= \frac{p(x_i, x_{N(i)})}{\sum_{x'_i} p(x'_i, x_{N(i)})}$$ (2.12)

$$= p(x_i | x_{N(i)})$$ (2.13)

Equation (2.10) replaces the distributions $p(\cdot)$ in (2.9) using the definition in (2.8). We use
the prime symbol, $x'_i$, in the denominator to avoid confusion with the variable $x_i$. In the
denominator the sum over $x'$ is constrained such that all elements of the vector are equal
to $x$ except for at index $i$, i.e., we are still summing over a single variable. The potential
functions over cliques not involving $X_i$ can be cancelled as they appear as factors in the
the numerator and denominator. This results in (2.11) where the summations, $\sum_{c \in C : i \in c}$,
now range only over cliques which include index $i$. This can be further simplified to
(2.12) and finally (2.13). Equation (2.13) encodes a Markov assumption inherent in the
graphical structure: that the distribution over a variable is conditionally independent of
all other variables, given the values of its neighbouring nodes: its Markov blanket.

In summary, directed and undirected graphical models have similar graphical forms,
independence semantics and functional forms. The subtle differences in the indepen-
dence semantics of the two models leads to a different set of normalisation constraints –
directed models require their ‘potentials’ to be normalised over each edge, while undi-
rected models require one global normalisation. As a consequence the potentials in the
undirected models are considerably less constrained, which affords the modeller addi-
tional flexibility in designing the potential functions. As we will see in Chapter 3 this
flexibility allows the use of rich feature based potential functions which are capable to
very accurately fitting data. The flip-side is that the global normalisation function ties
together all the features in maximum likelihood estimation, which can render training
very costly or even intractable.

2.3 Inference

Both directed and undirected graphical models factorise a complex joint distribution
in component functions. The two types of model have subtly different conditional inde-
pendence semantics, yielding factorisations which differ primarily in the manner of
normalisation. They are also estimated from data using different methods. (These meth-
ods are described in Chapter 3 for HMMs, CMMs and CRFs.) Both types of model are useful, but the choice of which to use depends on the nature of the task.

The process of inference describes the operations which we can perform using the graphical model. In practical NLP tasks, inference is often required for training a graphical model on a data sample and in using the resulting model to predict maximum a posteriori realisations. Inference over acyclic graphs can be performed using the belief propagation algorithm. This section first describes the belief propagation algorithm for marginalisation (sum-product) and maximisation (max-product) over realisations, and then goes on to describe loopy belief propagation, an approximate inference method for cyclic graphs. Finally, some graph transformation methods are presented, which convert a cyclic graph into an equivalent acyclic one, such that belief propagation may be used for exact inference.

### 2.3.1 The Sum-Product Algorithm

The sum-product algorithm is used to find marginal distributions over single nodes or cliques in the factor graph. These marginals are required in many practical situations, for example, in maximum likelihood training of Conditional Random Fields, as described in Section 3.6. The marginal distribution over a single variable, \( X_i \), can be found by marginalising out the remaining variables:

\[
p(x_i) = \sum_{x' \cap x'_i = x_i} p(x') = \frac{1}{Z} \sum_{x' \cap x'_i = x_i} \prod_{c \in C} \psi_c(x'_c)
\]  

(2.14)

where the undirected factorisation from (2.4) has been used. This marginalisation requires summing out every possible realisation, which would be intractable if done naively, as there are exponentially many realisations in the number of random variables. Belief propagation assumes an undirected tree, and therefore the right hand side of (2.14) can be decomposed into the product of independent factors for each branch of the graph emanating from \( X_i \). That is, the cliques in the graph may be partitioned into sets which lie only in one subgraph or else connect \( X_i \) to that subgraph. This reflects the conditional independence relations inherent in the underlying directed or undirected graph, i.e., \( X_A \perp X_B \mid X_i \) where \( A \) and \( B \) are the variable indices of two disjoint subgraphs incident on \( X_i \). The marginals from (2.14) can be restated as:

\[
p(x_i) = \frac{1}{Z} \psi_i(x_i) \prod_{t \in T_i} \left( \sum_{x'_t = x_i} \prod_{c \in C_t \setminus \{i\}} \psi_c(x'_c) \right)
\]  

(2.15)

where \( T_i \) are the undirected subtrees incident on node \( X_i \), and rightmost product over potentials ranges over all the cliques in \( t \) while excluding the single node clique for \( X_i \). This ensures that \( \psi_i(x_i) \) is calculated only once, as the first element in the numerator of (2.15).

The sum-product algorithm (Pearl, 1988) formalises the process of marginalisation over an undirected graph in an inductive manner. This is achieved using dynamic
2.3 Inference

Programming to calculate each bracketed element of the product in (2.15). These terms are referred to as messages and encode the effect of summing out all configurations of the source subgraph on the state of node $X_i$. Messages are vector quantities, with an element for each realisation of $X_i$, and are denoted $m_{X_j \to X_i}(x_i)$ where $X_j$ is the nearest node of the source subgraph (i.e., it is directly connected to $X_i$ with an edge). The sum-product algorithm defines the messages inductively, such that the full set of messages over every edge in both directions may be found efficiently, and thereby allowing the marginal distributions to be recovered at every node in the graph.

**Factor Graphs** Before describing sum-product belief propagation, we will first describe a useful notational aid: the factor graph (Kschischang et al., 2001). Factor graphs explicitly represent the factorisation of the joint probability distribution. Factor graphs are not strictly required for belief propagation, but simplify the exposition of the algorithm for graphs with cliques of three or more nodes. This is illustrated in Figure 2.3, where the first two graphs show directed and undirected graphs with a triple clique (the directed model is parameterised by a distribution over the three variables). The remaining two graphs are factor graphs which show factors in the decomposition as squares. Either of these factor graphs could be used to describe the directed or undirected graph; the choice of which makes explicit whether the distribution is factored into one function or three separate functions.

Factor graphs are bipartite graphs, with nodes for each random variable and for each (possibly maximal) clique, which are called factors. Undirected edges connect variable nodes to factors, such that each factor is connected to all the nodes in its clique (the variable $X_c$ in its potential function $\psi_{X_c}(x_c)$). Formally, $G = (X, F, E)$, where $X$ are the random variables, $F$ are the factors, and $E = \{(X_i, F_j)\}$ are the (undirected) edges linking nodes to factors. These graphs do not describe any conditional independence relations,
Chapter 2. Graphical Models

Figure 2.4. Example factor graph, corresponding to the graphs in Figures 2.1 and 2.2.

but instead only describe the factorisation of a complex function:

\[ g(x) = \prod_{f_j \in F} g_{f_j}(x_{f_j}) \]  

(2.16)

where the function \( g(x) \) is represented as the product of functions \( g_{f_j} \) over subsets of the variables \( X \). These graphs can describe equivalent factorisations to the joint probability factorisation in undirected graphs by equating the factor-functions to the potentials \( g_{f_j} = \psi_{f_j} \) and introducing a zero-sized factor with potential \( \frac{1}{Z} \).

An example factor graph is shown in Figure 2.4; this is the equivalent representation of the undirected graph in Figure 2.2. There are three factors (shown as squares), one for each maximal clique in the undirected graph, or conditional probability distribution in the directed graph. Associated with each factor is a function, the potential for the corresponding clique. The factor graph shown in Figure 2.4 also represents the factorisation of the directed graph in Figure 2.1. Directed graphs can be converted into equivalently factorised undirected graphs by the process of \textit{moralisation}, which links the parents of any node which has more than one parent (“marrying” them), after which the directions on the edges are dropped to yield the undirected graph (Jordan, 2002).

Message Passing  We review sum-product belief propagation on factor graphs, following Kschischang et al. (2001) and Yedidia et al. (2003). The belief propagation (BP) algorithm uses the notion of message passing over edges in the factor graph, with messages being passed over every edge in the graph, in both directions. After this process, the node and factor marginals can be calculated simply from the message values.

We now consider messages between connected factor graph elements: from factors to nodes and from nodes to factors. We modify our notation to support these two types of message, denoting messages from factor \( j \) to node \( i \) as \( m_{f_j \rightarrow X_i}(x_i) \) and messages from node \( i \) to factor \( j \) as \( m_{X_i \rightarrow f_j}(x_i) \). Both the incoming and outgoing messages from node \( i \) are parameterised over \( x_i \), and not the larger event space of the factor \( f_j \).

The messages from nodes to factors are defined as:

\[ m_{X_i \rightarrow f_j}(x_i) \propto \prod_{f_k \in N(X_i) \setminus f_j} m_{f_k \rightarrow X_i}(x_i) \]  

(2.17)
2.3. Inference

Figure 2.5. Illustration of “readiness” within belief propagation. The node must wait before sending a message over the dashed arrow, until it has received incoming messages on all other edges (the solid arrows). This same notion also applies to factors.

where $A \setminus b$ denotes the set $A$ after excluding member $b$, and $\mathcal{N}(\cdot)$ is the neighbourhood function, returning the neighbours of the given node or factor in the factor graph. Because the factor graph is bipartite all the neighbours of nodes are factors and all the neighbours of factors are nodes. The proportional relation in (2.17) does not require the messages to be normalised as long as the message values for each $x_i$ in $m_{X_i \rightarrow f_j}(x_i)$ use the same scaling constant. Together the message values over the same edge and direction are called a message vector, or just a message. For the practical reasons of numerical overflow, these messages are usually normalised to sum to one, $\sum_{x_i} m_{X_i \rightarrow f_j}(x_i) = 1$. These messages simply aggregate all incoming messages into the node for each value $x_i$.

The messages from factors to nodes are defined as:

$$m_{f_j \rightarrow X_i}(x_i) \propto \sum_{x'_j : x'_j = x_i} \psi_{f_j}(x'_j) \prod_{x_i \in \mathcal{N}(f_j) \setminus X_i} m_{X_i \rightarrow f_j}(x'_i)$$  \hspace{1cm} (2.18)$$

where $x_{f_j}$ is the vector of values included in the factor, which must include $x_i$. The message combines the influence of all incoming messages to the factor, together with the potential, $\psi_{f_j}$. As with messages from nodes to factors, the messages in (2.18) are often normalised to sum to one. The sum ranges over all realisations of the variables, connected to the factor constrained such that $X_i = x_i$. As a result, calculating these messages for large factors can be computationally expensive: the number of realisations is exponential in the factor size.

The message passing process begins at the leaves in the factor graph, i.e., those nodes and factors which are connected to only one factor or node. As the algorithm requires an acyclic factor graph, these leaves are guaranteed to exist. Each leaf, which is either a node or a factor, passes a message vector over its only edge. The ensuing steps require nodes and factors which have received incoming messages to then send messages to their neighbours. These outgoing messages may only be sent over an edge when incoming messages have been received on all other edges, such that the message values on the right hand sides of (2.17) and (2.18) are properly defined. Therefore, once messages have been received from (at least) all its neighbours bar one, the node or factor is ready to send a message. This is illustrated in Figure 2.5, where the message following the dashed arrow cannot be sent until all three input messages have arrived, as shown with dark arrows.

Message passing continues from ready nodes and factors, gradually covering the graph. At completion, when there are no further ready entities, messages will have been
sent over all edges, in both directions. Deadlock or infinite looping cannot occur, as the factor graph is (assumed) acyclic. Therefore propagation starting at the leaves will activate parent nodes recursively and reach the root of the tree, after which messages can flow back down the tree to the leaves.

Figure 2.6. Illustration of the order of message passing for belief propagation over an example factor graph. The numbers on the arrows indicate the order in which the messages must be sent. The left graph shows the ready based schedule, and the right shows the collect and distribute schedule with \( f_1 \) chosen as the root, where the two passes are shown with solid and dashed arrows, respectively.

The order of message passing for an example factor graph is shown in Figure 2.6a. Note that this graph was chosen to be different to the previous running example from Figure 2.1 which would have first required cycle-removal (see Section 2.3.4.2) which would then leave a chain factor graph: this would be inappropriate for illustrating general inference in undirected factor trees. The arrows show message calculation, and the numbers on each arrow indicate the order in which these messages are passed. The messages labelled 1 are sent before messages labelled 2, etc. We assume for simplicity that any number of messages may be sent simultaneously at a given time (in practise, the order specified here would give rise to many synchronous one-message-per-time schedules). Here we can see that the messages from \( X_1 \) to \( f_1 \) and \( X_2 \) to \( f_1 \) can both occur immediately at \( t = 1 \), as \( X_1 \) and \( X_2 \) are both leaves. After these messages have been sent, this licenses \( f_1 \) to send its message to \( X_3 \), because \( f_1 \) has received incoming messages over all its remaining edges. This then licenses the messages from \( X_3 \) to \( f_2 \) and so forth.

The message passing schedule can equivalently be considered as a two-pass schedule, with an inward sweep (collect) followed by an outward sweep (distribute). First, we must nominate an arbitrary root node. Then we collect the messages towards this root, starting from the leaves, and progressively moving up the tree. Secondly, we distribute messages back down the tree, starting at the root and terminating at the leaves. In such a manner, messages are only sent when their input messages have arrived, and at completion, messages have passed in both directions along every edge. An example tree based schedule is shown in Figure 2.6b, where \( f_1 \) was arbitrarily chosen as the root. The messages in collect pass are numbered 1–3, and messages numbered 4–6 form the
Beliefs  Once the message passing process has finished, these messages can be used to find the marginals at each node. Each node, \( X_i \), now has incoming messages over all its edges: each message summarising the effect of its respective subgraph on the node. As such, we simply combine the messages in a product to find the marginals, as follows:

\[
b_{X_i}(x_i) \propto \prod_{f_j \in N(X_i)} m_{f_j \rightarrow X_i}(x_i)
\]  

(2.19)

where the normalisation constant can be found by summing the right hand side of (2.19) over all \( x_i \). The per-message normalisation constants are irrelevant here, as each occurs equally often for different values of \( x_i \). Thus they only affect the value of the final normalising constant. We refer to the resulting marginals as beliefs, denoted \( b_{X_i}(x_i) \) (following Pearl (1988), who developed the algorithm for belief networks).

We can also find the marginal distributions over larger sets of variables. This is quite simple for the variables involved in a factor, \( f_j \):

\[
b_{f_j}(x_{f_j}) \propto \psi_{f_j}(x_{f_j}) \prod_{X_i \in N(f_j)} m_{X_i \rightarrow f_j}(x_i)
\]  

(2.20)

where once again we normalise over the set of realisations of \( x_{f_j} \). The beliefs for larger sets of variables can be reconstructed from the node and factor beliefs in (2.19) and (2.20). The beliefs for a complete realisation, \( x \), is given below:

\[
b(x) = \prod_{X_i} b_{X_i}(x_i) \prod_{f_j} b_{f_j}(x_{f_j}) \prod_{X_i \in N(f_j)} b_{X_i}(x_{f_j})
\]  

\[
\quad = \prod_{f_j} \frac{b_{f_j}(x_{f_j})}{\prod_{X_i \in N(f_j)} b_{X_i}(x_{f_j})} \prod_{X_i} b_{X_i}(x_i)^{|N(X_i)|-1}
\]  

(2.22)

Equation (2.21) is an elegant factorisation of the joint probability in terms of the node and factor marginals, and (2.22) is a simplified restatement, where \( |N(X_i)| \) is the number of factors to which node \( X_j \) is connected. For trees, this formulation simplifies to the chain rule expansion for the joint probability after application of the conditional independence relations inherent in the graph. For the running example in Figure 2.6a, the joint probability can be represented as:

\[
p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2, x_3|x_1)p(x_4|x_3)p(x_5|x_3)
\]  

(2.23)

\[= p(x_1)\frac{p(x_1, x_2, x_3)p(x_4|x_3)p(x_3|x_5)}{p(x_1)p(x_3)p(x_5)}
\]  

(2.24)

\[= \frac{p(x_1, x_2, x_3)p(x_4|x_3)p(x_3|x_5)}{p(x_3)^2}
\]  

(2.25)

which is consistent with the formula in (2.22). This method for finding the joint probability avoids separately calculating the partition function, \( Z \). Due to the arbitrary message normalisation constants from (2.17) and (2.18), the belief normalising constants will no longer
equal the partition function, $Z$. However, these per-message normalisation constants are of practical value: numerical overflow and underflow is common during message passing, where the floating point representation of real numbers is incapable of storing such large or small numbers, and instead sets them to infinity or zero. Keeping the messages suitably normalised (e.g., such that each message vector sums to one) limits these numerical problems. Using log-potentials, $\psi$, in the place of the potentials, $\phi$, in the message calculations ((2.17) and (2.18)) coupled with per-message normalisation provides a more robust solution.

### 2.3.2 The Max-Product Algorithm

Now that we’ve seen how belief propagation can be used to find the marginal distributions over nodes and factors, and to calculate the joint probability over the full graph, we now consider how this algorithm can also be used for decoding. This process finds the maximising realisation for the joint probability, i.e., $x^* = \arg\max_x p(x)$. This corresponds to the typical testing scenario: given partially observed data, predict the best labelling for the hidden parts. In many natural language scenarios, the partially annotated data is a sequence of unannotated tokens forming a passage of text. The model must predict a complete annotation for these tokens; the predicted annotation is chosen to maximise the joint probability.

Belief propagation can be used to solve the decoding task, where it is called the max-product algorithm. This algorithm uses the same messages from nodes to factors as the sum-product algorithm, in (2.17), along with a different definition of the message from factors to nodes:

$$
m_{f_j \to X_i}(x_i) \propto \max_{x'_j} \psi_f(x'_j) \prod_{X_k \in \mathcal{N}(f_j) \setminus X_i} m_{X_k \to f_j}(x'_k)
$$

The sum from (2.18) has been replaced with a max operator, and as such the messages describe the maximum probability, rather than the total probability, over all assignments.\(^3\)

The max-product algorithm follows a similar message passing schedule to that previously describe for the sum-product algorithm. However messages only need to be passed in a single direction over each edge. This corresponds to a single collect pass, where the beliefs are aggregated to a nominated root node, $X_i$. After this pass, the maximum a posteriori value for $X_i$ is known:

$$
x^*_i = \max_{x_i} \prod_{f_j \in \mathcal{N}(X_i)} m_{f_j \to X_i}(x_i)
$$

\(^3\)More generally, belief propagation can operate with any pair of operations that form a commutative semi-ring, in place of the sum and product operations in the sum-product algorithm. Both operations $\otimes$ and $\oplus$ must be associative and commutative, and obey the distributive relation $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$. Such pairs of operators include $\otimes = \times$ and $\oplus = +$ (sum-product); $\otimes = \times$ and $\oplus = \max$ (max-product) and $\otimes = +$ and $\oplus = \max$. This last pair (max-sum) is useful for maximising in logarithmic space, where logarithmic addition is equivalent to multiplication, which can be used to avoid floating point over- and under-flow.
2.3. Inference

We now have a single value, $x^*_i$, from the maximising configuration, $x^*$, but still must find the remaining elements. If we were to perform a distribute pass, we could find values for each variable which form part of a maximising configuration. However, there is no guarantee that they all would form part of the same maximising configuration. Therefore, we supplement the messages from factor to node, as used in the collect pass, with the indices of the maximising configuration:

$$
\delta_{f_j \rightarrow X_i}(x_i) = \arg \max_{x'_j = x'_i} \psi_{f_j}(x'_j) \prod_{X_k \in N(f_j) \setminus X_i} m_{X_k \rightarrow f_j}(x'_k) \quad (2.28)
$$

The maximising configuration can be found from these indices, $\delta$. Starting with $X_i = x^*_i$, we choose the maximising configuration for each connected factor, which includes $X_i = x^*_i$, i.e., $x^*_j = \delta_{f_j \rightarrow X_i}(x^*_i)$. After this first step, we will have the maximising values for a larger set of nodes: those which are connected to $X_i$ by a factor. We continue this process, expanding $x^*$ to include the maximising configuration over each factor encountered (and thus connected nodes). Once we have finished this process, we have a complete maximising configuration, $x^*$.

2.3.3 Partially Observed Data

Now we have seen how belief propagation can be applied to acyclic factor graphs, both in the form of the sum-product and max-product algorithms. These algorithms can be used to find marginal probabilities over subgraphs, and to find a maximising configuration, respectively. These inference methods can be performed efficiently, when compared to the naive method of enumerating all possible configurations.

The earlier discussion of the sum-product and max-product algorithms assumed that the values of all variables were hidden. Marginalisation and maximisation were performed over the full set of variables. In many instances inference must be performed given some evidence, i.e., after observing the values of a subset of the variables. In these scenarios we wish to operate on the conditional distribution $p(x_H | x_E)$, where $H$ and $E$ refer to the sets of hidden and evidence variables respectively. Thus the marginalisation and maximisation operations only range over variables from $X_H$, while treating $X_E$ as fixed.

Partially observed scenarios are the norm in natural language processing, which typically involve the annotation of raw text. Often, only the text itself is provided, and the model must perform inference operations over the space of annotations. This is often required for training of graphical models, and in order to use these models for prediction on a testing sample, where the (hidden) annotation with maximal probability is chosen. Even when fully observed data is provided, inference over partially observed subsets may be necessary: e.g., for supervised (fully observed) training of conditional undirected models, as described in Section 3.6.

Fortunately, the same algorithms may be used for such partially-observed configurations. The effect of the evidence nodes can be factored into the potential functions,
and thereafter the evidence nodes can be ignored. Inference is then performed on the subgraph containing only hidden nodes, using standard sum-product and max-product as described above.

Figure 2.7a shows an example partially observed graph, where three of the six variables are observed. This factor graph corresponds to a Hidden Markov Model or Conditional Random Field, with three state variables. In inference, the evidence nodes are removed, as shown in figure Figure 2.7b, and the potentials over the factors \( f_c, f_d \) and \( f_e \) replaced such that the evidence variables are clamped to their observed values. For example, the new potential for \( f_c \) is \( \psi_{f_c} (x_1) = \psi_{f_c} (x_1, \bar{x}_4) \), where \( \bar{x}_4 \) is the observed value for \( X_4 \). If an observed node is not associated with any factors linking it two a hidden node, it is ignored: it contributes a equal factor to the numerator and denominator of \( p(x_H|x_E) = \frac{p(x_H,x_E)}{p(x_E)} \), which cancels out.

### 2.3.4 Inference in Cyclic Graphs

While the elimination and belief propagation algorithms described earlier presupposed an acyclic factor graph, in practice cyclic graphs are common: we would like to be able to perform inference on such graphs. For example, Taskar et al. (2002) uses cyclic graphs to model of web-page classification. Here each page is allocated a node in graph an edges are created between nodes whenever one pages links another. Cycles are present in the graph whenever there is a cyclic chain of hyper-links. Cyclic graphs also feature in other models used in natural language processing, such as factorial Hidden Markov Models (Ghahramani and Jordan, 1995) and dynamic Conditional Random Fields (Sutton et al., 2004), as described in Chapter 3.

We present two alternative ways to deal with cyclic graphs: (a) convert the factor graph into an acyclic graph by graph transformation, which conjoins or expands the nodes and factors, or (b) use loopy belief propagation on the graph to perform approximate inference (Pearl, 1988).
2.3. Inference

2.3.4.1 Loopy BP

We first review *loopy BP* – belief propagation applied to cyclic graphs – following Yedidia et al. (2003) and Kschischang et al. (2001). Loopy BP is an approximation which uses repeated application of standard belief propagation until convergence. (For undirected trees, convergence is guaranteed after two passes.) At convergence, the beliefs or maximising configuration can be calculated in the same way as for standard belief propagation.

The message definitions in (2.17), (2.18) and (2.26) are unmodified, as are the belief equations: (2.19), (2.20) and (2.22). The message passing schedule, however, must differ, as the previously described method required a tree for completion. That is, the messages are sent around the graph until no nodes or factors are “ready”, i.e., they have received incoming messages on all edges bar one and have not already sent a message over the remaining edge. In a cyclic graph, such messages can be sent *ad-infinitum* around a loop, as shown in Figure 2.8. After the four messages are sent anti-clockwise from $X_1$ to $X_2$ to $X_3$ to $X_4$ to $X_1$, the first node $X_1$ will be ready to send a second message to $X_2$. This in turn provokes another round of message passing, which provokes another, and so on.

Instead of passing messages until no more messages can be passed, we assess the convergence of the message values. In many graphical structures the message values settle to a fixed-point after a time. Therefore, we stop message passing when we find one of these fixed-points, i.e., when messages have been passed over every edge in both directions, and this results in a sufficiently small change in the message values. There is no general proof of convergence of loopy-BP, and while there are certain graphs for which loopy-BP will provably diverge, convergence is achieved in the majority of cases (Weiss, 2000; Tatikonda and Jordan, 2002; Heskes, 2004; Ihler et al., 2005). Furthermore, at convergence, the beliefs are not guaranteed to match the true marginals (or maximum values in the max-product case). In practice, the beliefs are usually quite a close approximation to the real values (Ihler et al., 2005).

The question still remains how to start propagation around a loop, considering no nodes or factors may be “ready” at the start of message passing. For example, in Figure 2.8, before any messages are sent, every node and factor will be blocked waiting for an incoming message. This is solved by initialising all messages with uniform values (e.g., one), which allows message passing to start at any node, regardless of readiness.

Figure 2.8. Messages passed around a loop can continue for ever.
There are many different strategies for the message passing schedule, the choice of which greatly affects the time required to converge. These schedules are typically synchronised to a clock, with messages sent each clock tick. Serial schedules require that only one message is passed each tick, while in the flooding schedule, messages are passed over every edge in both directions each tick. The flooding schedule is usually offline, in that the messages at time $t$ are calculated using the messages at time $t-1$. Serial schedules on the other hand, work with only one set of messages, updating one message at each tick, which then becomes input for subsequent ticks. Serial schedules usually provide faster convergence in terms of total running time, because distant interactions can be propagated over the graph in fewer iterations than with the flooding schedule. However, these serial schedules can also introduce a bias due to their order of message passing.

Typical serial schedules include the pending schedule, which sends messages over an edge which is ‘ready’ to do so; the random schedule, which randomises the schedule at regular intervals during message passing; and tree-based schedules, such as tree-based reparameterisation (Wainwright et al., 2003), which performs exact inference over a set of spanning subtrees within the graph.

### 2.3.4.2 Graph Transformations

While loopy BP allows inference on a cyclic graph, it is an approximation, and furthermore, it provides no theoretical guarantee of convergence or of the fidelity of the approximation. Moreover, it can take many rounds of message passing before converging, particularly so when there are many cycles.

Graph transformations are an alternative exact approach for inference in cyclic graphs, which transform the graph into an acyclic graph. This can be achieved by clustering and/or stretching the nodes (Kschischang et al., 2001). Clustering involves replacing a pair of nodes, $X_i$ and $X_j$ with a single node $(X_i, X_j)$. The factors neighbouring the removed $X_i$ and $X_j$ nodes are reconnected to the new merged node. These new merged nodes have a considerably larger parameter space than the original nodes. If $X_i$ and $X_j$ could assume $I$ and $J$ possible values respectively, the new node will assume $I \times J$ values: their Cartesian product. Clusters of three or more nodes can be formed by repeated clustering. By repeatedly clustering, all the loops can be collapsed together. An example is shown in Figure 2.9, where the original graph in Figure 2.9(a) is transformed by two clustering steps. The first clusters $X_1$ and $X_2$, resulting in the graph in figure Figure 2.9(b). The second step clusters $X_3$ and $X_4$, as shown in figure Figure 2.9(c). Note that the two factors $f_b$ and $f_c$ can now be merged to remove the final loop. This factor merging simply associates the product of the potential functions for $f_a$ and $f_b$ with the merged factor. In such a manner, the transformed graph still preserves the same factorisation as the original.

The stretching transformation replicates a variable all the way around a loop, by adding the variable into each node in the loop. After this process, the original variable
Figure 2.9. Examples of the clustering and stretching graph transformations which convert a cyclic factor graph (Figure 2.9(a)) into a factor tree. Clustering is shown in Figure 2.9(b) and Figure 2.9(c), where nodes $X_1$ and $X_2$; and $X_3$ and $X_4$ are merged, respectively. Figure 2.9(d) shows the stretching transform, where $X_1$ is stretched around the cycle. This leaves a redundant $X_1$ node, which is removed to yield Figure 2.9(e).
node itself will be redundant, and can be removed from the graph, hence removing the loop. This process is shown in Figure 2.9, again starting with the cyclic graph in Figure 2.9(a). The node $X_1$ is replicated around the loop, resulting in figure Figure 2.9(d). The node for $X_1$ can then be removed, and its factor potentials can be evaluated from the new joint $(X_1, X_3)$ and $(X_1, X_2)$ nodes, thus preserving the same factorisation. The final tree is shown in figure Figure 2.9(e).

Both these transformations expand the value-space of the nodes, and where the graph has many cycles, this expansion can prove incredibly costly. The new merged variable nodes range over the Cartesian product of the ranges of the original nodes, which is exponential in the number of nodes merged. For this reason, such graph transformation techniques are intractable in general. The two transforms differ in the manner in which nodes are collapsed together. The clustering method often results in combined factors, as seen in Figure 2.9(c), where the factors $f_b$ and $f_c$ are to be merged, while the stretching method does not. The cost of inference varies with the size of the factor functions, and with the state space at each node. The choice of which (or combinations of both) transforms to use depends on the graph in question and the corresponding cost of inference.

2.4 Conclusion

In this chapter we have seen how graphical models can be used to factorise a joint probability distribution over a set of random variables into the product of functions defined over subsets of variables. Directed and undirected models provide different factorisations and different conditional independence semantics. Each component function in a directed model is a conditional probability distribution and therefore must be appropriately normalised. The component functions for undirected models, however, are only constrained to be positive. In these models normalisation is instead performed globally, via the partition function, $Z$. The largely unconstrained nature of undirected models allows considerable freedom in how the models are parameterised.

Belief propagation is a method for inference in directed and undirected graphical models. This algorithm allows the calculation of marginal probabilities, and the maximum a posteriori realisation, with hidden or partially observed data. While belief propagation theoretically only applies to acyclic graphs, it often provides good approximations when applied to cyclic graphs. Alternatively, graph transformation methods (e.g., clustering and stretching variables) can be used to remove loops from the graph, thus allowing exact inference using standard belief propagation. These two approaches are not silver bullets. They both incur substantial costs in complexity, requiring either arbitrarily many iterations of processing with no convergence guarantee, or else an exponentially increased state space.

This chapter presented the basic mechanics of graphical models: graphical structures,
independence assumptions and the inference process. The next chapter presents Conditional Random Fields, a type of undirected graphical model, and shows how they can be applied to language problems. Many natural language tasks are time series problems in which a stream of text is produced word-by-word over time. Graphical models commonly used in NLP are designed to reflect the sequence nature of language by making a Markov assumption. This leads to simple sparse graphs and therefore efficient inference using belief propagation. Fast inference is vital for graphical models – particularly for Conditional Random Fields, which require repeated sum-product inference for estimation.
Chapter 3

Conditional Random Fields

In this chapter, we describe Conditional Random Fields (CRFs; Lafferty et al. (2001)), undirected graphical models which are particularly well suited to natural language processing (NLP) tasks. CRFs model distributions over structured labellings conditioned on some context. This structure suits many NLP tasks, which require the prediction of complex labellings for spans of text. For example taggers predict a sequence of labels, one for each word in the text, while parsers predict syntax trees or graphs which yield the given text.

CRFs are discriminative models, as they model the conditional distribution over labellings given some contextual observations, \( p(s|o) \), where \( s \) is the labelling and \( o \) is the context. This contrasts with generative models, which model the joint distribution over labellings and the context, \( p(s,o) \). These models are commonly used for decoding test instances where only the context is observed. In this case the maximising labelling of the conditional \( p(s|o) \) is required, \( s^* = \arg\max_s p(s|o) \). Discriminative models can be used directly in this instance, where generative models first require normalisation, \( p(s|o) = \frac{p(s,o)}{\sum_{s'} p(s',o)} \). This is an advantage of discriminative models, which are trained to maximise the conditional likelihood of the training sample. Discriminative models allow a richer feature representation, which provides more natural and accurate modelling. This benefit often comes at the cost of increased training complexity and reduced flexibility with partially observed data. However, for many NLP tasks the advantages of discriminative models outweigh the disadvantages.

CRFs are most commonly used to model sequencing tasks, where the contextual observations are a sequence of tokens, \( o = o_1, o_2, \ldots, o_N \), and the labelling is a sequence of labels of the same length, \( s = s_1, s_2, \ldots, s_N \). This corresponds to labelling each token with a single label, as is the case for most tagging tasks. These sequencing CRFs are often referred to as linear chain CRFs; this refers to the chain graphical structure used to describe Markov assumptions over the label sequence. The name Conditional Random Field denotes the modelling of the labelling, \( S = s \), as a network of inter-dependent random variables (a random field), while conditioning over another set of random variables: the context, \( O = o \).

Before reviewing CRFs, we first present two similar sequencing models, the generative
Chapter 3. Conditional Random Fields

Hidden Markov Model (HMM; Rabiner (1990)), and the discriminative Conditional Markov Model (CMM; Ratnaparkhi (1996)). Both these models provide historical context, from which CRFs may be understood: the CMM was designed to solve perceived problems in the HMM, and the CRF to solve those remaining problems in the CMM. All three models can be considered as weighted finite state automata, where each label has a corresponding state (or many states) and the transitions between states are each weighted by a probability distribution, \( p(s_t|s_{t-1}) \), where \( k \) is the Markov order. We assume a 1:1 mapping between labels and states, and use the term label and state interchangeably. A label sequence can thus be interpreted as a path through the state machine, which in the case of an HMM also generates an output sequence, with one output generated by each state in the path.

Section 3.1 describes the Hidden Markov Model (Rabiner, 1990), a generative directed model, which has had wide success in many labelling tasks (e.g., Brants, 2000). Such generative models make stringent independence assumptions over the output sequence. Take for example POS tagging, where the HMM represents POS tags as states and tokens as outputs. In this task we might like to include features over the token identity, its capitalisation, whether it ends in the suffix -ing, the word’s lemma, the identity of the previous token, token bigrams etc. These features would allow the model to learn a useful bias over the tags, allowing for more accurate modelling. However, these features are not independent and would prove difficult to integrate into a HMM. Non-independent features are very common and modelling the dependencies between such features, as required for use in a HMM, is often very difficult. Moreover, HMMs are generative models which are typically trained to maximise the joint likelihood of the data, \( \prod_{(s,o) \in D} p(s,o) \). Within NLP HMMs are almost exclusively used to decode the maximum conditional likelihood state sequence: the mismatch between the objective at training and testing time is undesirable (Klein and Manning, 2002).

Section 3.2 describes the maximum entropy Conditional Markov Model (CMM), a discriminative model which directly describes the conditional distribution, \( p(s|o) \) (Ratnaparkhi, 1996). The CMM assumes the form of a directed chain and uses a log-linear feature representation. This representation imposes no restriction over the output sequence, instead allowing arbitrary, overlapping and conjunctive features over the outputs. While CMMs often outperform HMMs – due to their more flexible feature representation – they are not without problems. Their directed structure requires that they are locally normalised over each outgoing transition: each state must pass on to its successor states the full probability mass it received. This can lead to undesirable situations where outputs are (partially) ignored during decoding, which is referred to as the label bias problem (Lafferty et al., 2001).

Section 3.3 presents the Conditional Random Field (Lafferty et al., 2001). CRFs have a similar form to the CMM, but instead of using a directed graph, they use an undirected graph. The CRF is globally normalised over the full sequence, in contrast to the CMMs
local normalisation over each transition. Therefore, the label bias problem does not arise for CRFs, which can lead to accuracy improvements (Lafferty et al., 2001; Klein and Manning, 2002).

All three models can be applied to tasks other than sequence labelling, such as jointly tagging a sequence with many layers of annotation (e.g., Ghahramani and Jordan, 1995; Sutton et al., 2004), or even parsing (e.g., Collins, 1999; Johnson et al., 1999; Clark and Curran, 2003, 2004). These tasks typically have large labelling spaces over many variables and complex interdependencies between variables, and therefore inference is very costly or even intractable. For this reason approximation techniques are often required to model such tasks. Despite this problem, using graphical models to perform such complex tasks can result in state-of-the-art accuracy, and these techniques present exciting new avenues for research. CRFs are presented in Section 3.3 for general graphical structures, along with the standard chain.

The remainder of the chapter describes the inference methods used for CRFs and their estimation. As the task of estimating (training) a CRF is often computationally demanding, for chains as well as for richer graphical structures, numerous approximation strategies have been used to reduce this cost. The chapter concludes with a survey of such alternative estimation methods which can reduce the training time.

### 3.1 Hidden Markov Models

**Hidden Markov Models (HMMs)** are generative, directed graphical models, which describe the joint probability over a state (label) sequence and output sequence, \( p(s, o) \). They are doubly stochastic models: the state sequence is ‘generated’ by a stochastic process, from which the outputs are stochastically generated. When given an output sequence, one can not uniquely determine the labelling: many different labellings (state sequences) may have generated the outputs. Instead, inference must be used instead to identify the most likely labelling. The ensuing description follows the excellent tutorial of Rabiner (1990).

When considering HMMs as weighted finite state automata, the ‘generation’ of a set of labels (states) and words (outputs) proceeds as follows: the machine selects a start state, and from this state generates an output. It then transitions to another state, where it generates another output. This continues until it reaches a designated final state. The sequence of emissions forms the output sequence, and the state sequence followed can be interpreted as the labelling.

**Graphical Structure**  Hidden Markov Models are a kind of directed graphical model, as shown in Figure 3.1, which illustrates a first order HMM. Here the state sequence forms a directed chain, with each state, \( s_t \), linked to adjacent states, \( s_{t-1} \) and \( s_{t+1} \), and outputs \( o_t \)
linked only to $s_t$. This encodes a first order Markov assumption over the state sequence, $s_{t+1} \perp s_1, \ldots, s_{t-1} \mid s_t$ and $s_{t-1} \perp s_{t+1}, \ldots, s_N \mid s_t$, i.e., the current state, $s_t$ completely summarises the past ($t' \leq t-1$) states and outputs, for subsequent times ($t' \geq t+1$). The reverse also holds: $s_t$ summarises the effect of all future states and outputs on the past. The HMM also imposes a strong independence condition over the outputs: $o_t \perp o_{t'}, s_{t'} \mid s_t$, where $s_{t'}$ is the state sequence after omitting $s_t$, and correspondingly for $o_{t'}$. This independence-of-outputs assumption is commonly violated in practice.

**Factorisation** The joint probability factorisation for an HMM can be read off the graphical structure, or equivalently from the conditional independence relations. This yields a joint distribution of the form:

$$p(s, o) = p(s_1) \prod_{t=2}^{N} p(s_t \mid s_{t-1}) \prod_{t=1}^{N} p(o_t \mid s_t)$$

(3.1)

Most commonly, the transition distributions $p(s_t \mid s_{t-1})$ are assumed invariant over time, $t$: the HMM is said to be *homogeneous*. These distributions are described by a transition matrix, $A$, with an entry for each state pair. The remaining parameters of an HMM are the starting probability distribution $\pi$, and the output distributions, typically another matrix, $B$, with an entry for each state, output symbol pair.

**Sequence Labelling** Sequence labelling involves finding the most probable state sequence given observed outputs. This state sequence maximises the conditional probability distribution of states given the outputs:

$$s^* = \arg \max_s p(s \mid o) = \arg \max_s \frac{p(s, o)}{p(o)}$$

(3.2)

As the HMM describes the joint probability over states and outputs, we must represent the required conditional in terms of the joint, as shown on the right hand side of (3.2). We can further simplify the equation, by recognising that the output probability on the denominator is independent of the state sequence, and thus can be omitted from the maximisation. This yields

$$s^* = \arg \max_s p(s, o)$$

(3.3)
3.1. Hidden Markov Models

which can be efficiently calculated using dynamic programming, via the Viterbi algorithm (Viterbi, 1967). This algorithm tracks the probability of taking any path through the state machine starting at \( t = 1 \) and ending with \( S_t = s_t \). This probability is characterised in an inductive manner, which leads to an efficient dynamic programming solution. The Viterbi algorithm is a specialisation of the max-product algorithm in Section 2.3.2, where the graphical structure forms a chain.

Model Estimation

The optimal HMM parameters, \( \theta = (\pi, A, B) \), can be found by using a maximum likelihood estimate. This method maximises the probability of a training sample \( D \), which is either partially or fully observed. In other words, it might only consist of the output sequences, while the state sequences remain hidden, or otherwise might consist of both outputs and state sequences.\(^1\) This approach is justified as follows:

\[
\theta_{ML} = \arg\max_{\theta} p(\theta|D) = \arg\max_{\theta} \frac{p(D, \theta)}{p(D)} = \arg\max_{\theta} \frac{p(D|\theta)p(\theta)}{p(D)}
\]

where we note that the denominator of (3.5) is independent of the maximising variable, \( \theta \), and can be omitted. The prior over parameter values, \( p(\theta) \), is assumed to be uniform, and thus the MLE simply maximises \( p(D|\theta) \).

Where the data is fully observed, the values for \( \theta_{ML} \) can be found easily: the parameter for each event is given by the frequency of that event in the training sample. In the case of partially observed data, the maximisation process is more involved, requiring iterative ascent of the log-likelihood, a non-convex function. This process is called expectation-maximisation (EM), or Baum-Welch (Baum et al., 1970). At each iteration, it requires the marginal distributions over each state node, which are provided by the forward-backward algorithm. As with the Viterbi algorithm, this method uses dynamic programming to find the marginals without naively summing over the (exponential) state space. The forward-backward algorithm is a specialisation of the sum-product algorithm in Section 2.3.1, where the graphical structure forms a chain, and is described in Section 3.5.1 for CRFs.

Issues

While the Hidden Markov Model is a powerful method for labelling sequence data, it is not without flaws. It has two major drawbacks: one stemming from its independence assumptions, and the other from its generative nature. The first issue arises from the implicit independence assumed between an output, \( o_t \), and all other outputs in the sequence. This relation prohibits us from easily crafting features over multiple outputs without incorporating their specific dependencies into the graph. For instance,

\(^1\)Other partial observed scenarios are possible, where some, but not all, state sequences are known or the outputs themselves may be partially hidden. These scenarios are rare in natural language processing, and therefore are not considered further.
in a semantic tagging task, we may wish to detect that the phrase *shown the door* has an idiomatic reading (of being fired from a job). If each word in the phrase is assumed independent, then we cannot easily encode such knowledge. Moreover, we are often interested in non-independent features of each output itself. For instance, in POS tagging the word *Accordingly*, we wish to consider not only the word form, but also that the word has an initial capital and ends with the suffix *-ly*. Clearly these supplementary features are dependent on the word form. Incorporating such overlapping non-independent features of the outputs into a HMM often proves difficult.

The second problem faced by HMMs stems from their modelling of the output distribution. Their generative construction means that they not only model the distribution over state sequences, but also that of the outputs, i.e., they are trained to maximise the joint likelihood of the data, rather than the conditional. This conveys a benefit when used in a generative setting, i.e. to generate output sequences or assess the probability of an output sequence, as required for natural language generation. In addition HMMs can be efficiently trained on partially observed data using the Baum-Welch algorithm. However, in the vast majority of NLP settings the outputs sequences are supplied both during training and testing, and therefore the joint distribution defined by the HMM must first be conditionalised before maximisation. Models trained to maximise an objective most similar to the decoding setting produce better results (Klein and Manning, 2002). For HMMs, this means maximising conditional likelihood instead of maximising joint likelihood in training. This change in training objective results in a model which is almost identical to a simple CRF or CMM.

### 3.2 Conditional Markov Models

The *Conditional Markov Model* (CMM) addresses both of the problems faced by the Hidden Markov Model (Ratnaparkhi, 1996). We use the term CMM to describe conditionally trained maximum entropy models, encompassing the work of Ratnaparkhi (1996) and McCallum et al.’s (2001) *Maximum Entropy Markov Model* (MEMM). These works describe the same graphical model, but differ in their sets of permissible features. CMMs are **discriminative** models and model the distribution over states, *given* the outputs, $p(s|o)$. This is exactly the distribution required for decoding, where we find the maximum (posterior) probability state sequence, $\text{argmax}_s p(s|o)$. Further, CMMs employ a rich feature representation which allows the use of arbitrary and overlapping properties of the output sequence. As such, features can easily reference multiple outputs while also covering non-independent properties of each of the outputs.

**Graphical Structure and Factorisation** The CMM takes the graphical form of a directed chain, as shown in Figure 3.2. Each (observed) output is connected to every state,
therefore each transition distribution between adjacent states has access to the complete set of outputs. The outputs are part of the conditioning context and therefore the edges between these nodes and state nodes do not complicate inference which is instead performed efficiently over the hidden chain of states.

To perform POS tagging with a CMM we treat the words as the observations while the POS tags give the state sequence. Within this setup the model learns how to build a sequence of POS tags from left to right one tag at a time. Each decision is guided by the sequence of words and the most recent chosen tag(s).

The graphical structure in Figure 3.2 yields the following factorisation:

$$p(s|o) = \prod_{t=1}^{N} p(s_t|s_{t-1}, o)$$  \hspace{1cm} (3.7)$$

where we assume $s_0$ is fixed at a special start state, such that $p(s_1|s_0 = \text{start}, o) = p(s_1|o)$. The CMM is homogeneous: the same component conditional distribution, $p(s_t|s_{t-1}, o)$, is used for each time, $t$. This distribution has a fixed set of features, each referencing parts of $o$ possibly about $t$. The features combine the observational tests with functions over $s_t$ or $s_{t-1}$ and $s_t$. In this manner, these features allow the model a flexible form of context-dependence.

**Feature Representation** The transition distribution, $p(s_t|s_{t-1}, o)$, is parameterised according to the maximum entropy principle, leading to a log-linear distribution. Note that the CMM cannot be described as a log-linear model over the full distribution, $p(s|o)$, but rather only over the component transition distributions. This title is reserved for Conditional Random Fields (CRFs), which are described in Section 3.3.

A full discussion of the principle is deferred until Section 3.3.2, where it is presented in the context of CRFs. In short, we start with a set of features which are functions ‘detecting’ aspects of the observation sequence coupled with a given label, $s_t$, or label pair, $(s_{t-1}, s_t)$. For instance, a feature which detects whether a word (output) ends in ly and is assigned

---

2In many other CRF and CMM publications, the illustrations shown each output connected only to its corresponding state – not to every state (McCallum et al., 2001; Lafferty et al., 2001, e.g.). This condition is overly restrictive, and is purely used in order to keep the diagrams uncluttered.
the tag RB, has the form:

\[ f_{22}(s_{t-1}, s_t, o, t) = \llbracket o_t \text{ ends with } 1y \rrbracket \cdot \llbracket s_t = \text{RB} \rrbracket \] (3.8)

where the subscript is the arbitrary feature index. This feature combines an observation test and a labelling test, where \( \llbracket \cdot \rrbracket \) denotes the indicator function which returns one if the enclosed statement is true, and zero otherwise.

We count how many times each feature is activated in a training set, which we call the observed or empirical counts. We seek to find a distribution with expected feature counts equal to these empirical counts. As there may be many such distributions, we choose the distribution which is also closest to the uniform: the one with maximum entropy.

The maximum entropy principle leads to a parameterisation of the form:

\[ p(s_t|s_{t-1}, o) = \frac{1}{Z(s_{t-1}, o, t)} \exp \sum_k \lambda_k f_k(s_{t-1}, s_t, o, t) \] (3.9)

where \( \lambda_k \) are the model’s parameters, \( f_k \) are feature functions, and \( Z \) is the partition function, which ensures that \( p \) is appropriately normalised. It is defined as follows:

\[ Z(s_{t-1}, o, t) \triangleq \sum_{s_t} \exp \sum_k \lambda_k f_k(s_{t-1}, s_t, o, t) \] (3.10)

Each feature function, \( f_k \) must produce a real number, but is otherwise unconstrained.\(^3\) Therefore features are not constrained to activate only in a single conditioning context – for a given predecessor state, \( s_{t-1} \) – as required in McCallum et al.’s (2001) MEMM. For instance, the feature in (3.8) will activate for all state pairs ending with RB, as it ignores the value of \( s_{t-1} \). McCallum et al.’s condition is overly restrictive in comparison to the earlier approach of Ratnaparkhi (1996).\(^4\)

Sequence Labelling  The CMM is used for sequence labelling in much the same way as the Hidden Markov Model. The factorisation in (3.7) has a similar structure to that of the HMM in (3.1): both models have a hidden structure forming a directed chain. Therefore the Viterbi algorithm can again be used for decoding, as described in Section 3.5.2.

Model Estimation  Compared to HMMs, the training (or estimation) of a CMM requires fully observed training data, \( D = \{(s, o)\} \), consisting of pairs of state sequences and observation sequences, respectively. When these (state sequence, observation sequence) pairs are independently and identically distributed (iid), the probability of the data is given by:

\[ p(D|\Lambda) = \prod_{s, o} \left( \prod_{t=1}^N p(s_t|s_{t-1}, o, \Lambda) \right)^{p(s, o)} \] (3.11)

\(^3\)When using GIS or IIS for training the feature functions must be non-negative, however gradient-based methods do not impose this constraint.

\(^4\)McCallum et al.’s (2001) feature restriction does simplify training, allowing the log-likelihood objective function in (3.12) to be decomposed into parts which can each be independently optimised. However this saving comes at the cost of making poorer use of sparse data.
where $\tilde{p}(s, o)$ is the empirical distribution over the training set, i.e., $\tilde{p}(s, o) = \frac{\text{count}(s, o)}{|D|}$, and the dependence of the transition distribution on the parameters, $\Lambda = \{\lambda_k\}$, is made explicit.

It is more convenient to work with the logarithm of (3.11), called the log-likelihood. This is given below:

$$L(D, \Lambda) = \sum_{s, o} \tilde{p}(s, o) \sum_{t=1}^{N} \left( \sum_{k} \lambda_k f_k(s_{t-1}, s_t, o, t) - \log Z(s_{t-1}, o, t, \Lambda) \right)$$  \hspace{1cm} (3.12)

The parameters, $\Lambda$, which maximise this objective function, $L$, cannot generally be found in closed form. This objective is a convex function and therefore iterative numerical optimisation techniques may be used to find the optimal parameters. Such techniques include generalised iterative scaling (GIS), improved iterative scaling (IIS), conjugate gradient and quasi-Newton techniques, such as L-BFGS (Darroch and Ratcliff, 1972; Della Pietra et al., 1997; Nocedal and Wright, 1999; Nocedal, 1980). This optimisation process is common to all log-linear models, and is described in more detail in the context of Conditional Random Fields in Section 3.6.1.

The Label Bias Problem As a consequence of their directed parameterisation, and thus local normalisation, CMMs suffer from the label bias problem (Lafferty et al., 2001; Bottou, 1991). In its extreme form, this bias can lead to outputs being ignored or else previous states being ignored. This effect has been demonstrated in Johnson (2001) and Klein and Manning (2002), where conditional model structures led to accuracy degradation in shift-reduce parsing and part-of-speech tagging. This effect was offset by increases in accuracy due to conditional estimation – i.e., maximising the objective being evaluated – therefore explaining why conditional models do not consistently outperform their generative counterparts. This phenomena is best presented in the form of an example. Assume we wish to model the part-of-speech tagging of the following sentences:

1. *TelstraNNP sharesNNS roseVBD sharplyRB .*
2. *TelstraNNP sharesVBZ itsPRP$ successNN .*

The finite-state acceptor (FSA) for these sentences is shown in Figure 3.3, where the transitions between states are annotated with *token:tag* input/emission pairs. A CMM modelling the data can be considered as such an FSA, supplemented with edge weights reflecting the probability of each transition.

Now consider the act of processing sentence (2). The FSA will first transition from state 0 to 1, accepting *Telstra* and emitting *NNP*. Now the machine can transition to either of states 2 or 6, as both transitions share the same input *shares*. Both transitions are followed, each taking a portion of the probability mass from state 1. If we now consider the top path, it is clear that no outgoing transitions from state 2 will accept our input *its*. However, as there is only one outgoing transition (to state 3), and each transition function
is a probability distribution and thus sums to one, the lone transition must be taken, and the full probability mass must be transferred to state 3. Thus the token its was completely ignored from consideration. Once both paths meet at the final state 5, their probabilities will only differ by one term: the allocation of probability mass over transitions 1 → 2 and 1 → 6. The analysis of the sentence was thus entirely contingent on the next-state distribution at state 1, and not the full sequence of tokens. This could easily lead to the mistagging of the sentence.

The example described an extreme case, where some states had only one outgoing transition. The problem still arises, albeit in a less extreme form, whenever there is a low-entropy transition distribution: a common situation in practice. Such distributions have relatively few possible next states, and can again ignore or dampen the influence of an observed token on the transition (Lafferty et al., 2001). An MEMM (McCallum et al., 2001) would be expected to suffer more markedly from label bias than a general CMM (Ratnaparkhi, 1996), as the MEMM cannot include single label features. These features allow the model to model the state distribution for rare histories, rather than relying on sparse counts. The net result is lower entropy transition distributions in the MEMM, and therefore an increased prevalence of label bias.

The label bias problem does not occur in HMMs because they are not required to preserve the probability mass over each transition in the FSA: the observation probability at each state can dampen the path probability, and therefore avoid improbable states. For an HMM applied to the example in Figure 3.3, the probability of generating the token its from state VBD will be zero, and therefore the upper path will have a zero probability. This demonstrates how the HMM is unaffected by label bias.

### 3.3 Conditional Random Fields

In order to address the label bias problem in the CMM, Lafferty et al. (2001) proposed the **Conditional Random Field** (CRF). These models drop the local normalisation constraint,
3.3. Conditional Random Fields

and instead normalise *globally* over the full graph. Therefore each state needn’t preserve the probability mass over its outgoing transitions and thus the label bias problem does not occur. Instead each transition is given a positive score and the score for a path is given by the product of its transition scores. The path probability can then be calculated by normalising the path score by the sum of scores over all possible paths. A path which does not match the outputs will receive a very low score (probability), while a well-matched path will score highly. Therefore the label bias problem does not occur in CRFs. Lafferty et al. (2001) showed how CRFs significantly outperformed CMMs on a synthetic task designed to highlight the label bias problem, and on the real-word task of part-of-speech tagging. This bias was also demonstrated by Klein and Manning (2002) for conditionally trained HMMs, which are equivalent to simple CRFs, and possibly also for conditional models for parsing (Johnson, 2001). This section follows Lafferty et al.’s (2001) presentation.

CRFs retain a similar structure to that of an CMM. They are discriminative, directly modelling the conditional distribution $p(s|o)$. Moreover, they use the same feature representation, and therefore also describe a log-linear distribution. Consequently CRFs share the same benefits of the CMM over generative models such as the HMM: they allow for arbitrary, overlapping and conjunctive features over the output sequence, while also directly modelling the required conditional distribution.

### 3.3.1 Graphical Structure and Factorisation

A CRF is an undirected graph with two parallel chains of random variables, where one layer describes the state sequence, $s$, and the other describes the output sequence, $o$. The output sequence is always fully observed, and each output is connected to every state. A first order CRF is shown in Figure 3.4, where the state sequence forms a chain.

The probability distribution described by a CRF is that of an undirected model, as described in Section 2.2: a globally normalised product of clique potentials. Recall that a clique is a fully connected subset of nodes. In the case of the first order chain in Figure 3.4, each node and each adjacent pair of nodes form the cliques, i.e., $C =$

![Figure 3.4. A first order conditional random field. The top layer shows the state sequence, while the bottom layer shows the output sequence. The grey shading indicates that the output nodes are observed. The dotted lines summarise many outgoing edges which connect the output to every other state.](image-url)
Chapter 3. Conditional Random Fields

\[ \{S_t\}_{t=1}^N \cup \{(S_{t-1}, S_t)\}_{t=2}^N \]. The clique potential functions, \( \psi(\cdot) \), map the labelling for a given clique (the state configuration) to a strictly positive real value, but are otherwise arbitrary. The factorisation is given below:

\[
p(s|o) = \frac{1}{Z(o)} \prod_{c \in C} \psi_c(s_c, o) \quad (3.13)
\]

where \( Z(o) \triangleq \sum_s \prod_{c \in C} \psi_c(s_c, o) \) is the globally normalising partition function. Note that for CRFs this normalising term is now a function of the outputs, \( o \), whereas for unconditional (generative) undirected models it is simply a constant (in (2.4)). This reflects the conditional behaviour of the CRF: the potential functions are allowed unfettered access to the observed output sequence and therefore normalisation must be performed independently for each different output sequence.

As with other undirected models the potential functions, \( \phi \), have an exponential form. This representation explicitly represents the positivity constraints on the potentials, \( \psi \), in terms of the unconstrained log-potentials, \( \phi \), i.e., \( \psi_c(s_c, o) = \exp \phi_c(s_c, o) \). This equivalent representation is given by:

\[
p(s|o) = \frac{1}{Z(o)} \exp \sum_{c \in C} \phi_c(s_c, o) \quad (3.14)
\]

with the corresponding definition for the partition function,

\[
Z(o) \triangleq \sum_s \exp \sum_{c \in C} \phi_c(s_c, o) \quad (3.15)
\]

The form of (3.14) closely resembles the form of the undirected graphical model in Section 2.2, (2.8). A CRF is simply a conditional form of undirected model, where the conditioned nodes affect the log-potential functions, \( \phi \), and therefore also the partition function, \( Z(o) \).

### 3.3.2 Features and the Maximum Entropy Principle

As with the CMM, the CRF is parameterised according to the maximum entropy principle. This method allows us to find the distribution which fits all of our observations, in the forms of feature counts. For instance, if we observe in the training sample that the words “put up” are often POS tagged as verb-particle, VBN and RP, we can encode this pattern with a simple indicator feature:

\[
f_{38}(s_c, o, c) = [o_c = \text{(put, up)}] \cdot [s_c = \text{(VBN, RP)}] \quad (3.16)
\]

where \([\cdot]\) is the indicator function, as described in Section 3.3.2.

This feature is defined over pair-wise cliques, \( c = (t-1, t) \), states, \( s_c = (s_{t-1}, s_t) \), and the observed outputs, \( o \). In fitting a model to our training sample, we desire that if \( f_{38} \)
was active $k$ times on the labelled sample, then our model should also predict that same feature count, $k$. Formally, this constraint is given by:

$$\sum_{s, o} \tilde{p}(s, o) \sum_{c} f_{38}(s_c, o, c) = \sum_{o} \tilde{p}(o) \sum_{s} p(s|o) \left( \sum_{c} f_{38}(s_c, o, c) \right)$$  \hspace{1cm} (3.17)

where $\tilde{p}(o)$ is the empirical distribution over the output sequences in the training data, and $\tilde{p}(s, o)$ is the joint empirical distribution over both state and output sequences.

Features are not constrained to only binary indicator functions, but instead may be any real valued function, $f_k(s_c, o, c) \in \mathbb{R}$. For example, a feature might detect the length of a word, or a calculate the score from another model. The features must discriminate between the different state configurations, $s_c$; otherwise they have no predictive power. Moreover, the features are constrained to consider only state configurations over cliques within the graph. This stems from the Markov assumption inherent in the CRF: clearly, defining features over larger subsets of states would require a more richly connected state sequence, with correspondingly larger cliques.

(3.17) encodes a constraint for a single feature: that its expected count should equal its observed count on the training data. This can be presented more generally for all features, $f_k$, as:

$$\sum_{s, o} \tilde{p}(o)p(s|o)F_k(s, o) = \sum_{s, o} \tilde{p}(s, o)F_k(s, o)$$  \hspace{1cm} (3.18)

$$E_{\tilde{p}(o)p(s|o)}[F_k] = E_{\tilde{p}(s, o)}[F_k]$$  \hspace{1cm} (3.19)

where $F_k(s, o) = \sum_{c \in C} f_k(s_c, o, c)$ are the aggregated features over the sequence.

Armed with the constraints for each feature in (3.19), we seek a conditional distribution $p(s|o)$ which satisfies these constraints. However, there may be many such distributions and ideally we would like a criterion to select the best among them. We discriminate between such distributions using the principle of maximum entropy: we choose the distribution which makes no additional assumptions about the data beyond our supplied constraints. This bears a resemblance to Occam’s Razor, which states that no more assumptions should be made than are strictly required.

Entropy is an information theoretic measure of the amount of uncertainty in a distribution. A distribution which is strongly biased towards certain outcomes has low entropy, while a uniform distribution has high entropy. Strongly biased distributions are highly informative in that they differ from the default baseline of uniformity. Maximising entropy ensures that the model is as uniform as possible (while fitting the constraints). Conditional entropy measured in bits is given by:

$$H(S|O) = -\sum_{o} \tilde{p}(o) \sum_{s} p(s|o) \log_2 p(s|o)$$  \hspace{1cm} (3.20)

We drop the base of the logarithm and instead use the natural logarithm. This scales the entropy by an constant, which is irrelevant for maximisation.
We maximise the conditional entropy, $H(S|O)$ under the constraints in (3.19), and also with the following additional constraint which ensures correct normalisation:

$$\sum_s p(s|o) = 1 \quad \forall o$$  \hspace{1cm} (3.21)

We model this constrained optimisation problem using a Lagrangian, with Lagrange multipliers, $\lambda_k$, for each feature constraint (3.19), and $\mu_o$ for the normalisation of each output sequence (3.19). This gives us the following:

$$L = -\sum_o \tilde{p}(o) \sum_s p(s|o) \log p(s|o)$$

$$- \sum_k \lambda_k \left( \sum_{o,s} \tilde{p}(s,o)F_k(s,o) - \sum_o \tilde{p}(o) \sum_s p(s|o)F_k(s,o) \right)$$

$$- \sum_o \mu_o \tilde{p}(o) \left( \sum_s p(s|o) - 1 \right)$$  \hspace{1cm} (3.22)

which we optimise with respect to $p(s|o)$, by taking the partial derivatives:

$$\frac{\partial L}{\partial p(s|o)} = -\tilde{p}(o) (1 + \log p(s|o)) + \sum_k \tilde{p}(o) \lambda_k F_k(s,o) - \mu_o \tilde{p}(o)$$  \hspace{1cm} (3.23)

which we set to zero. After cancelling the factor $\tilde{p}(o)$, and rearranging, this yields the general form of $p(s|o)$:

$$p(s|o) = \exp(-1 - \mu_o) \exp \left( \sum_k \lambda_k F_k(s,o) \right)$$  \hspace{1cm} (3.24)

Using the normalisation constraint in (3.21), the first exponential term can be re-written as:

$$Z(o) \triangleq \exp(1 + \mu_o) = \sum_s \exp \left( \sum_k \lambda_k F_k(s,o) \right)$$  \hspace{1cm} (3.25)

which we refer to as the partition function, $Z$. This has the form of an exponential family, with sufficient statistics $F_k$, and canonical parameters $\Lambda = \{\lambda_k\}$, given by:

$$p(s|o) = \frac{1}{Z(o)} \exp \left( \sum_k \lambda_k F_k(s,o) \right)$$  \hspace{1cm} (3.26)

The above form of distribution also has the pleasing property that its maximum likelihood estimate (MLE) satisfies the original constraints in (3.19). Recall that the maximum likelihood estimate (MLE) is the parameter configuration which maximises the likelihood of a training sample, $\mathcal{D}$, i.e., $\Lambda^{\text{ML}} = \arg \max_{\Lambda} p(\mathcal{D} | \Lambda)$. The derivation of the MLE is deferred until Section 3.6.1.

### 3.3.3 Potentials

In the previous section, we showed how the maximum entropy principle could be used to define an exponential form for the CRF’s conditional distribution, $p(s|o)$. However, we
still must relate this to the distribution’s factorisation in terms of products of potential functions, defined by its graphical structure in (3.14). When we expand the per-graph aggregated feature functions, $F_k$ in (3.26), we obtain:

$$p(s|o) = \frac{1}{Z(o)} \exp \left( \sum_{c \in C} \sum_k \lambda_k f_k(s_c, o, c) \right)$$

(3.27)

whereas earlier, $Z(o)$ sums out the numerator for every state sequence. This representation can be unified with (3.14), yielding the following definition for the log-potentials:

$$\phi_c(s_c, o) = \sum_k \lambda_k f_k(s_c, o, c)$$

(3.28)

If only considering maximal cliques, the potentials of each non-maximal clique must be factored into exactly one of its subsuming maximal cliques, as discussed in Section 2.2. This re-arrangement still describes the exact same distribution but may simplify computation.

### 3.4 Alternative Graphical Structures

Until now we have considered only sequence labelling tasks, where each output is given a single label, and using a first order Markov assumption between labels at adjacent sites. This corresponds the linear chain CRF graphical structure. Graphical models can be used to model other labelling tasks where the interdependencies (and lack thereof) are encoded in a graph. CRFs were first applied to large interconnected networks by Taskar et al. (2002) (under the name relational Markov networks). An undirected graphical model was used for collective web-page classification by representing page labels as hidden variables and hyper-links as edges connecting the page variables. This structure is illustrated in Figure 3.5, showing a network of 10 web-pages where each is labelled for topic. Pages on the same topic tend to be more inter-connected and thus form tight clusters in the graph.

A second application of CRFs to large graphical networks was the factorial labelling of Sutton et al. (2004), under the moniker dynamic CRFs (DCRFs). Their tagging task requires a sequence of outputs to be labelled with a number of different labels for each output, where each type of label is connected to form an undirected chain. These chains are interconnected, such that interdependencies between their respective states can be readily modelled. An example DCRF structure is illustrated in Figure 3.6, describing the task of jointly tagging a sequence of tokens for part-of-speech (POS) and syntactic chunks. This models the probability of $p(c, s|o)$, where $c$ is the chunk sequence, $s$ is the POS sequence and $o$ are the observed tokens. In this way tagging decisions in one chain can affect the other, and therefore the resultant joint tagging will produce the best compromise, or mutually consistent, overall tagging. This contrasts with the more typical cascade model for NLP, where each task is performed in order, and the prediction from one model becomes input
Figure 3.5. Relational Markov Network (RMN) for web-page classification. Nodes correspond to web-pages, and the edges mirror hyper-links between pages. The shading of each node denotes the topic label for its web-page.

Figure 3.6. Example factorial dynamic Conditional Random Field (DCRF). This models the joint task of POS tagging and chunking, where each token is given a label for each. All observed node are connected to all hidden nodes, as indicated by the dashed lines.

for the next. Errors in early stages propagate down the cascade, resulting in profound errors in the final predictions. In contrast, the joint model allows decisions at each stage to interact with decisions at earlier and later states, thereby reducing error propagation. In general, joint inference better copes with interacting decision processes, and therefore outperforms piecewise inference. This is observed in the accuracy improvements of CRFs over CMMs as well as factorial DCRFs over cascades of independent models.

The skip-chain CRF is another non-chain based CRF (Sutton and McCallum, 2004). These models are used for named-entity tagging, modelling the interdependence between the tagging of repeated instances of the same word in a document. These repeats should usually receive a homogeneous tagging. The skip-chain models each sentence in a document with a separate chain, and adds additional links between the nodes for repeated (capitalised) words. In this manner the tagging of the words in context can inform the other instances in the other chains, and therefore arrive upon a mutually consistent tagging.

While these structures are very exciting and present new and powerful ways of solving
3.5 Inference in Conditional Random Fields

For both linear chain CRFs and the richer RMN or DCRF models, we require mechanisms for inference. These operations are required for training and decoding. As with the HMM and CMM, the CRF uses belief propagation for inference. For chains, as used in sequence labelling, specialised variants of the sum-product and max-product algorithms are used: the forward-backward and Viterbi algorithms, respectively. While the process of marginalisation and maximisation can be understood using belief propagation (see Section 2.3), the simpler chain-based variants are given here for completeness, following Lafferty et al. (2001).

3.5.1 Marginalisation: The Forward-Backward Algorithm

The forward-backward algorithm is used to find the marginal probability distributions over single states, $p(s_t|o)$, and pairs of states (the maximal cliques), $p(s_t, s_{t+1}|o)$. This technique can also be used to find the probability of a full state sequence, $p(s|o)$, or equivalently, the partition function, $Z(o)$.

The forward values, $\alpha_t(s_t)$, are the sum of the unnormalised probabilities (or ‘scores’) for all partial paths starting at $t = 1$ and converging at $S_t = s_t$ at time $t$.

$$\alpha_t(s_t) = \sum_{s_1, s_2, \ldots, s_{t-1}} \prod_{t'=1}^{t-1} \psi_{t', t+1}(s_{t'}, t, o) \tag{3.29}$$

where for simplicity of notation, we have assumed that we are only dealing with maximal cliques: for the chain structures under consideration, there is a maximal clique for every pair of adjacent times, $(t, t+1)$. The forward values can be calculated efficiently by the following recursion, thereby avoiding the $t-1$ summations in (3.29):

$$\alpha_1(s_1) = 1 \tag{3.30}$$

$$\alpha_t(s_t) = \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1}) \psi_{t-1, t}(s_{t-1}, t, o) \tag{3.31}$$

where each $\alpha_t$ is a vector quantity over the label set at $S_t$. (3.30) forms the base case for the recursion, from which (3.31) can be calculated for $t = 2, 3, \ldots, n$. The final forward values can be summed to find the partition function, $Z(o) = \sum_{s_N} \alpha_N(s_N)$.
The backward values, $\beta_t(s_t)$, similarly define the sum of unnormalised probabilities for all partial paths starting at time $t$ with state $s_t$ and continuing until the end of the sequence, $t = n$. These values are also defined by a recursion:

\[
\begin{align*}
\beta_N(s_N) &= 1 \quad (3.32) \\
\beta_t(s_t) &= \sum_{s_{t+1}} \beta_{t+1}(s_{t+1})\psi_{t,t+1}(s_{t,t+1}, o) \quad (3.33)
\end{align*}
\]

Together the forward and backward values can be used to find the marginal distributions over single states, and pairs of states, as shown below:

\[
\begin{align*}
p(s_t) &= \frac{\alpha_t(s_t)\beta_t(s_t)}{Z(o)} \quad (3.34) \\
p(s_t, s_{t+1}) &= \frac{\alpha_t(s_t)\psi_{t,t+1}(s_{t,t+1}, o)\beta_{t+1}(s_{t+1})}{Z(o)} \quad (3.35)
\end{align*}
\]

### 3.5.2 Decoding: The Viterbi Algorithm

The Viterbi algorithm is used for finding the maximum probability state sequence, $s^* = \arg\max_s p(s|o)$. This algorithm roughly mimics the forward pass of the forward-backward algorithm, but where the forward algorithm sums out the effect of previous state configurations, Viterbi maximises. The Viterbi algorithm can be defined with the following recursion:

\[
\begin{align*}
v_1(s_1) &= 1 \quad (3.36) \\
v_t(s_t) &= \max_{s_{t-1}} v_{t-1}(s_{t-1})\psi_{t-1,t}(s_{t-1,t}, o) \quad (3.37)
\end{align*}
\]

After calculating $v_t$ for $t = 2, \ldots, n$, the maximising final state can be read from $s_N^* = \arg\max_{s_N} v_N(s_N)$. The remaining states in the maximising configuration can be found by constructing a set of back-pointers, $b_t(s_t)$, which index for each $s_t$ the best previous state: the state, $s_{t-1}$, which won the maximisation in (3.37). These back-pointers are defined analogously to the forward-recursion:

\[
\begin{align*}
b_t(s_t) &= \arg\max_{s_{t-1}} v_{t-1}(s_{t-1})\psi_{t-1,t}(s_{t-1,t}, o) \quad (3.38)
\end{align*}
\]

from which the second-to-last state can be found, $s_{n-1}^* = b_N(s_N^*)$, and so on back to $s_1^*$, at which point we have a full maximising state sequence, $s^*$.

In practical terms, such maximisation is often performed in log-space, such that the multiplications in (3.37) and (3.38) become additions and the potentials, $\psi$, are replaced with the log-potentials, $\phi$. This helps to avoid numerical over- and under-flow.

### 3.6 Estimation of Conditional Random Fields

Now that we have seen how CRFs can be used for sequence labelling, we turn to the task of estimation, or training. This task requires fitting the model to some training data $D$. 

We assume that the training data is fully observed, i.e., $D = \{(s, o)\}$, and that each state sequence, output sequence pair is independently and identically distributed (iid).

The CRF’s parameters, $\Lambda^*$, are chosen to maximise the following:

$$
\Lambda^* = \arg \max_\Lambda p(\Lambda | D) \\
= \arg \max_\Lambda p(D | \Lambda) p(\Lambda)
$$

in the same manner as the HMM, in (3.6). We describe two scenarios for maximising the quantity in (3.40): maximum likelihood, where the prior distribution, $p(\Lambda)$, is assumed uniform; and maximum a posteriori, where the prior distribution is given.

### 3.6.1 Maximum Likelihood

The conditional probability of the training sample given parameters $\Lambda$ is as follows:

$$
p(D | \Lambda) = \prod_{s, o} p(s | o, \Lambda) \tilde{p}(s, o)
$$

where the dependence of $p(s | o)$ on the parameters, $\Lambda$, is now being shown explicitly.

The maximum likelihood estimate (MLE) seeks to optimise this function, finding the parameter configuration, $\Lambda$, which best models the data. We optimise the logarithm of this quantity rather than work with it directly: because the logarithm is monotonic the two functions will have the same maximising argument, $\Lambda$, while in the logarithmic form the derivatives are more manageable. The log-likelihood of the sample is:

$$
\mathcal{L} = \sum_o \tilde{p}(o) \left( \sum_s \tilde{p}(s | o) \sum_{c \in C} \sum_k \lambda_k f_k(s_c, o, c) - \log Z(o) \right)
$$

The maximum (log-)likelihood is achieved when the partial derivatives of $\mathcal{L}$ with respect to each parameter, $\lambda_k$, are zero. These partial derivatives are given by:

$$
\frac{\partial \mathcal{L}}{\partial \lambda_k} = \sum_{o, s} \tilde{p}(s, o) \sum_{c \in C} f_k(s_c, o, c) - \sum_o \tilde{p}(o) \sum_s p(s | o) \sum_{c \in C} f_k(s_c, o, c) \\
= E_{\tilde{p}(s, o)}[F_k] - E_{\tilde{p}(o)p(s|o)}[F_k]
$$

where $F_k$ is the same shorthand for the aggregated feature counts introduced earlier in (3.19). Setting these derivatives to zero yields the feature constraints in (3.19), confirming our earlier statement of duality between the maximum entropy formulation and the MLE in Section 3.3.2.

The parameters which maximise the log-likelihood in (3.42) cannot, in general, be found in closed form. Instead we rely on numerical optimisation techniques to find the parameters. The log-likelihood is a convex function and therefore greedy hill-climbing methods will find the global optimum. A variety of such numerical techniques have been used to date, including Improved Iterative Scaling (IIS), Generalised Iterative Scaling (GIS) and gradient based techniques (Darroch and Ratcliff, 1972; Berger et al., 1996; Nocedal...
Chapter 3. Conditional Random Fields

Malouf (2002) showed that general gradient based techniques outperformed the traditional GIS and IIS for estimating maximum entropy classifiers. In particular, L-BFGS (Nocedal, 1980), a quasi-Newton method, was shown to be superior in terms of the quality of the solution and the time required to train compared to IIS, GIS and other gradient based techniques. Sha and Pereira (2003) also presented similar findings for CRFs, as did Wallach (2002), both for shallow parsing.

In recent work Vishwanathan et al. (2006) showed how Stochastic Meta-Descent (SMD) could reduce training time by an order of magnitude compared to L-BFGS. This method uses an online update and scales sub-linearly with the amount of training data (due to redundancy in the data). The experiments in this thesis use L-BFGS optimisation although we expect that our accuracy and training time comparisons would be mostly invariant to the optimisation method.

L-BFGS is a limited-memory, quasi-Newton method for large scale optimisation (Nocedal, 1980). It optimises an objective function in an iterative manner, at each iteration using the function value and its first and second order partial derivatives to chose the next step direction. The Hessian matrix of second order partial derivatives is approximated from the previous function evaluations and their corresponding first order derivatives. Therefore the Hessian need not be explicitly calculated which is ideal given this calculation would be intractable for the CRF’s log-likelihood. However, the use of second order derivatives (curvature) allows for faster maximisation, with better informed steps than available simply using first order derivatives. L-BFGS limits the memory requirements by only storing a finite number of these past evaluations. In practical terms, a history of the last 5 to 10 evaluations is sufficient, although an even larger history can further reduce the number of iterations required before convergence.

In CRF estimation the L-BFGS computation time and memory requirements (to calculate the Hessian approximation and step direction) are often dwarfed by the cost of evaluating the log-likelihood objective at each iteration: the log-likelihood in (3.42) and its first order partial derivatives in (3.44) must be calculated for each parameter, \( \lambda_k \). Each derivative is the difference between the observed and expected values for feature \( k \). The observed values can be quickly and easily computed, as these are simply counts of the feature’s activation on the labelled training set. The expected values pose more difficulties, requiring for each training instance an expectation over all possible labellings, \( \sum_s p(s|o) \sum_{c \in C} f_k(s_c, o, c) \). Thankfully this expectation can be calculated without summing over the exponential set of configurations, \( s \). The expectation can be rewritten as:

\[
\sum_s p(s|o) \sum_{c \in C} f_k(s_c, o, c) = \sum_c \sum_s p(s_c|o) f_k(s_c, o, c) \tag{3.45}
\]

where the distribution, \( p(s_c|o) \) is the marginal distribution over clique \( c \), obtained from the sum-product algorithm in (2.20). For forward-backward inference over chains, as de-

\[\textit{Although Altun et al. (2004) present an approximation to the Hessian allowing second-order methods to be used to estimate CRFs.}\]
scribed above, these marginals are given in (3.34) and (3.35). The partition function, \( Z(o) \), which is required in the log-likelihood, is a by-product of forward-backward inference.

### 3.6.2 Maximum a Posteriori

An alternative estimation method to the maximum likelihood estimation (MLE) is that of maximum a posteriori (MAP) estimation. While the former maximises (3.40) under the assumption of a uniform prior, the latter maximises the same equation without this assumption. The prior, \( p(\Lambda) \), encodes some prior knowledge that we might have about the distribution over parameter values.

Typically we know very little about the parameter values. Instead the prior is used for smoothing (regularisation), such that over-fitting of the training sample is minimised and therefore generalisation accuracy is not compromised. Typical priors, such as the Gaussian prior, cluster parameters about zero. When a parameter, \( \lambda_k \), is zero the corresponding feature, \( f_k \), is ignored. When all the parameters are zero the distribution \( p(s|o) \) is uniform. Making an a priori assumption of uniformity is a reasonable one and is consistent in nature with the maximum entropy principle, which also maximises uniformity.

Typically the prior not only encodes the parameter expectations of zero, but also states that few parameters should have high magnitude. This effect discourages single features from being relied upon too heavily, by receiving a large magnitude weight. Instead, this weight is spread more evenly between a number of lower quality features. This aids in a decoding scenario where the highly predictive feature might be absent, but some of the less predictive features are still present. Here the model will still be able to make a confident prediction using their combined weights. This also has the effect of limiting the ability to (over-)fit idiosyncrasies of the training sample. This discourages heavy reliance on features which only occur a few times in the training sample and therefore are not likely to constitute a representative sample.

The maximum entropy principle prevents the model from being overly specific beyond what is necessary to describe the training set. If given a small set of clean and highly discriminatory features, then a maximum entropy distribution should generalise well to unseen data. However, in common usage maximum entropy models are provided with thousands or millions of features, many of which are noisy, redundant or otherwise largely uninformative. The model can use these myriad of features to fit the training sample extremely precisely. Such over-fitting can seriously compromise accuracy, which motivates punitive regularisation with a tight prior distribution.

A prior distribution should exhibit a regularising influence, penalising high magnitude parameter values. Three such distributions have been used to date: the Gaussian, Laplacian, and Hyperbolic (Chen and Rosenfeld, 1999; Sha and Pereira, 2003; Peng and McCallum, 2004). These priors all produce roughly similar regularisation effects; therefore the Gaussian, the simplest of the three, is most commonly used. For these same
reasons, in this thesis we will follow standard practice and use a Gaussian prior. The Gaussian probability density function is:

\[
p(\lambda_k) = \frac{1}{\sigma_k \sqrt{2\pi}} \exp\left(-\frac{(\lambda_k - \mu_k)^2}{2\sigma_k^2}\right)
\]

where \( \mu_k \) is the mean for parameter \( \lambda_k \), and \( \sigma_k \) the standard deviation. This can be incorporated into the earlier MLE log-likelihood in (3.42), using the likelihood equation in (3.40), which yields the MAP log-likelihood, after excluding constant terms:

\[
L^{MAP} = L - \frac{1}{2} \sum_k \left(\frac{\lambda_k - \mu_k}{\sigma_k}\right)^2
\]

(3.47)

At the optimal parameter configuration, the MAP estimate satisfies

\[
E_{p(s,o)}[F_k] - E_{p(o)p(s|o)}[F_k] = \frac{\lambda_k - \mu_k}{\sigma_k^2}
\]

(3.48)

which can be interpreted as a smoothing discount: the model is permitted to under/over-estimate the feature count by a small margin. Usually the mean, \( \mu_k \), is assumed to be zero, and for simplicity all the parameters’ standard deviations are assumed equal, \( \sigma_k = \sigma \). This limits the number of prior hyper-parameters to only one, which considerably simplifies the process of tuning the prior on development data.

### 3.7 Approximate Estimation Techniques

As described in the previous section, we can estimate a CRF using an iterative technique (L-BFGS), where at each iteration we must calculate the marginal distribution over each clique and the partition function. These calculations often prove costly using exact inference with belief propagation. The asymptotic time complexity of the forward-backward algorithm is \( O(NFL^2) \), where \( N \) is the number of cliques over all training sequences, \( F \) is the size of the size of each sparse feature vector, and \( L \) is the size of the label set (state space). The Viterbi algorithm also has the same complexity.

The quadratic complexity in the label set size is particularly problematic when the state space is very large. For instance, part-of-speech tagging often uses tag sets with hundreds of tags and even the very concise Penn Treebank tag-set has 45 different tags (Marcus et al., 1993). Performing inference with such large tag-sets poses considerable demands on time and processing power. The other terms in the complexity, \( N \) and \( F \), are not trivially small for most NLP tasks. Instead we often have corpora with millions of words or other linguistic units, leading to a large \( N \). The freedom over features afforded by log-linear models has led to very rich and highly engineered feature sets. These

---

6The clique (log) potentials are required by the sum-product algorithm; each potential value is the result of a dot product between the parameters and the feature counts of a clique, as shown in (3.28). As the features are most often sparse, this dot product can be performed in \( F \) sum and product operations, where \( F \) is the number of non-zero features.
growing feature sets lead to increases in $F$, particularly as dense features are added to the mix, which cannot be optimised in the same manner as sparse features.

For richer graphical structures than the usual chain – for instance, second order Markov chains, or lattices – the cost of inference is much higher. Such graphs often contain larger maximal cliques than the pairwise cliques in a chain. The time required by belief propagation is $O(NF^C)$, where $C$ the size of the largest clique. Therefore graphs with larger cliques lead to more costly inference. Such graphs are often cyclic and therefore first require graph transformation (see Section 2.3.4.2) to remove loops before running belief propagation. This process clusters many variables into single nodes which expands the state space at each node exponentially, further increasing the cost of inference. For these reasons exact inference in general graphs is intractable, with an exponential complexity. Only for suitably constrained graphical structures, such as linear chains or, more generally, undirected trees with a small branching factor, can exact inference be performed in polynomial time.

Approximation is often necessary in order to perform fast (and tractable) estimation. Such techniques can broadly be split into two groups: those that have been developed for use in general (non-conditional) undirected graphs (MRFs), and those developed specifically for CRFs. We now review those approximation techniques which are applicable to CRFs.

### 3.7.1 Loopy belief propagation

Loopy belief propagation (Pearl, 1988) is a means of approximate inference in cyclic graphs, and was presented in detail in Section 2.3.4.1. There are no guarantees that loopy BP will converge and nor are the resultant marginal distributions guaranteed to be true to the actual distribution. Despite the lack of theoretical justification there is much empirical evidence supporting the method (Weiss, 2000; Ihler et al., 2005). Sutton and McCallum (2004) demonstrate that when loopy BP is used for CRF training divergence is quite rare, and training still finds the globally optimal parameters.

### 3.7.2 Pseudolikelihood

Besag (1974, 1975) introduced the pseudolikelihood measure, a tractable approximation to the likelihood function which eliminates the need for the partition function, $Z$. This measure restates the joint probability over all variables, $p(x_1, x_2, \ldots, x_N)$, as the product of $N$ independent probabilities, $p(x_i|x_{\setminus i})$, $i = 1, 2, \ldots N$, where $x_{\setminus i}$ is the realisation, $x_i$ excluding $x_i$. In each distribution only one variable is allowed to change and all others are clamped to their gold standard values. Therefore normalisation is trivialised, requiring summing over the values of the single variable, $x_i$, regardless of the underlying complexity of the graph.
The pseudolikelihood is only exact in the trivial case where no two states are connected, i.e., where \( x_i \perp x_j, \ i \neq j \). Despite its seemingly coarse approximation the maximum pseudolikelihood estimator has been proven consistent under certain conditions (Besag, 1975; Gidas, 1988).

In the context of CRFs, the pseudolikelihood is given by:

\[
PL(s|o) = \prod_{i \in I} p(s_i|s_{\setminus i}, o)
\]  

(3.49)

where \( I \) is the index set of all state nodes in the graph, and \( s_{\setminus i} \) is the state sequence excluding \( s_i \). Each component distribution \( p(s_i|s_{\setminus i}, o) \) is locally normalised, and thus is extremely cheap to evaluate, as observed in

\[
p(s_i|s_{\setminus i}, o) = \frac{1}{Z_i(o, s_{\setminus i})} \prod_{c \in C: i \in c} \psi_c(s_c, o)
\]  

(3.50)

where the product ranges only over cliques which include the current variable \( S_i \). The effect of all remaining cliques only scales the numerator by a constant in (3.50), which is normalised by the partition function, \( Z_i(o, s_{\setminus i}) \). Therefore the conditioning of the distribution in (3.50) is confined to \( i \)'s Markov blanket, i.e., \( p(s_i|s_{\setminus i}, o) = p(s_i|s_{N(i)}, o) \), where \( N \) is the neighbourhood function. The partition function is the sum of the numerator in (3.50) for every value of \( S_i \):

\[
Z_i(o, s_{\setminus i}) \triangleq \prod_{c \in C: i \in c} \psi_c(s_c, o)
\]  

(3.51)

The pseudo-log-likelihood can be used in place of the log-likelihood for maximum likelihood estimation in (3.42) or maximum a posteriori estimation in (3.47), and regular L-BFGS (or other) optimisation can proceed. The pseudo-log-likelihood is given by:

\[
\mathcal{L}_{PL} = \sum_{s, o} \hat{p}(s, o) \sum_{i \in I} \log p(s_i|s_{\setminus i}, o)
\]  

(3.52)

\[
= \sum_{s, o} \hat{p}(s, o) \sum_{i \in I} \left( \prod_{c \in C: i \in c} \phi_c(s_c, o) - \log Z_i(o, s_{\setminus i}) \right)
\]  

(3.53)

where the log-potentials are used for brevity rather than their expanded feature representation in (3.28).

While pseudolikelihood can greatly simplify estimation, it cannot be used for decoding. The gold standard labels are required in the pseudolikelihood decomposition in (3.49); these labels are, by definition, unavailable at test time. Therefore other techniques, such as belief propagation, must be used for decoding.

### 3.7.3 Piecewise training

**Piecewise training** is similar to pseudolikelihood in that it decomposes the graph into smaller, more tractable, parts and therefore also avoids explicit calculation of the global
3.7. Approximate Estimation Techniques

The partition function, \( Z \) (Sutton and McCallum, 2005). This approximation performs inference over sub-graphs when taken in isolation, ignoring the effect of the remaining nodes and edges. Each sub-graph is locally normalised, thus avoiding the costly (or intractable) calculation of the global normalisation function, \( Z \).

The piecewise approximation gives a upper-bound for the partition function:

\[
\log Z(o) \leq \sum_r \log Z_r(o)
\]  

(3.54)

where the graph is split into \( r \) disjoint pieces, and each piece has its own partition function \( Z_r \). The proof for the proposition in (3.54) is given in Sutton and McCallum (2005).

Although the bound in (3.54) is not a tight one, it leads to a convenient lower-bound on the log-likelihood from (3.42), \( L \geq \mathcal{L}^{PW} \):

\[
\mathcal{L}^{PW} = \sum_o \tilde{p}(o) \left( \sum_s \tilde{p}(s|o) \sum_{c \in C} \sum_k \lambda_k f_k(s_c, o, c) - \sum_{c \in C} \log Z_c(o) \right)
\]  

(3.55)

where we assume that each graph is broken into individual pieces for each maximal clique (factor), denoted here as \( c \). This is the context in which Sutton and McCallum illustrated the method, although as they state, it could instead be applied to any combination of tractable sub-graphs.

Armed with the lower-bound in (3.55), the standard optimisation techniques can be applied (e.g., L-BFGS). The evaluation of \( \log Z_c(o) \) only involves summing out the realisations over the clique, \( s_c \), and calculating its derivatives is also similarly simple. Piecewise training has the same time complexity as standard forward-backward training on chain structured graphs: \( O(NFL^2) \), as described at the start of Section 3.7. Here we can reinterpret \( N \) as the number of pieces into which the graph is split. The method is more compelling for cyclic graphs, where exact inference is intractable. In such graphs, its complexity is \( O(NFL^C) \), where \( C \) is the size of the maximal clique. While it is still intractable in general, many cyclic graphs have small cliques; for these graphs piecewise training only results in a small increase over the cost of training for linear chains.

Sutton and McCallum demonstrated the estimator’s effect on three natural language data-sets, where it showed superior accuracy over that of pseudolikelihood, and in two cases, over the accuracy of belief propagation. This last is a surprising finding, especially considering it occurred for a linear chain CRF, where belief propagation is exact. They refrained from commenting in the paper on this result, but in personal communication attributed it to the exact method over-fitting the training set. This smoothing effect can also be observed in CMMs, another model which decomposes the joint distribution into small pieces. In CMMs the smoothing effect is offset by the label bias problem, and therefore overall accuracy improvements over CRFs are relatively uncommon.
Chapter 3. Conditional Random Fields

3.7.4 Mean field approximation

Mean field approximation is another approach which decomposes the graph into smaller, tractable parts (Jordan and Wainwright, 2005). These assume that the interactions between nodes (or tractable sub-graphs) can be summarised by their mean values, i.e., $p^{\text{MF}}(s_i|o) \propto p(s_i|\mu_{N(S_i)}, o)$, where $\mu_{N(S_i)}$ are the expected values of the node in $S_i$’s Markov blanket. The expected value of $S_i$ can be calculated from the expected values of its neighbours, and these expectations are iteratively calculated until reaching a fixed-point. This iterative process can be slow, and for CRFs, requires additional assumptions.

3.7.5 Sampling

Sampling, or stochastic simulation, allows the calculation of probabilities by counting event frequencies over a series of simulation runs. The simulation draws samples from the posterior distribution, and thus the observed event frequencies will converge to the correct values. These sampling approaches fall under the banner of Markov chain Monte Carlo (MCMC) methods.

The Metropolis-Hastings algorithm may be used to provide such samples (Metropolis and Ulam, 1949; Hastings, 1970) from a distribution of the form $p(\theta) = \frac{f(\theta)}{Z}$. It is initialised with a starting value for $\theta_0$, and then proceeds to ‘jump’ to other parameters, $\theta_{t+1}$ by sampling from a jumping distribution $q(\theta_{t+1}, \theta_t)$. This jumping distribution is chosen such that is simple to sample from. Moves are accepted with probability $\alpha = \min\left(1, \frac{f(\theta_{t+1})q(\theta_{t+1}, \theta_t)}{f(\theta_t)q(\theta_t, \theta_{t+1})}\right)$.

Over time the Markov chain generated by successive samples, $(\theta_0, \theta_1, \ldots, \theta_N)$, converges to samples from $p(\theta)$ without explicit calculation of $Z$, which is often intractable.

Gibbs sampling is a particularly useful MCMC technique for use with graphical models. It was conceived by Geman and Geman (1984) in the context of lattice MRFs for computer vision, and allows for sampling from the true posterior distribution of the MRF by iterative local updates. Each of these updates changes only the state at a single node, by sampling from its posterior given the values of the remaining nodes. For a CRF, the state is sampled from the following distribution:

$$s_i \sim p(s_i|s_{\neg i})$$

$$p(s_i|s_{\neg i}) = \frac{1}{Z_i(o,s_{\neg i})} \exp \sum_{c \in C:i \in c} \sum_k \lambda_k f_k(s_c, o, c)$$  \hspace{1cm} (3.56)

where $Z_i(o, s_{\neg i}) = \sum_{s_i} \exp \sum_{c \in C:i \in c} \sum_k \lambda_k f_k(s_c, o, c)$ is a local normalisation constant, only summing out the scores for the single state, $s_i$. This distribution is cheap to compute, as the numerator in (3.56) only sums over the few cliques which involve $S_i$ (the effect of all the other cliques cancel), and as stated above, the partition function itself only sums out a single state.

The Gibbs sampler iteratively samples each node in the graph, updating the states in a stochastic manner. Typically the sampler is run in a fixed order over the nodes, such
as running left-to-right. This iteration process continues for many passes over the graph. Most often the state configuration after each full pass is used as a sample for measuring event frequencies. For example, to estimate the marginal probability \( p(S_i = s_i) \), we can take \( N \) samples of \( s \), and find the average number of times these samples had the desired value for \( S_i \):

\[
p(s_i) \approx \frac{1}{N} \sum_{j=1}^{N} \delta(s_j^{(j)}, s_i)
\]  

(3.57)

where \( s_j^{(j)} \) is the \( j^{th} \) state configuration sample. Once a sufficient number of these samples have been taken, the Gibbs sampler is stopped. Variance and correlation based measures are often used as stopping criteria. Without this ‘early’ stopping, the Gibbs sampler can be proved to sample from the exact posterior distribution, however this proof only applies in the limit. Another practical consideration is how to start the sampler. Often we start with a default state configuration, and let the sampler settle down to a more realistic configuration over time. This requires a sufficient ‘burn in’ duration, after which time the sample will not reflect the starting configuration.

The Gibbs sampler is an attractive method, as it does away with the global normalisation constant, \( Z(o) \), which can prove intractable to calculate. Furthermore, if allowed to run for a long duration, the marginals produced will closely match the true marginals. However, in practice, this long duration is not feasible within the inner loop of an iterative optimisation process, as in ML and MAP estimation of the CRF (see Section 3.6.1 and Section 3.6.2). This sampler is more appropriate for generative MRFs, which have only a single normalisation constant, \( Z \), with no dependence on any observations. Therefore, for these models the sampler only needs to be run once to estimate \( Z \), which can be reused thereafter. In contrast, CRFs require a separate sampling run for each training instance, and therefore the high cost of the sampler becomes prohibitive.

One aspect of CRF inference which may be more amenable to Gibbs sampling is decoding. Geman and Geman (1984) presented presented the Gibbs sampler as a decoding algorithm to find the maximum posterior distribution. This was achieved by way of simulated annealing, a technique of slowly lowering the temperature over time, akin to the cooling of physical systems resulting in the formation of crystals and other interesting molecular structures. In the MRF context, the temperature is a sharpening constant which exaggerates the peakedness of the distribution. This results in the following distribution:

\[
s_i \sim p_T(s_i|s_{\neg i}) = \frac{1}{Z_{T,i}(o,s_{\neg i})} \exp \left\{ \frac{1}{T} \sum_{c \in C \cap i} \sum_{k} \lambda_k f_k(s_c, o, c) \right\}
\]  

(3.58)

where \( T \) is the temperature parameter, and the partition \( Z_{T,i}(o) \) is modified to include the effect of \( T \). By lowering the temperature (sharpening the distribution), the sampler will converge on the MAP distribution over time, assuming a sufficiently slow cooling schedule. The maximising configuration could also be found by running a non-annealed Gibbs sampler and picking the best seen configuration. However, the annealing process
Chapter 3. Conditional Random Fields

accelerates the sampling process, gradually transitioning to a greedy search, which results in a faster runtime.

Finkel et al. (2005) used a Gibbs sampler for CRF decoding, replicating Sutton and McCallum’s (2004) skip chain CRF as a product of two CRFs. The first CRF was a simple chain model and the second CRF linked sites on pairs of the chains. The chain model was learnt in the standard manner while the parameters of the second model were set by hand. Gibbs sampling was used in decoding the product of these two models which had a cyclic graph and therefore required approximate inference. This product model was shown to outperform a simple chain CRF on two NER tasks, although Finkel et al. did not empirically compare their approach to loopy belief propagation, instead describing it as “ill-founded and inherently unstable.” In our experience, loopy-BP is highly efficient and reliably converges to accurate estimates. Moreover, Finkel et al. did not apply Gibbs sampling during training to replace the heuristic step, probably for efficiency reasons: they report that Gibbs decoding is roughly thirty times slower than Viterbi inference over a chain CRF.

3.7.6 Minimisation of the Bethe free energy

Welling and Sutton (2005) present an exciting approach which minimises the Bethe free energy using variational methods and Monte Carlo sampling. The Bethe free energy is a measure from statistical physics, which is intrinsically related to belief propagation: BP fixed-points are minima of the Bethe free energy (Yedidia et al., 2003). Welling and Sutton show how these minima can be directly approximated without resort to costly belief propagation, using some variational approximation and a number of short samplers. While their approach can result in faster processing, it only achieved this with careful hand tailoring – for example, stopping the Monte Carlo simulations after only a small number of iterations. Further, in their experiments test-set accuracy declines with successive iterations. This produces mixed results, with many scores significantly lower than the belief propagation scores.

3.7.7 Sparse Forward-Backward

Pal et al. (2006) presents a technique which uses ideas from beam search in forward-backward inference. Beam search approximates Viterbi inference for decoding, by limiting the number of states considered at each time to those that are most probable. Recall that in Section 3.5.2 (3.37) finds the maximum scoring partial path from the beginning of the sequence to time \( t \), ending at state \( s_t \), denoted \( v_t(s_t) \). Beam search performs the same calculation, and then truncates the resultant vector, \( v_t \), such that some of low scoring paths are deemed impossible, i.e., \( v_t(s_t) = 0 \). The subsequent step for time \( t + 1 \) then exploits the sparsity of \( v_t \) to constrain the range of the maximisation in (3.37). This sacrifices optimality for efficiency, limiting the decoding complexity to \( O(NFLB) \), where \( B \) is the size of
3.7. Approximate Estimation Techniques

The choice of beam size is task dependent: for some tasks a narrow beam of 5 gives reasonable results, however in general the beam width needs to be determined by experimentation, and may be much larger. While narrowly focussed beams are common in decoding scenarios, a wider beam may be required during training where accurate marginal distributions are required for efficient traversal of the objective surface.

Pal et al. use these ideas to produce sparse forward and backward vectors in the forward-backward algorithm (see Section 3.5.1). They compare several different beam heuristics, finding the best heuristic to be one which takes the best values from $\gamma_t(s_t) = \frac{a_t(s_t) b_t(s_t)}{Z}$ up to a fixed probability mass, while ensuring a minimum beam size. This requires the forward and backward values before any pruning can occur, and leads to an iterative method which repeats forward and backward passes (and beam pruning) until reaching a fixed-point. They do not specify how they start this process, but presumably used a fixed size beam in an initial backward pass. They claim a four-fold speed up in CRF training time, however this technique does not address the real underlying issue of scalability: it is unclear whether the complexity is lower than that of standard forward-backward inference.

3.7.8 Feature Induction

Another way in which CRF inference can be made more efficient is to limit the size of the feature set. Usually the feature set is fixed and the CRF training process finds the best parameter values for these features. However if we relax the fixedness assumption and instead only use the truly important features, the number of parameters of the model will be greatly reduced. This has the effect of speeding inference by reducing the numbers of active features, $F$, in the complexity $O(NFL^2)$, as described in Section 3.7. This reduced cost inference hastens both ML or MAP training and decoding, which are both performed in the usual way but with the new feature set. Furthermore, this also has the effect of limiting the modelling power of the CRF, thereby reducing its tendency to overfit the training sample. McCallum (2003) presented a technique for feature induction with CRFs based on Della Pietra et al.’s (1997) induction technique for MRFs. This method starts with a basic feature set which it iteratively expands to greedily improve the log-likelihood of the training data. This avoids the addition of many redundant features which have little modelling power or replicate other features.

This technique appears very attractive, often finding small feature sets with only tens of thousands of features (rather than the usual hundreds of thousands or millions) and achieving good accuracy (McCallum, 2003). However, the technique requires many iterations of feature expansion, and each of these steps require the training of a CRF. This adds up to a very heavyweight and slow process, despite the fact that the final model is comparatively lean. Furthermore, the magnitude of the gain in accuracy by using induction was over-stated due to a bug in the baseline implementation (McCallum,
personal communication).

### 3.7.9 Averaged Perceptron

All of the previous training methods were based on maximum likelihood training, where the parameters are chosen to maximising the (penalised) likelihood of the training set. There are numerous other common objective functions, with which we can fit the training set. For instance, perceptrons (Rosenblatt, 1958), use a 0-1-loss function, \( l(x, y) = \delta(x, y) \), which allocates a score of 1 for a correct prediction \((x = y)\), and 0 otherwise \((x \neq y)\), regardless of the probability distribution over labellings.

A perceptron is a single layer neural network, which represents a linear function, \( y = \text{sign}(v \cdot x) \), where \( y \) is the binary prediction, \( v \) is a vector of parameters, \( x \) is the vector valued feature representation of the input, and \( \text{sign} \) is a function returning +1 or −1 for positive or negative arguments, respectively. The online perceptron training method starts with parameters \( v = \{0\} \), then repeatedly iterates over the training sample. On each instance, it makes a prediction, \( \hat{y} = \text{sign}(v \cdot x) \). If the prediction differs from the true label, \( y \), the model parameters are updated, \( v \leftarrow v + yx \). For correct predictions, the parameters remain unchanged. The training terminates after a predefined number of iterations, or else when there is no training error. The final parameters, \( v \), are then used for decoding the test set.

The averaged perceptron (and voted perceptron) algorithm is a variant of the classical perceptron algorithm modified to include regularisation (Freund and Schapire, 1999). The training phase is as before, except that instead of using the final parameters for decoding, an averaged set of the parameters are used. These parameters are the arithmetic mean of the parameter values at each stage of training. The voted perceptron caches all the parameter values and then decodes test instances using a majority vote. Freund and Schapire show how both voting and averaging lead to less over-fitting than the standard perceptron, both theoretically and empirically.

Perceptron training has been applied to training of multiclass sequence models, including HMMs and CRFs (Collins, 2002a; Roark et al., 2004). This requires only a small variation in the training regime, where classification is performed as follows:

\[
\mathbf{s} = \arg \max_{\mathbf{s}} \sum_{k} \lambda_k \Phi_k(\mathbf{s}, \mathbf{o})
\]

(3.59)

where \( \Phi_k(\mathbf{s}, \mathbf{o}) \) is the value of feature \( k \) on the given instance with state sequence \( \mathbf{s} \) and outputs \( \mathbf{o} \). In order to mimic a CRF, the feature function is set to:

\[
\Phi_k(\mathbf{s}, \mathbf{o}) = \sum_{c \in C} f_k(s_c, o, k)
\]

(3.60)

\(^7\)The perceptron can mimic other models, such as HMMs, simply by using a different definition for \( \Phi \).
3.8. Conclusion

and thus (3.59) maximises the log probability, \( \log p(s|o) \), after omitting the log partition function. In other words, (3.59) performs max-product (or Viterbi) decoding, as described in Section 3.5.

The perceptron algorithm requires modification to cope with the multiclass sequence decoder in (3.59). The algorithm starts with a zero parameter vector, and then iterates over the training set many times. If the predicted state sequence for an instance does not exactly match its correct state sequence, then an error is said to have occurred, and the parameters are updated. The parameter update is also modified to include the difference between the feature representation of the correctly labelled instance and the feature representation for the erroneous predicted labelling, i.e., \( \Lambda \leftarrow \Lambda + \Phi(s,o) - \Phi(\hat{s},o) \). The voted and averaged perceptron variations remain unchanged.

This model can no longer be described as a CRF, as it no longer models a probability distribution. However, it does still optimise a similar functional form to that of the CRF, in a manner directly relevant for decoding. Furthermore, it also shares the exact same usage of the max-product algorithm. Empirical studies suggest that averaged or voted perceptron training is fast, requiring only a small number of epochs, while yielding good models with impressive accuracy (Freund and Schapire, 1999; Collins, 2002a; Roark et al., 2004). Often beam-search decoding is used, which can further speed the perceptron training and potentially lead to complexity improvements over Viterbi inference.

3.8 Conclusion

This chapter has described Conditional Random Fields (CRFs) in the context of their forebears, Hidden Markov Models (HMMs) and Conditional Markov Models (CMMs). Specifically, it showed how CRFs address issues arising from the independence assumptions HMMs and the label bias problem of CMMs. This allows a richly parameterised model with globally optimal estimation and decoding. However, for general graphs, such as those used for joint inference over many separate layers of annotation, the standard inference technique used in maximum likelihood training and decoding is intractable. Furthermore, even when only considering traditional linear chains, the cost of inference can be high with poor scaling characteristics. The chapter reviewed and contrasted a number of mechanisms which approximate inference in CRFs. The following chapters compare the empirical accuracy of some of these approximations on natural language data-sets.
Chapter 4
Experimental Setup

This thesis deals with the application of Conditional Random Fields (CRFs) to natural language processing tasks (NLP), presenting novel algorithms which allow CRFs to be scaled to large tasks. These methods aim to reduce CRF training time and its asymptotic complexity without harming the strong accuracy of standard training (presented in Chapter 5). In order to gauge the extent of reductions in training time and to measure accuracy differences, the approximate CRF training methods are compared on a range of NLP tasks.

This chapter presents three NLP tasks: part-of-speech (POS) tagging, named entity recognition (NER) and joint POS tagging and noun-phrase chunking (NPC). The tasks are all tagging tasks and can therefore be modelled by a CRF in a straight-forward manner using either a linear chain or a two dimensional lattice graph (Lafferty et al., 2001; McCallum and Li, 2003; Sutton et al., 2004). These tasks have been chosen to highlight the most significant factors in the CRF’s training complexity: the label set size, the size of the training sample and the graphical structure. This chapter includes a discussion of the software architecture and the most critical design decisions. These considerations can radically affect the efficiency of the software.

The chapter is structured as follows. Sections 4.1, 4.2 and 4.3 present the three tasks. They describe the graphical structure and the feature set used to model each task. Section 4.4 describes the manner in which the model’s timings and test set accuracy are measured and the statistical significance tests. Section 4.5 describes the CRF software developed for the thesis and Section 4.6 presents the conclusions of this chapter.

4.1 Part-of-speech Tagging

Part-of-speech (POS) tagging describes the process of assigning a POS tag to each token\(^1\) in a body of text. POS tags indicate the basic syntactic function of that token, such

\(^1\)The standard notation of *token* and *type* is used to indicate either an *instance* of a word in context, or the word’s identity, respectively.
Chapter 4. Experimental Setup

Table 4.1. Part-of-speech corpus sample statistics using the training/development/test split of Collins (1999). Novel types refers to the number of types in the sample that are not present in the training sample.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sections</td>
<td>2–21</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>Tokens</td>
<td>1,023,863</td>
<td>54,397</td>
<td>56,824</td>
</tr>
<tr>
<td>Types</td>
<td>46,071</td>
<td>8,584</td>
<td>8,443</td>
</tr>
<tr>
<td>Novel types</td>
<td>-</td>
<td>1,229 (14.3%)</td>
<td>1,068 (12.6%)</td>
</tr>
<tr>
<td>Sentences</td>
<td>43,189</td>
<td>2,244</td>
<td>2,397</td>
</tr>
</tbody>
</table>

as *noun* or *verb*, as well as other grammatical information, such as number and tense. POS tagging is a fundamental preprocessing step for many other NLP applications (e.g., syntactic parsing). Typically POS tags provide general shallow syntactic information to these downstream applications.

The Penn Treebank III was used for this task (Marcus et al., 1993, 1994). The corpus consists of parsed and tagged text collected from a number of different sources, of which only the 1989 Wall Street Journal (WSJ) portion was used. An example sentence from the corpus is shown below, with gold standard POS tags shown as subscripts:

```
PierreNNP VinkenNNP , 61CD yearsNNSS oldJJ , willMD joinVB theDT boardNN asIN aDT nonexecutiveJJ directorNN NovNNS 29CD .
```

These tokens have each been annotated using Penn Treebank tag-set of 48 tags, of which only 45 appear in the WSJ sample. Some tokens are tagged with more than one tag in cases where the annotator judged that there was true ambiguity. In these rare instances the first tag was treated as the gold standard.

We have used training split of Collins (1999) and the resulting sample statistics are summarised in Table 4.1 which shows token, type and sentence counts as well as the number of types which are not present in the training sample. These unseen types present the most difficulties for automatic systems.

Following the standard approach of Lafferty et al. (2001), we use a linear chain CRF to model the POS task. For each sentence a graph is created with a node for each word representing its labels, and edges connect each adjacent pair of nodes. This corresponds to a first order Markov assumption over the label sequence, which is clearly incapable of modelling the true distribution over labels. The Markov assumption should be viewed as an approximation which captures the greatest source of variability – neighbouring tags – but discards any long distance dependencies. There is evidence that higher order Markov assumptions are useful in modelling POS sequences (Brants, 2000; Ratnaparkhi, 1996; Toutanova et al., 2003), allowing features over label trigrams and larger sequences. However, we constrain ourselves to a first order Markov assumption for efficiency reasons: the cost of training a first order CRF is already very large, while the cost of training a higher order model would be prohibitive (many orders of magnitude larger).
4.1. Part-of-speech Tagging

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Singles?</th>
<th>Pairs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>always on</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token</td>
<td>the token at the current index</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token$_{lc}$</td>
<td>the current token lower-cased</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Token$_j$</td>
<td>the token at index $j = -2, -1$ or 1</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

The remaining features are all orthographic tests on the current token

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Singles?</th>
<th>Pairs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit</td>
<td>contains a digit</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Number</td>
<td>is a number (real or integer)</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Upper</td>
<td>contains an upper case letter</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Upper$_0$</td>
<td>starts with an upper case letter</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Hyphen</td>
<td>contains a hyphen</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Slash</td>
<td>contains a forward- or back-slash</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Prefix</td>
<td>prefix of lengths 1, 2, 3 and 4</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Suffix</td>
<td>suffix of lengths 1, 2, 3 and 4</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4.2. Feature templates used for POS tagging. The singles and pairs column indicate how the given contextual predicates are paired with clique labellings to create the features.

The features used for modelling the POS tagging task were loosely based on those used by Ratnaparkhi (1996). Most recent work on POS modelling has employed a very similar feature set, suggesting that these features are sufficient to accurately model the task. The features are shown in Table 4.2, where each row describes a function over the observations (referred to as a contextual predicate). This function is typically applied the ‘current’ token, $o_t$, or tokens with a window of the current token. The contextual predicate is paired with a clique labelling: either a single label, $s_t$, or a pair of adjacent labels, $(s_{t-1}, s_t)$.\(^2\)

We refer to each row as a feature template; these specify a set of feature functions (or just features) which all share the same contextual predicate. Each feature tests for a specific value of the contextual predicate and also for a specific value of the clique labelling. For example, the following feature matches the token feature template:

$$f_{\text{fish}/\text{NN} \,(s_c, o, c)} = [s_c = \text{NN} \land o_c = \text{“fish”}]$$

The vocabulary of (value, clique labelling) pairs used to define these features are taken from the training set, as is the standard approach for maximum entropy models (Sha and Pereira, 2003). In this manner, the above feature would be included only if the labelled token fish$_{NN}$ was attested in the training set. Features which were not attested can also be of use, as they allow the model to learn a dispreference for incorrect clique labellings. The full set of unattested features (created by pairing each contextual predicate value with all

\(^2\)Note that contextual predicates which reference observations outside of the sequence (e.g., $token_{-1}$, when at position 0) will return a special start or end value.
possible clique labellings) is typically very large, which affects the cost of training and decoding. For this reason we only use the attested set, which already presents tractability problems with standard training.

The feature templates do not draw a distinction between rare types and common types, as in Ratnaparkhi (1996). This distinction is used to exclude common word types from orthographic tests. The orthographic tests are most useful for unseen types, which tend to have a different distribution over parts-of-speech compared to common types. A count threshold allows the rare types in the training data to be used to learn the distribution for unseen tokens occurring at test time. We performed some simple experiments which indicated that imposing a threshold slightly degraded accuracy: i.e., any improvement was offset by increased sparsity of the orthographic feature counts.

4.2 Joint Part-of-Speech Tagging and Noun-Phrase Chunking

The second of the three tasks is noun-phrase chunking (NPC), a subtask of shallow parsing which requires bracketing of non-embedded noun-phrase constituents. This process is useful for information extraction and other tasks in which only very limited syntactic information is required and the cost of full parsing is impractical. We use a joint CRF model to describe both POS tagging and NPC following Sutton et al. (2004). This model improves over typical cascade models which perform NPC using predicted POS sequences, and therefore often do not find the best joint POS and NPC labellings.

Experiments were carried out on the CoNLL 2000 shared task data (Tjong Kim Sang and Buchholz, 2000), which consists of a fraction of the WSJ section of the Penn Treebank II corpus (Marcus et al., 1993) (sections 15–18) annotated for syntactic chunks. The annotation of this corpus was a fully automatic process, projecting the parse trees from the Treebank into flat syntactic structures of non-overlapping or embedded chunks. For this task only the noun chunks are considered while the other chunk types are ignored.

In this corpus there were 45 POS tags. An IOB style labelling (Tjong Kim Sang and Veenstra, 1999) was used to represent the syntactic chunks; this resulted in 3 labels considering only the noun chunks. The IOB system allows for segmentation tasks to be represented as per-token tagging tasks. Each token is labelled with Begin, Inside or Outside label depending on whether the token starts, continues or is not part of a segment. The IOB method only uses the Begin tag to separate two adjacent segments (i.e., only when necessary), while the IOB2 system uses the tag at the start of each segment.

An example sentence from the corpus is given below, where chunks are indicated with bracketing, and POS as subscripts:

\[And_{CC} [ship_{NN} lines_{NNS}] carrying_{VBG} [containers_{NNS}] are_{VBP} also_{RB} trying_{VBG} to_{TO} raise_{VB} [their_{PRP}$ rates_{NNS}] .\]

The training sample from the CoNLL 2000 task was randomly split into two samples
4.3 Named Entity Recognition

The last of the three tasks is named entity recognition (NER), which involves finding instances of person names, organisations, locations and other entities of potential interest in text. An example from Tjong Kim Sang and De Meulder (2003) is shown below:

\[ \text{[organisation U.N.] official [person Ekeus] heads for [location Baghdad]} \]

The tags indicate the entity type, and the bracketing indicates the extent of the entity. Named entity recognition is a critical sub-task of information extraction and question

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokens</td>
<td>168,281</td>
<td>43,446</td>
<td>47,377</td>
</tr>
<tr>
<td>Types</td>
<td>17,131</td>
<td>8,066</td>
<td>8,119</td>
</tr>
<tr>
<td>Sentences</td>
<td>7,099</td>
<td>1,837</td>
<td>2,012</td>
</tr>
</tbody>
</table>

Table 4.3. Chunking corpus sample statistics, using the CoNLL 2000 shared task data set (Tjong Kim Sang and Buchholz, 2000). The original training sample was split into the training and development samples.

using a 4:1 ratio over sentences, in order to provide a held-out set for parameter tuning. The sample statistics are given in Table 4.3.

The joint tagging task requires two layers of annotation for POS and NPC sequences. Here we follow Sutton et al. (2004) and model the task with a two dimensional lattice consisting of two interconnected chains. This lattice structure allows features over POS or NPC labels and over (POS, POS), (POS, NPC) and (NPC, NPC) label pairs. These graphs are cyclic and therefore inference is considerably more difficult than for simple chains. In order to perform exact inference the graph first needs to be transformed to remove cycles. This can be achieved by collapsing all the layers of nodes at each point into a single node, resulting in a chain. Each of these new nodes has a much larger state space: the Cartesian product of the label sets in each layer. While this might seem to be a viable approach for a two layer lattice it is not a general solution: the size of the label set in the resulting chain is exponential in the number of layers, and therefore inference is intractable. Even for the two layer lattice exact inference was found to be extremely slow compared to approximate inference using loopy belief propagation (BP) (Pearl, 1988). For this reason, loopy BP was used instead, which was found to converge for most graphs in fewer than ten iterations, while each iteration had a similar cost to forward-backward inference in a linear chain (with at most 45 labels).

The feature set used in the joint POS and chunking experiment is shown in Table 4.4, where the righter-most columns indicate what types of clique labellings are detected in a conjunction with the contextual predicate. As with the POS tagging task only attested features were used (see Section 4.1).
Table 4.4. Feature templates used for joint POS tagging and chunking. The final five columns indicate how the given contextual predicate are paired with clique labellings to create the features. Here, \( p_j \) and \( c_j \) refer to POS and chunk cliques, respectively, of size \( j \), while \( p&c \) refers to the cliques connecting the POS and chunk chains.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( c_1 )</th>
<th>( c_2 )</th>
<th>( p&amp;c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>always on</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token</td>
<td>the token at the current index</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token,( j )</td>
<td>the token at index ( j = -2,-1,1 ) or 2</td>
<td>✓</td>
<td>( \times )</td>
<td>✓</td>
<td>( \times )</td>
<td>✓</td>
</tr>
<tr>
<td>Bigrams</td>
<td>tokens at (-2,-1; -1,0; 0,+1;) and (+1,+2)</td>
<td>( \times )</td>
<td>( \times )</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Trigrams</td>
<td>tokens at (-2,-1,0; -1,0,+1; ) and (0,+1,+2)</td>
<td>( \times )</td>
<td>( \times )</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Digit</td>
<td>current token contains a digit</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Upper</td>
<td>current token contains an upper case letter</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Hyphen</td>
<td>current token contains a hyphen</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Prefix</td>
<td>current token prefix of lengths 1, 2, 3 and 4</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Suffix</td>
<td>current token suffix of lengths 1, 2, 3 and 4</td>
<td>✓</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>

Table 4.5. Named-entity corpus sample statistics, using the CoNLL 2003 shared task data set (Tjong Kim Sang and De Meulder, 2003).

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokens</td>
<td>203,621</td>
<td>51,362</td>
<td>46,435</td>
</tr>
<tr>
<td>Sentences</td>
<td>14,987</td>
<td>3,466</td>
<td>3,684</td>
</tr>
</tbody>
</table>

We use the English CoNLL 2003 named-entity corpus (Tjong Kim Sang and De Meulder, 2003) which consists of Reuters news stories. The training and development sets consist of ten consecutive days of news-wire stories, while the test set consists of stories from a day three months later. The data was annotated for four types of entity (organisation, person, location and miscellaneous) using the IOB tagging scheme (Tjong Kim Sang and Veenstra, 1999), resulting in 8 labels. The data was also automatically POS tagged as described in Tjong Kim Sang and De Meulder (2003). The sample statistics are given in Table 4.5.

The NER task features a single layer of annotation with one IOB label for each token in each sentence. As with POS tagging, a linear chain CRF was used. Linear chain CRFs have been used previously with strong results on the NER task (McCallum and Li, 2003; Smith et al., 2005). The task is small enough in terms of labels and training sentences that the cost of standard training is quite acceptable, only taking a few hours. The inclusion of this task in our set of three tasks allows for comparison with previous work.
The feature templates are shown in Table 4.6. These features were developed in collaboration with Andrew Smith and were based on Curran and Clark (2003). Only attested features were generated from these templates, as described in Section 4.1. The token shape contextual predicates use the approach described in Collins (2002b), which applies some simple character level transformations to the token, producing a general representation of the token’s orthographic form. This process collapses all adjacent sequences of uppercase letters to \( A \), lower-case letters to \( a \) and digits to 0. For example, Pierre becomes \( Aa \) and non-committal becomes \( a-a \).

4.4 Evaluation Techniques

The previous sections described the three experiments which we have used to analyse the time and accuracy of various CRF training methods in the following chapters. In this section we describe the performance evaluation metrics and how the training and decoding times were measured. We also describe the statistical significance tests used to compare accuracy scores.

4.4.1 Accuracy Measures

The accuracy for each task has been evaluated using standard measures already used in the field. POS tagging quality was measured using accuracy on per-token and per-sentence bases. It is important to note that accuracy must be interpreted relative to a baseline – in many tasks, using naive methods such as predicting the most frequent label yield a high accuracy. For the POS tagging task, simply predicting the most frequent tag for each word and \( \texttt{NN} \) for unknown words achieves an accuracy of 92.0%. We also report accuracy on known and unknown tokens separately.

Chunking and NER accuracy was measured using the precision, recall and \( F_1 \) measures over chunks (entities).\(^3\) The precision, \( p \), and recall, \( r \), are defined as:

\[
P = \frac{tp}{tp + fp} \quad (4.1)
\]
\[
R = \frac{tp}{tp + fn} \quad (4.2)
\]

where \( tp \) is the number of \textit{true positives} (correctly identified chunks), \( fp \) is the number of \textit{false positives} (incorrectly identified chunks) and \( fn \) is the number of \textit{false negatives} (incorrectly omitted chunks). We wish to achieve a high precision, such that most of the identified chunks are correct and also a high recall, such that we identify most of the chunks. One can choose to maximise either precision or recall at the expense of the other, by under- or over-predicting chunks. In most applications we desire a compromise

\(^3\)We use the term \textit{accuracy} in its general sense of test set performance which is measured using a range of different metrics, including per-token accuracy.
### Table 4.6. Features templates used in the named entity recognition task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Singles?</th>
<th>Pairs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>always on</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token</td>
<td>the token at the current index</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Token bigrams</td>
<td>the bigrams of tokens at -2,-1 and +1,+2</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>POS tag</td>
<td>the POS tag at the current index</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>POS bigrams</td>
<td>the bigrams of POS tags at -2,-1 and +1,+2</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Shape</td>
<td>shape of the token at -1, 0, and 1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Shape bigrams</td>
<td>token shapes at -2,-1; -1,0; 0,+1; +1,+2</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Shape trigrams</td>
<td>token shapes at -2,-1,0; -1,0,+1; and 0,+1,+2</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

The remaining features are all orthographic tests on the current token and only apply to rare types: those occurring fewer than 5 times.
between the two, which is embodied in the F score, a harmonic mean between precision and recall:

\[ F_\alpha = \frac{pr}{\frac{\alpha}{2}p + (1 - \frac{\alpha}{2})r} \]  

In the absence of information about the relative costs of predicting false negatives or false positives, we assume the costs are equal, and choose \( \alpha = 1 \). This measure is commonly referred to as \( F_1 \), or simply the F score.

For the joint chunking and POS tagging task, in addition to the chunking \( F_1 \) score, we also measure per-token accuracy. This accuracy is measured for the POS labels and also jointly for the (POS, NPC) label pairs.

### 4.4.2 Timings

The training and decoding times were measured in wall-clock seconds on an IBM Power5 1.6GHz machine with 16Gb of RAM running SUSE Linux. These timings only include the core training and decoding processes and exclude all preparatory steps, such as feature detection and creating or loading the event structures, which are described in Section 4.5. As there are a number of parameters which can affect the training and decoding times, these were held constant. The L-BFGS optimiser (Nocedal, 1980) (used to optimise the loglikelihood, as described in Section 3.6) used an absolute convergence threshold of \( 10^{-4} \) and a history of up to fifty previous evaluations.\(^4\) In each experiment the Gaussian prior was fitted by hand in order to maximise the development accuracy. The training time varies considerable using different prior variance – by as much as three times – which reflects the increasing difficulty of fitting the parameters under the soft constraints imposed by a prior. Therefore when reporting the run-time of CRF training and approximate training, we have used a fixed prior with a zero mean and variance of \( \sigma^2 = 10 \). This value is a good default value across many tasks and training methods.

In instances where additional memory resources beyond 16Gb were required, the software was run in parallel over a cluster of identical machines. The results obtained using a cluster of machines are explicitly flagged and the aggregate times are reported. These figures incorporate the extra communication time incurred in the parallel implementation as we measure wall-clock time. In order to minimise the communication cost we have used the smallest number of CPUs possible for each experiment, typically only using two or four processors for tasks with very high memory demands. We estimate that in such small parallel configurations the communication cost contributes to no about 15% of the total runtime.

### 4.4.3 Significance Testing

\(^4\)Although a history of 5-10 is sufficient for convergence, using a larger history typically leads to faster convergence.
for all labelled sequences, \((s^{(i)}, o^{(i)})\), in test set do
    \[ a \leftarrow \text{number of errors made by model } A \text{ on } o^{(i)} \]
    \[ b \leftarrow \text{number of errors made by model } B \text{ on } o^{(i)} \]
    \[ Z_i \leftarrow a - b \]
end for

\[ \hat{\mu} \leftarrow \sum_{i=1}^{n} \frac{Z_i}{n} \]
\[ \hat{\sigma}^2 \leftarrow \frac{1}{n-1} \sum_{i=1}^{n} (Z_i - \hat{\mu})^2 \]
\[ W \leftarrow \frac{\hat{\mu}}{\hat{\sigma} / \sqrt{n}} \]

Figure 4.1. Algorithm for calculating the matched-pairs statistic given two models and a test set, where \( n \) is the number of sequences in the test set.

The statistical significance of the accuracy was assessed using McNemar’s matched-pairs test, as advocated by Sha and Pereira (2003) and Gillick and Cox (1989). This test measures the difference in the number of errors made by two different models on each sentence in a testing set and is illustrated in Figure 4.1. The null hypothesis is that \( Z = 0 \), where \( Z_i = a - b \) denotes the difference in the number of errors made on a sentence by classifiers \( A \) and \( B \). This is assessed on a per-sentence rather than per-token basis because each token in a sequence is clearly not independent of other tokens in the sequence. With sufficient data the distribution can be accurately approximated with a normal distribution using the sample statistic, \( W \), in Figure 4.1. A two tailed test was used reflecting that we are testing for a significant difference in accuracy regardless of sign. This equates to testing if the magnitude of \( W \) exceeds 1.960, 2.576 or 3.290, which correspond to the thresholds for \( p < 0.05, 0.01 \) and 0.001 respectively.

Note that this test does not consider any underlying chunk labelling inferred by an IOB style labelling, and thus does not measure the significance of an \( F_1 \) score. Instead it measures the significance of the accuracy. In most cases the ranking of classifiers using accuracy and \( F_1 \) score remain the same, with similar separating margins for both measures. Therefore the test should still provide a reasonable proxy for the significance of difference in these other measures. Bootstrapping approaches can be used to measure the significance of the \( F_1 \) score, however Sha and Pereira (2003) report that these measures can be swamped by variance and therefore advocate the use of the McNemar test in preference.

### 4.5 Software

We now turn to the software used to carry out these experiments. This application allows for the training and testing of CRFs on the above natural language tasks. The software was developed for this purpose, and was written mainly in C++ with small auxiliary components written in Python. The use of C++ allowed for fast execution of time (and space) critical code, while also allowing access to a range of efficient open source libraries.
Figure 4.2. Architecture flow diagram, showing the files and processes involved.
Chapter 4. Experimental Setup

The general architecture of the application is shown in the data-flow diagram in Figure 4.2. The data stores (input and output files) are shown as pairs of parallel horizontal lines, and processes are shown as rectangles. The overall process takes as input three files – the feature templates, the training set and the development/test set – and outputs a file of predicted label sequences. The processing steps involve the detection of features over the input text, the extraction of events (the feature values for all clique labellings), maximum likelihood training, and max-product decoding. Each process is described below.

4.5.1 Feature Detection

Feature detection takes as input a training set and a set of feature templates, from which it finds both the set of labels in the training set and the set of attested features. The training set is supplied in CoNLL format, consisting of tokens and a number of layers of annotation, including the gold-standard labels. The format also marks sentence boundaries.

The feature templates are described in XML. A sample is given in Figure 4.3 showing the template from the NER task (see Table 4.6) which combines the token identity with a label pair. The clique type of 1 specifies that the template should be applied to label pairs (type 0 would indicate single labels) and the threshold of 1 indicates that it need only be attested once to generate a feature. The predicate templates section allows specification of any number of contextual predicates; the features use the conjunction of these predicates. Here only one predicate template is given to select the current token. There are also a number of different predicate template types which allow for different regular expression matches against the entry, table lookup, affixes etc. The index and offset specify which item is to be examined. The offset is a relative offset from the righter-most node in the clique, and the index identifies which layer of annotation should be used. In this case the offset and index are both 0, specifying the current token.

From these templates the features are created (attested), where each binary feature is formed by combining a set of predicate values and a labelling (e.g., token identity and label pair). These feature definitions are saved to file along with the label set used in the training sample.

4.5.2 Event Extraction

The process of event extraction takes as input the features, labels and a data set, and creates an event matrix. This matrix describes the feature values for every possible labelling for every clique in the data. The data is processed by first creating a graph for each sentence. For linear chains this graph simply contains a node for each token in the sentence and links between adjacent nodes. The two-layer POS/NPC lattice requires two nodes for each token – one for POS and one for NPC – and creates edges linking pairs of aligned POS and NPC nodes, pairs of adjacent POS nodes and pairs of adjacent NPC nodes. The event matrix stores the feature values (an event) for each clique in the graph combined
with a clique labelling. There are often millions of events in an event matrix due to the number of possible labellings of each clique (of which there are often very many).

The event matrix is usually very sparse, in that for each clique labelling only a tiny fraction of the features are ever active. All the features used here are binary and therefore all active features will have value 1. Accordingly, the events are stored as a sparse matrix in compressed row storage (CRS) format (Balay et al., 2001), which records the indices of the active features for each event. This format was found to provide much smaller storage than other alternative formats and also led to faster run-times due to better cache utilisation.

Finally, the event matrix is either written to file or is stored in memory for the subsequent training and/or decoding phases. As the matrix is usually very large and the features are usually quite simple, it often takes a similar amount of time to extract the events from the data file as it does to read the events file from disk.

### 4.5.3 Training

The training process finds the maximum likelihood parameters given an event matrix by optimising the log-likelihood of the training sample using L-BFGS, an iterative numerical solver (Nocedal, 1980). This solver searches for the global maximum value by gradient ascent of the objective (log-likelihood) surface, as described in Section 3.6. The Portable Extensible Toolkit for Scientific Computation (PETSc) and Toolkit for Advanced Optimisation (TAO) libraries were used for this purpose (Balay et al., 2004; Benson et al., 2005).

**Figure 4.3.** Sample XML snippet from a feature templates file.
Chapter 4. Experimental Setup

PETSc provides a suite of efficient generic vector and matrix data structures and manipulation operations and TAO builds upon PETSc to add a number of numerical solvers, including line search, conjugate gradient and L-BFGS. PETSc and TAO also provide in-built support for parallelisation using the Message Passing Interface (MPI) to allow for massively parallel execution (Gropp et al., 1996). We used two MPI implementations – MPICH and LAM – for efficient distribution of workload between processors in a cluster (Gropp and Lusk, 1997; Burns et al., 1994).

Training with L-BFGS requires repeated evaluation of the log-likelihood objective function and its first order partial derivatives, as described in Section 3.6.1. The equations are reproduced below:

\[
L = \sum_o \hat{p}(o) \left( \sum_s \hat{p}(s|o) \sum_{c \in C} \sum_k \lambda_k f_k(s,c,o) - \log Z(o) \right)
\]  

(4.4)

\[
\frac{\partial L}{\partial \lambda_k} = \sum_{o,s} \hat{p}(s,o) f_k(s,c,o) - \sum_{o} \hat{p}(o) \sum_s p(s|o) \sum_c f_k(s,c,o)
\]  

(4.5)

Note that in both (4.4) and (4.5) the outer summation ranges over the observations, \( o \). Therefore the calculation can be trivially parallelised, by partitioning the training sample over a number of CPUs. Each CPU calculates (4.4) and (4.5) for its training subsample, and the results of these independent calculations are then summed together.

The training algorithm uses the method described above for parallelisation, as illustrated in Figure 4.4. The first processor (the ‘root’) performs the L-BFGS optimisation, and communicates the parameters to all other processors in each iteration. The root and remaining nodes then calculate the log-likelihood and its derivative for their training subset, with the results communicated back to the root and summed (using a MPI reduce operation). In this configuration all processors calculate their part of (4.4) and (4.5) in parallel, and then must wait for the root to receive all incoming messages, compute the next L-BFGS step and broadcast the new parameters. As training usually only takes a few hundred iterations, this waiting time is only a small proportion of total time, especially when using a fast network and a small number of processors.\(^5\)

Sum-product belief propagation (described in Section 3.6) is required to evaluate (4.4) and (4.5). Belief propagation (BP) has been implemented in three guises: for chains, for trees and for cyclic graphs. For linear chains sum-product belief propagation reduces to the forward-backward algorithm, as described in Section 3.5.1. The chain implementation has been extensively optimised, as linear chains are used in the majority of CRF applications. Loopy BP is used for cyclic graphs using a random schedule and a convergence threshold of \( 10^{-4} \) and stopping after fifty iterations. This schedule was found to provide faster convergence than other fixed or tree based schedules (see Section 2.3), and was found to converge in fewer than ten iterations in most instances. Tree BP uses mostly the

\(^5\)Most of the experiments which required parallelisation were run on four processors. We estimate that this setup increased the total CPU time by about 15% compared to running on a single machine.
same implementation as loopy BP but with a tree based schedule which requires only two rounds of processing (the distribute and collect passes).

BP was implemented in log-space in order to keep the messages and exponential dot products small and therefore avoid overflow. This simplifies multiplication and division operations, which become addition and subtraction in log-space. However, this complicates addition and subtraction operations, which cannot be naively evaluated without risking numerical overflow. Addition is performed using the following trick (Taylor et al., 1988):

\[
\log(\exp x + \exp y) = \log((\exp x) \times (1 + \exp(y - x))) = x + \log(1 + \exp(y - x))
\]

(4.6) (4.7)

This avoids calculating either \(\exp x\) or \(\exp y\). Instead \(\exp(y - x)\) is calculated, which, if \(x\) is chosen to be the larger of \((x, y)\), will always be less than or equal to one. This allows for much larger values of \(x\) and \(y\) before numerical overflow becomes a problem. This same trick also works for subtraction, with the caveat that negative values cannot be represented. For our purposes, we only deal with positive values, although we must careful to avoid negative intermediate results. Note also that underflow is less of a problem, as the logarithmic number system is more sensitive to tiny values than the standard floating point representation (Taylor et al., 1988).

Alternative methods can be used to prevent numerical under- and over-flow. For example, scaling constants can be included during inference to keep intermediate values near to one (Rabiner, 1990). However, scaling is not always sufficient to prevent numerical problems, especially in loopy BP which requires many rounds of message passing which can lead to escalating values with each round. This can also cause problems for the logarithmic representation. For this reason, we normalise each message to sum to one. This ensures that the messages do not continue to grow or shrink during many iterations of message passing and allows for simple convergence testing after each round.
4.5.4 Decoding

After training has finished, the maximum likelihood parameters are written to file, which are used as input to decoding along with the events for the development or test set. This process uses max-product belief propagation to find the maximising label configuration for each sequence. Decoding uses a similar parallel implementation to that used for training – the observation sequences are split evenly between the processors. In decoding only a single processing iteration is required for each sequence, and the resulting predictions are written separate files (for each processor) in parallel. These files are then pasted together when all processors are finished. This resulting file contains the predicted label sequences which can then be used for further processing or scoring.

4.6 Conclusion

This chapter presented the experimental setup used to evaluate the accuracy and timing of CRF training and decoding algorithms. The following three chapters present a collection of existing and novel CRF training methods; the evaluation methods outlined here are used to compare these methods. Specifically, this chapter presented the following:

- Three tagging tasks which covered a range of different classical NLP applications: part-of-speech tagging, named-entity recognition and joint part-of-speech tagging and noun-phrase chunking. These tasks feature a variety of label set sizes and numbers of training instances in order to highlight different factors in the complexity of training and decoding.

- Evaluation measures for measuring the predictive accuracy and for measuring the training and decoding times.

- The software implementation of the CRF, including its high level design and implementation considerations.

The following chapter presents baseline results on these tasks using standard training and some existing approximation methods, while Chapters 6 and 7 show how training and decoding efficiency can be improved in two novel ways.
Chapter 5

An Empirical Comparison of CRF Training Methods

This thesis deals with the efficiency of CRF training, with the primary aim of scaling CRF training to larger tasks than are currently possible. Chapter 3 described the standard training method, which uses L-BFGS, a numeric optimisation algorithm (Nocedal, 1980), to optimise the log-likelihood of a fully observed training sample. This process requires repeated application of sum-product belief propagation (Pearl, 1988) to each graph in the training sample. The complexity of BP inference varies with the size of the graph, the number of features and the number of labels squared (in chains). For this reason training is often very slow for tasks with many features, a sizeable label set and/or a large training sample – these factors often preclude the use of CRFs.

This chapter provides an empirical study of CRF training and decoding times and accuracy on a number of natural language tasks. These tasks were described in Chapter 4 and were chosen to highlight the critical factors in the training complexity. This chapter also presents results using popular alternative training methods for CRFs, namely pseudo-likelihood training, piecewise training and averaged perceptron training (Besag, 1974, 1975; Sutton and McCallum, 2005; Collins, 2002a). We use the averaged rather than voted perceptron as both methods have been shown to achieve similar accuracy, while only the averaged perceptron can still be represented as a linear classifier, which simplifies decoding (Freund and Schapire, 1999). These alternative methods either approximate the true likelihood with a simpler objective function which can be evaluated quite cheaply, or else optimise a different loss function. The methods are described in detail in Section 3.7.

These three alternative training methods have been applied to CRFs in previous work with reported improvements in training time over standard maximum likelihood training (Sutton and McCallum, 2005). The cost of averaged perceptron training has not been explicitly compared to standard training, although the results in Collins (2002a) show that perceptron training can find good parameter estimates in only a small number of iterations, which they compare to a CMM which requires an order of magnitude more
iterations to converge. We would expect a CRF to train in a similar number of iterations to a CMM. Roark et al. (2004) show that the cost of an iteration of perceptron training is around two times greater than the cost of a CRF training iteration. Together, these reported results suggest that the perceptron training method should train in a significantly shorter time than standard CRF training.

Pseudolikelihood and piecewise training are both approximation methods which make restrictive assumptions about the training data. Despite this, Sutton and McCallum (2005) showed that piecewise training can outperform standard training on natural language tasks (although they also found that pseudolikelihood uniformly underperformed standard training). This is a surprising result and can be attributed to inherent smoothing in the piecewise decomposition, which can help reduce the extent of over-training. Collins (2002a) showed that voted or averaged perceptron training outperformed an equivalent CMM on two natural language tasks, suggesting that the method may be able to rival standard CRF training. Other studies have found the perceptron method provides only slightly poorer parameter estimates than standard CRF training (Roark et al., 2004; Vishwanathan et al., 2006).

This chapter analyses the efficiency and the accuracy of standard CRF training and of the three alternative training methods. The results also provide an empirical motivation for further work in scaling CRFs, as they demonstrate that approximate training methods can greatly reduce the training time, while also leaving room for better accuracy – these methods are often found to significantly under-perform standard training. The chapter is structured as follows. Section 5.1 presents the results for part-of-speech tagging. This includes the accuracy and time characteristics when trained on different amounts of data, highlighting many of the scaling properties of standard CRF training. Results for the named entity task are presented in Section 5.2 and the results from joint POS and noun-phrase chunking are shown in Section 5.3. Section 5.4 discusses the general findings over these three tasks, and Section 5.5 presents the conclusions of the chapter.

5.1 Part-of-speech tagging

The part-of-speech (POS) tagging task is described in Section 4.1. This task has a large label set (45 tags) and a comparatively large training set (45,000 sentences and one million tokens). For these reasons, the cost of standard training is very high – in fact, even representing the events from the full data set requires a massive amount of computer memory, while the training time is so slow as to be impractical. For this reason, we present results on a sub-sample of the training set, for which standard training is practical. Training alternative sequence labelling models, such HMMs and CMMs, can be much more efficient, which allows for the use of the full dataset.
5.1. Part-of-speech tagging

5.1.1 Characteristics of Standard Training

Before presenting the results for this task, we first examine the timing and accuracy characteristics of standard CRF training on this dataset. We measure the time and accuracy of standard training as the amount of training data is increased. Standard training uses sum-product belief propagation to calculate the log-likelihood within a L-BFGS numerical optimiser, as described in Section 3.6. We use a zero-mean Gaussian prior with variance $\sigma^2 = 10$ and measure the times on a Intel Xeon 3GHz machine with 4Gb of RAM.\(^1\)

The training set was repeatedly doubled, starting with 100 sentences and finishing with 6,400. Using larger amounts of training data would impose memory requirements beyond the capacity of a single machine and also lead to extremely large training times. The largest 6,400 sentence sample requires more than a day and 3GB of memory to train; these requirements would expand with larger sample sizes.

Training time

Figure 5.1 plots the training time against the training set size. The ‘growing features’ curve uses standard training where the feature set is attested on the training sample and therefore the feature set grows with the training set. The ‘fixed features’ curve shows the training time when the feature set is held constant (using the features from the largest 6,400 sentence set). Interestingly, the growing and fixed features curves are almost identical, which suggests that increasing the size of the feature set does not significantly affect the cost of training. This can be explained by the number of non-zero (active) features, which directly affect the cost of the many dot-products required during training. Figure 5.2 shows the total number of attested features and the average number of active features for each event. Although the total number of features grows quickly, the active features grows much more slowly. This is because many features incorporate contextual predicates which are only present in very few training instances.

The two remaining curves in Figure 5.1 show the training time when the solver is started with parameters from another trained model. The ‘PL seed’ uses the maximum pseudolikelihood parameters (with the same prior variance), and the ‘MLE seed’ uses the maximum likelihood parameters (i.e., using no prior). Seeding training with reasonable starting parameters is championed by Lafferty et al. (2001) as a way of speeding convergence, and contrasts with the standard approach of starting with a zero parameter vector. As the log-likelihood objective function is convex, the search process will find the globally optimal solution regardless of the starting point (subject to the convergence thresholds). However, starting close to the solution may allow the solution to be found in fewer iterations. If the seeds can be found cheaply and are close to the optimal parameters then this should result in a faster overall training time. Figure 5.1 indicates that this results in a modest proportional saving in training time for the pseudolikelihood seed.

\(^1\)This machine is a different specification to the standard machine used for all other experiments in the thesis, as described in Section 4.4.2. For this reason, the times presented in Section 5.1.1 should not be directly compared to times reported elsewhere.
Chapter 5. An Empirical Comparison of CRF Training Methods

Figure 5.1. Training time for part-of-speech tagging with varying sized training samples

Figure 5.2. Feature counts for part-of-speech tagging with varying sized training samples, showing the average number of active features per clique (left axis) and the total number of detected features (right axis).
Surprisingly, the MLE seed did not improve training time, even though one would expect these parameters to be close to the desired MAP parameters – at least, more so than the pseudolikelihood parameters, which will reflect a bias introduced by the pseudolikelihood assumption. Therefore this method cannot be considered as a generally applicable technique for scaling the model, but rather a simple heuristic only yielding a mild gain in some circumstances.

All the curves in Figure 5.1 are polynomial. This was confirmed by fitting a polynomial curve of the form, \( \log t = \alpha \log sz + \beta \), to the data using linear regression where \( t \) is the time and \( sz \) the number of training instances. This curve was fit to the ‘growing features’ data resulting in a low relative standard error (RSE) of 0.0792 and the polynomial power \( \alpha = 1.446 \). The PL seed data had a similarly good fit and a power of \( \alpha = 1.474 \). We would not expect the cost of each objective evaluation to grow more than linearly, as each evaluation has a fixed cost for each training instance (the cost of sum-product inference). The polynomial growth can be explained instead by the number of objective evaluations required by the solver. Figure 5.3 shows that the number of evaluations also grows with the number of training instances. This suggests that larger samples have more complex objective surfaces: each training instance presents novel combinations of features and the interactions between these features must be considered during training. This data can also be modelled accurately with a polynomial resulting in a power between 0.40 and 0.43. This power is dependent on the degree to which the feature set grows more dense as the features are instantiated over larger training samples. In summary, larger training samples lead to an increase in the number of evaluations and also an increased cost of each evaluation, which must perform inference over the additional instances. Together,
these two factors lead to a training time which is polynomial in the size of the training set.

**Accuracy**  Figure 5.4 shows the learning curve, which measures the accuracy on the development set as the training set is enlarged. Accuracy increases for each larger set, although these increases diminish with more data. This is as expected, given that a new training instance will provide much information to a data-impoverished model but will replicate a lot of existing information for a data-rich model. Therefore the marginal benefit of extra data will reduce with the size of the training set.

When run on the full training set of about 43,000 sentences, the CRF achieves a test accuracy of 97.03%. This required training over a large cluster with 40 CPUs and resulted in a total training time of 2,305,160 seconds – roughly a month. The best reported accuracy to date on this set is 97.15%, using the full training set of about 43,000 sentences (Toutanova et al., 2003), although this evaluated was on a different test sample.

**Decoding**  Figure 5.5 shows the decoding time as a function of the number of training instances. These times are all very fast relative to the training time, taking seconds rather than hours. As shown earlier in Figure 5.2 the number of features (active and total) grows with increasing training data. The increasing number of active features slows down max-product inference which requires many dot-products between the feature vectors and the model parameters. This is apparent in the slight increase in decoding time with each larger training set.

These short decoding times exclude the cost of instantiating the features of the test
5.1. Part-of-speech tagging

Figure 5.5. Decoding time for part-of-speech tagging where the models are trained on varying sized training samples, showing the raw time (left axis) and the time including the cost of event extraction (right axis).

set to create the event matrix (see Section 4.5), which dwarfs the actual time required for max-product inference. The “+Extraction” curve in Figure 5.5 shows the aggregate time which grows much more rapidly with the size of the training sample (and therefore number of features).²

5.1.2 Baseline Results

As described above in Section 5.1, the cost of training a CRF on the full training set is prohibitive in terms of memory and time requirements. Therefore a sub-sample of 5,000 sentences (about 11%) was used instead. Table 5.1 shows the results for standard training and for three approximate training methods. The models have been trained and the times recorded using the methods described in Section 4.4.2, namely with a fixed prior and fixed L-BGFS thresholds and history size (used to approximate the Hessian). The perceptron does not support a prior, and therefore the model predictions were evaluated after each iteration over the training sample and the parameters from the best performing iteration were chosen (measured on the development set). The time required to reach the given iteration is shown as the perceptron training time. This is a somewhat generous measure as it requires an oracle to know when to stop.

The accuracy results reported are the standard per-token accuracy as well as the per-sentence accuracy and per-token accuracy for unseen types, as described in Section 4.4.1. The prior for each method was individually optimised by hand on the development set,

²Note that the cost of creating the events has been omitted from training time estimates as this cost is trivial in comparison to the full cost of training.
resulting in prior variances of 5, 5, and 15 for the standard training (BP), pseudo-likelihood (PL) training and piecewise (PW) training, respectively.

The results in Table 5.1 show that the standard training (BP) results in the best score under every measure. The accuracy differences are all statistically significant with $p < 0.05$ using the matched-pairs test (see Section 4.4.3). Note that although some accuracy results look quite similar, this difference can be significant due to the large size of the test set coupled with the significance test being measured on the difference in the number of errors made per-sentence. Overall, the difference in results between the different methods is quite mild. The pseudolikelihood (PL) method achieved quite strong accuracy and in a very short time. Pseudolikelihood also requires less memory than the other methods, as it needn’t store the events for clique configurations which do not include some gold standard labels.

The training time for the standard method is two orders of magnitude greater than that of pseudolikelihood (PL) and the averaged perceptron (AP). The averaged perceptron produces strong results while only taking 15 epochs to reach peak accuracy. The piecewise estimator was faster than BP but was considerably slower than PL or AP. Piecewise training took a considerable number of objective evaluations before convergence: 335, compared to BP’s 229. Each evaluation is cheaper than BP because forward-backward inference is not required, however the slower convergence reduces the overall time savings. The piecewise estimator considers all possible labellings for each clique but it cannot leverage the interactions between different cliques in the same graph to describe the data. Instead it must individually optimise the likelihood over each clique (piece) – a considerably harder task, with a more complex objective surface.

The decoding times for each of these models were all approximately equal. Regardless of the training method, the same max-product (Viterbi) algorithm was run over the same events. Therefore decoding only differed in the model parameters used, which should not noticeably affect the runtime. The time required to generate the events for the development set was 198 seconds while decoding itself took 11 seconds.

<table>
<thead>
<tr>
<th>Method</th>
<th>Runtime (s)</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train</td>
<td>decode</td>
<td>token</td>
</tr>
<tr>
<td>BP</td>
<td>61,939</td>
<td>114.2</td>
<td><strong>95.99</strong></td>
</tr>
<tr>
<td>PL</td>
<td>819</td>
<td>118.2</td>
<td><strong>95.93</strong></td>
</tr>
<tr>
<td>PW</td>
<td>11,351</td>
<td>123.5</td>
<td><strong>95.18</strong></td>
</tr>
<tr>
<td>AP</td>
<td>303</td>
<td>118.5</td>
<td><strong>95.68</strong></td>
</tr>
</tbody>
</table>

Table 5.1. Timing and accuracy results for POS tagging, trained on a 5,000 sentence subset.

†These results are statistically significantly worse than belief propagation, $p < 0.05$. 
Perceptron training The perceptron was fitted to the development set by retrospectively choosing the best parameters, as described in Section 5.1.2. This ‘stopping condition’ was determined on the held-out data, in a similar manner to the fitting of the prior distribution in the other methods. The accuracy profile on the development set is shown in Figure 5.6, where the accuracy of the averaged perceptron and the standard perceptron are shown at the end of each epoch, or full pass over the training data. The averaged model shows much better stability in accuracy, while the standard perceptron accuracy varies wildly between iterations. Furthermore, the averaged perceptron always attains better accuracy than the standard perceptron, despite the standard perceptron better fitting the training data. The accuracy of the averaged perceptron peaks at epoch 12, after which is slowly degrades (the accuracy continues to fall beyond epoch 30). Accordingly, the parameters from epoch 12 were chosen.

5.2 Named Entity Recognition

The second task was named entity recognition (NER), as described in Section 4.3. This task has a small label set (8 labels) and a small training sample of about 14,000 sentences. CRFs can be easily applied to this task, training in only a few hours and with a modest memory footprint. Previous work has shown how CRFs can be used to good effect on the task (McCallum and Li, 2003; McCallum, 2003; Smith et al., 2005).

The results are shown in Table 5.2, using the same training and evaluation techniques as used for POS tagging in Section 5.1.2. The prior for each method was independently optimised for development $F_1$ score, resulting in variances of 100, 50 and 500 for the
standard training, pseudolikelihood training and piecewise training, respectively. The perceptron was stopped after 295 epochs, where the development score had levelled off, as shown in Figure 5.7. Interestingly, the perceptron did not reach its optimal values quickly, as in the POS tagging task. Instead it continued to improve over hundreds of epochs.

Roark et al. (2004) propose that perceptron training be used to seed the parameter values before standard training and also to prune the feature set. After a few rounds of perceptron training the parameters are sparse, with non-zero entries only for features which helped to correct an error during perceptron training. The sparsity can be used as a feature selection heuristic in standard training by pruning the feature set to include only the features used by the perceptron. The results from this method on the NER task are presented in Table 5.3, where standard training is seeded from the averaged perceptron parameters after five iterations or with the final averaged perceptron parameters. The table shows the results from pruning the feature set compared to the full feature set. The full set of 324,605 features was reduced to 78,754 or 93,398 features using the perceptron based pruning after iteration 5 or 295, respectively. Despite this large reduction in feature counts, the training time remained largely unchanged compared the baseline. This can be attributed to the increasing difficulty of the training task, which must describe the training data with fewer features. The converged perceptron parameters were only slightly better than the early estimates. The accuracy of the models with the unpruned feature sets should all be equal – the small variation is due to the L-BFGS convergence threshold terminating the numerical optimisation before true convergence. The results using the pruned feature sets are worse than the baseline, although not significantly so. This indicates that the perceptron method has successfully found a small core set of features, and pruned away many less informative features. When the same seeding technique is applied to the POS tagging and joint POS and NPC tasks, the same trend is observed – there no more than a 10% saving in training time, and when feature pruning is used, there is an insignificant loss in accuracy.

In NER the difference in $F_1$ scores of the four techniques is much more marked than for POS tagging. This is partly due to the metric used: the $F_1$ score is extremely sensitive to errors in the tagging, rewarding and penalising begin and inside label assignments much.

<table>
<thead>
<tr>
<th>Method</th>
<th>Train (s)</th>
<th>Decode (s)</th>
<th>Devel. $F_1$</th>
<th>Test $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>belief propagation</td>
<td>5,516</td>
<td>1.63</td>
<td><strong>89.93</strong></td>
<td><strong>82.91</strong></td>
</tr>
<tr>
<td>pseudolikelihood</td>
<td>615</td>
<td>1.66</td>
<td>85.90†</td>
<td>79.44†</td>
</tr>
<tr>
<td>piecewise</td>
<td>4,305</td>
<td>1.58</td>
<td><strong>87.91†</strong></td>
<td>82.05†</td>
</tr>
<tr>
<td>averaged perceptron</td>
<td>11,125</td>
<td>1.58</td>
<td>89.12</td>
<td>82.22</td>
</tr>
</tbody>
</table>

Table 5.2. Baseline timing and $F_1$ results for NER. †These results are statistically significantly different to belief propagation using the matched-pairs test, $p < 0.05$. 
Table 5.3. Results from seeding standard training with perceptron weights and feature sets, applied to NER. All models are regularised with a zero-mean Gaussian with variance $\sigma^2 = 10$.

<table>
<thead>
<tr>
<th>Perceptron iterations</th>
<th>Train (s)</th>
<th>Devel. $F_1$</th>
<th>Test $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>5,516</td>
<td>89.80</td>
<td>82.97</td>
</tr>
<tr>
<td>5 pruned</td>
<td>5,749</td>
<td>89.36</td>
<td>82.81</td>
</tr>
<tr>
<td>295 pruned</td>
<td>5,196</td>
<td>89.71</td>
<td>82.84</td>
</tr>
<tr>
<td>5 unpruned</td>
<td>5,826</td>
<td>89.79</td>
<td>82.96</td>
</tr>
<tr>
<td>295 unpruned</td>
<td>5,171</td>
<td>89.77</td>
<td>82.99</td>
</tr>
</tbody>
</table>

Figure 5.7. Averaged perceptron training curve for named-entity recognition.
more so than outside labels; these labels are all rewarded equally in the accuracy score. As before, standard training yields the best results, however these differences in results are not statistically significant in many cases. The differences between the development scores and the test scores of 6 to 7% reflect that the development set is an easier dataset that the test set, although the magnitude of the difference here is higher than for the CoNNL shared-task contestants of about 5% (e.g., Florian et al. (2003)). This may be due to the process of fitting the prior to the development set resulting in a poor choice of prior, which effectively over-fits the development data.

The pseudolikelihood result is by far the worst, with piecewise performing better, but still well clear of the best. The perceptron achieves a very strong result, but at the expense of a very slow training time. Pseudolikelihood training is extremely fast, taking a fifth of the time required for standard training. The magnitude of this saving is much lower in the NER task than for POS tagging (cf. Table 5.1). This is to be expected because the cost of pseudolikelihood estimation is linear in the label set, while standard training cost is quadratic in the label set. Therefore the difference in label set sizes between the two tasks exacerbates the difference in training time.

The decoding times on the development set are shown in Table 5.2, and are all roughly equivalent. The time required for feature instantiation and event extraction averaged 17 seconds.

5.3 Joint Part-of-speech Tagging and Chunking

The last task was joint POS tagging and noun-phrase chunking, as described in Section 4.2. We model the task using two layer lattice (or dynamic CRF; Sutton et al. (2004)), which allows joint inference over both layers of annotation. Exact inference over these lattices would be prohibitively expensive: we approximate that training would take at least three weeks and require an unreasonable 50Gb of RAM. In contrast, approximate inference using loopy BP required only 2.7Gb of memory and roughly two weeks of processor time. Loopy BP (Pearl, 1988) is advocated by Sutton et al. (2004) and Taskar et al. (2002) for inference in cyclic CRFs in preference to exact inference. We follow Sutton et al. (2004) and use loopy BP with a random message passing schedule. We limit the method to no more than 50 full passes over each graph (to avoid infinite loops) and use a convergence threshold of $10^{-4}$. In these experiments loopy belief propagation converged in most instances, and within 6 to 15 iterations.

**Layered Pseudolikelihood**  For this experiment we also applied a novel variation of the pseudolikelihood method (see Section 3.7.2) which makes a weaker set of independence assumptions. This method, *layered pseudolikelihood*, makes a pseudolikelihood assumption over layers of annotation (regular pseudolikelihood makes the assumption over individ-
ual nodes). The layered pseudolikelihood of a labelled training instance \((s, o)\) is given by:

\[
PLL(s|o) = \prod_l p(s_l|s_{\not\in l}, o)
\]

where \(l\) ranges over layers of random variables and \(s_{\not\in l}\) denotes the vector of all states not in layer \(l\). For the current two-layer task, this decomposes the joint distribution over both layers of POS and NPC annotation into two independent distributions:

\[
PLL(s|o) = p(s_{POS}|s_{NPC}, o) \times p(s_{NPC}|s_{POS}, o)
\]

where \(s_{POS}\) and \(s_{NPC}\) are the labellings of the POS and NPC layers, respectively. This produces an objective function of the form:

\[
O = \sum_{s, o} \tilde{p}(s, o) [\log p(s_{POS}|s_{NPC}, o) + \log p(s_{NPC}|s_{POS}, o)] + \log p(\Lambda)
\]

which can be optimised using the standard methods (e.g., L-BFGS), as described in Section 3.6. This objective has the same complexity as for linear chain, as each layer forms a chain and therefore supports exact forward-backward inference.

On the surface this looks similar to training two separate models in a standard cascade, where we first predict one layer of annotation, and then predict the next layer given the first, and so forth. However, the cascade makes a strong ordering assumption, such that each layer of annotation can only access the previous layers of annotation. In the layered pseudo-likelihood, each layer can access all other layers without regard to ordering. In addition, the whole system is trained as a single unit. This allows the training process to better learn the weights for features over edges connecting the two layers.

**Results** The results are shown in Table 5.4. The scores presented are POS tagging accuracy, noun phrase chunking (NPC) \(F_1\) score, and joint POS and NPC accuracy, as described in Section 4.4.1. For each method the prior was tuned to produce the best joint accuracy on the development set. The averaged perceptron’s joint accuracy on the development set is shown in Figure 5.8, which peaks early at epoch 20 after which the accuracy slowly declines.

These results show a similar trend to the previous tasks, despite the use of a different graphical structure. Loopy belief propagation was the best performing for all measures (by a significant margin in all but one case), but with a very slow run-time of about 13 days. The pseudolikelihood and perceptron were both much faster, taking just a few hours, while the piecewise estimator took 38 hours. The improved relative speed of the piecewise estimator is unsurprising, as its local normalisation avoids belief propagation over the cyclic graph, which often takes many iterations to converge. However, the piecewise approximator produces the worst test set accuracy. The perceptron is particularly attractive, as it reaches a good result quite early. Each epoch is quite expensive, requiring
Chapter 5. An Empirical Comparison of CRF Training Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Runtime (s)</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train</td>
<td>decode</td>
<td>POS</td>
</tr>
<tr>
<td>BP</td>
<td>1,177,747*</td>
<td>71.3</td>
<td>96.45</td>
</tr>
<tr>
<td>PL</td>
<td>8,920</td>
<td>77.3</td>
<td>96.02</td>
</tr>
<tr>
<td>PLL</td>
<td>168,384</td>
<td>77.4</td>
<td>96.16</td>
</tr>
<tr>
<td>PW</td>
<td>135,155*</td>
<td>61.0</td>
<td>95.58</td>
</tr>
<tr>
<td>AP</td>
<td>6,210</td>
<td>75.9</td>
<td>96.34</td>
</tr>
</tbody>
</table>

Table 5.4. Timing and accuracy results for joint POS tagging and noun phrase chunking.

*These times were measured for parallel 4-processor runs.

†These results are statistically significantly worse than belief propagation, $p < 0.05$.

Figure 5.8. Averaged perceptron training curve for joint POS tagging and noun phrase chunking.

loopy belief propagation for max-product inference. Despite this, the training time for the perceptron is impressively low.

Layered pseudolikelihood training achieves much better accuracy than standard pseudolikelihood, although at the cost of a two-day training time. The accuracy is quite near to that of belief propagation, and therefore the layered pseudolikelihood presents a useful and considerably cheaper alternative for training. As with the previous tasks, the piecewise estimator takes a large number of iterations to converge, so despite having a faster iteration time than the layered pseudolikelihood, it takes a similar time to train. The layered pseudolikelihood under-performs the averaged perceptron on the development data, but matches its joint accuracy on the test data. This could be due to the layered pseudolikelihood using a Gaussian prior which provides a more effective method of regularisation than the averaged parameters of the perceptron.
5.4. Discussion

Decoding times varied with each method for this task, which was largely due to the number of iterations required by max-product belief propagation. Table 5.4 shows the time required to decode the development set. These values are all similar and the decoding speed is acceptable for the 1,837 development sentences. The true cost of decoding includes the cost of event extraction: about 160 seconds for all methods.

5.4 Discussion

Over the three experiments, some general trends were apparent. Firstly standard training uniformly outperformed the other training methods. This occurred for both exact forward-backward inference (in chains) and approximate loopy BP (in lattices). Pseudolikelihood and piecewise training both optimise approximations to the true likelihood, and therefore we would expect these methods to find inferior solutions. The averaged perceptron is not strictly an approximation method, but instead optimises a different loss function, which may be less appropriate than the log-likelihood for our tasks. Alternatively the inferior accuracy of the averaged perceptron could be explained in terms of regularisation: the averaging of parameters is quite a blunt method of smoothing and cannot be tuned to the same degree as a Gaussian prior (or similar) used in maximum log-likelihood training. The poorer relative accuracy of the perceptron has also been observed in other empirical studies (Roark et al., 2004; Vishwanathan et al., 2006). The perceptron does not attempt to model the probability density; instead it optimises the error rate of decoding accuracy. Therefore in downstream applications which require probability distributions rather than maximum \textit{a posteriori} predictions, the perceptron may be of less use than the other (probabilistic) alternatives. The accuracy metrics here consider only the max-product decoding and therefore does not highlight this possible short-coming.

The pseudolikelihood estimate produced quite varied results, modelling the POS task well but performing quite poorly in the two remaining tasks. Pseudolikelihood training maximises the likelihood of each node label conditioned on the labels of its neighbours. This has the effect of over-weighting parameters over edges relative to parameters over single nodes (Vishwanathan et al., 2006). During decoding this can lead to clumping of errors, where what would normally be localised errors instead proliferate over the labelling. This effect has been observed for image denoising in computer vision where pseudolikelihood ‘over-smoothes’ the model to predict entirely black images (Vishwanathan et al., 2006). Figure 5.9 illustrates over-smoothing on the joint POS and NPC task. The figure shows a histogram over contiguous error sequences\textsuperscript{3} expressed as a proportion of the total error shown on a log scale. While both proportions fall sharply, the baseline model (BP) makes many fewer large errors than the pseudolikelihood model (PL). The distribution

\textsuperscript{3} An error in either layer of the lattice is classed as an error and therefore we can treat the two-layer lattice as a linear chain.
Chapter 5. An Empirical Comparison of CRF Training Methods

Figure 5.9. Error distribution comparing the clumping of errors under the pseudolikelihood model and the standard model.

of errors of the other approximate training methods were all much closer to the baseline distribution, i.e., they had a similar amount of error-clumping.

In POS tagging many words can only be tagged with a single label; coupled with the rich tag set and relatively limited set of transitions, this limits the degree of uncertainty over decoding pathways. Therefore errors resulting from pseudolikelihood training can only propagate to a limited extent. In contrast, chunking tasks feature a greater amount of uncertainty over labellings, where spans of words may be labelled as a chunk or just receive background outside labels. This leads to ambiguity over longer sequences of labels, and therefore a higher error-rate by the pseudolikelihood model. This is supported by observing that even in the joint POS and NPC task, the POS accuracy of the pseudolikelihood model is only slightly worse than the accuracy of the MLE, while its chunking score is considerably worse. Even decoding in a lattice configuration, the errors must propagate at different rates over the two layers.

The layered pseudolikelihood method performed very well on the joint POS and NPC task. This suggests that using a weaker pseudolikelihood assumption over larger graph fragments than nodes produces better models. Sutton and McCallum (2005) had similar results when optimising the pseudolikelihood over pairs of nodes (edges) in a chain, which rivalled the accuracy of standard training. Our layered pseudolikelihood is a natural fit for factorial CRFs, allowing each layer of annotation to be modelled with a linear chain.

Piecewise training produced quite disappointing results on all three tasks. This finding is at odds with the results of Sutton and McCallum (2005), who found (to their
surprise) that piecewise training out-performed standard training. We cannot explain why our results differ from theirs, although we are using different feature sets and testing the method on different corpora. Together the poor accuracy of pseudolikelihood and piecewise training suggest that decomposing CRF training into pieces might be harmful in general. This conjecture is also supported by the common under-performance of CMMs (another decomposed model) relative to CRFs (Lafferty et al., 2001; Klein and Manning, 2002; Collins, 2002b).

The training times varied considerably between tasks and between the different training methods. Standard training took about 5,000 seconds for NER, 60,000 seconds for POS tagging and more than a million seconds for the joint task. Despite the NER task having the largest training sample, its small tag-set and simple linear chain led to an acceptable training time. The larger POS tag-set noticeably increased the training time, while the lattice structure resulted in infeasible training times. These trends in training time across the three tasks are consistent with the asymptotic complexity of inference, which is quadratic in the size of the label set and linear in the size of the training sample.\footnote{The stated complexity applies to inference in chains or to a single round of loopy belief propagation over the lattice, of which there may be unboundedly many. In practise, loopy BP converged in 6 to 15 rounds in the majority of cases and therefore we can loosely equate the per-round complexity with the complexity until convergence.}

The approximate training methods were often considerably faster than standard training. Pseudolikelihood was between 5 and 200 times faster, which is unsurprising considering the gross simplifications introduced by the pseudolikelihood assumption. The layered-pseudolikelihood used in the factorial lattice was also much faster than standard training. The averaged perceptron trained very quickly for two of the three tasks, while also modelling all tasks quite accurately. The method was slow to peak for the NER task, for reasons which are unclear. In this task the accuracy of the standard perceptron continued to improve over hundreds of iterations, while for the other tasks it started overfitting the training sample very early. Piecewise training converged three to ten times faster than standard training. This method has a similar complexity to standard training in chains, but obviates the need for expensive sum-product inference. Therefore, the reduction in training time is unsurprising, although the reduction is less than expected. This is due to the objective surface for piecewise training being more complex, as interactions between different cliques in the same graph cannot be harnessed to describe the training data. (These cliques are partitioned into separate graphs which are assumed independent.) Therefore L-BFGS training takes many more iterations to converge than for standard maximum likelihood estimation. The piecewise estimator did produce its best relative run-time for the lattice task (joint POS and NPC), where its local normalisation avoided the costly belief propagation around loops.
5.5 Conclusion

This chapter presented an empirical comparison of popular CRF training methods over a number of natural language processing tasks. The experiments presented training times and accuracy for standard maximum likelihood training and three alternative methods: pseudolikelihood, piecewise and the averaged perceptron. The main findings were as follows:

1. The time required for standard training is polynomial in the size of the training sample. This is a result of an increase in the average number of active features due to an expanding feature set and therefore more costly dot products, and also because larger training samples present more difficult learning problems and therefore require more iterations of L-BFGS optimisation are required before convergence.

2. Seeding standard training with good starting parameters can reduce the training time, but only modestly. Interestingly, using a pseudolikelihood seed with the same prior was more effective than seeding with the maximum likelihood values. This is a similar configuration to Lafferty et al. (2001), who use the parameters of an CMM to seed CRF training.

3. Using standard training to maximise the data likelihood produces significantly better parameter estimates that any of the three alternative methods. In general, decomposing the training into independent pieces (in the pseudolikelihood and piecewise methods) noticeably harmed accuracy. This contradicts the findings of Sutton and McCallum (2005), who reported improved accuracy using the piecewise method.

4. The averaged perceptron produced quite good parameter estimates, and often in a very short time. The training time was not uniformly low across all tasks: for named entity recognition, the accuracy continued to improve over hundreds of iterations. Overall, the perceptron did not model the data as well as standard training, which was perhaps due to its loss function being less appropriate than the data likelihood or due to its less flexible method of regularisation.

5. The pseudolikelihood estimate dramatically reduced the training time, but often resulted in very poor parameter estimates. This was less of a problem for POS tagging than for chunking tasks. We propose that this difference is a result of POS tagging having mostly local ambiguity during decoding which is similar to the pseudolikelihood training configuration. For chunking there is considerably more uncertainty over sequences of labels, which are often incorrectly predicted by the pseudolikelihood model.

6. We introduced a novel variant of the pseudolikelihood method, which made the
5.5. Conclusion

pseudolikelihood assumption between layers in a factorial lattice (DCRF; Sutton et al. (2004)). This resulted in much faster training than for standard training using loopy BP, as inference instead used the faster forward-back algorithm. This method modelled the data quite accurately, although not as well as standard training. This decomposition was similar to the per-edge pseudolikelihood of Sutton and McCallum (2005), suggesting that weaker pseudolikelihood approximations may in general perform better than the standard per-node pseudolikelihood method.

Each of the alternative training methods deeply altered the learning problem, which may explain why their accuracy is consistently much lower than that of standard training. The following chapters introduce two novel training methods which can reduce the training time and its asymptotic complexity, which is critical for scaling. These training methods also endeavour to preserve the strong generalisation of standard training, unlike some of the approximation methods presented in this chapter.
Chapter 6

Scaling CRFs using
Error-Correcting Output Coding

Current methods for training CRFs scale poorly to large tasks, as demonstrated in previous chapters. For standard linear chain CRFs, maximum likelihood training has a time complexity quadratic in the number of labels, as described in Chapter 3. Large label sets are common in natural language processing (NLP) tasks, and therefore the complexity of training often makes it infeasible to apply CRFs to these tasks. Chapter 5 showed the time required for standard training on three separate tasks, clearly illustrating the trend of increasing training time with larger label sets.

Alternative training methods can reduce the training time and also reduce its complexity. This enables CRF training to be scaled to large label sets. This chapter presents such a method, which reduces training time and improves the scalability of training, while also preserving the strong predictive accuracy of standard training. This is achieved using error-correcting output coding (ECOC), a method which allows a complex multiclass learning problem to be decomposed into an ensemble of considerably simpler binary problems (Dietterich and Bakiri, 1995).

Error-correcting codes (ECCs) have traditionally been used for communication over a noisy channel. In this context, a sender wishes to communicate some information to a receiver over a communication link (channel) which may randomly corrupt parts of the message. Sending the message as a simple binary string may result in data corruption. For this reason some redundancy is added to the message before transmission. The redundant information allows the receiver to check to see if the message is well formed, or detect whether errors have occurred during transmission. Furthermore, this redundancy may also enable the receiver to automatically correct the errors and recover the intended message without resort to retransmission.

Dietterich and Bakiri (1995) introduced ECOC, a technique which uses ECCs to reduce a multiclass classification problem into a series of binary problems. In terms of traditional error-correcting coding, this is akin to replacing the noisy communication channel with
the predictions of a learning algorithm. The task of modelling the multiclass problem is split into many binary classification tasks as follows: First, class labels are encoded into binary strings, using an ECC. These encodings represent the labels in binary form, typically with some redundancy. Second, each bit of these binary strings are modelled with a separate binary classifier, which we call a constituent model. In decoding each constituent model makes a binary prediction; these predicted bits are combined to create a binary string. The string can then be resolved back to a label using standard ECC decoding techniques, which can be made resilient to small numbers of individual constituent model errors. In this manner ECOC allows binary learning algorithms to be applied to multiclass classification tasks, while also providing robustness through an error-correcting capability. Alternatively, the technique can be used with a multiclass learning algorithm, where it has been shown to significantly increase predictive accuracy (Dietterich and Bakiri, 1995; Kong and Dietterich, 1995; Berger, 1999; Ghani, 2000).

The technique can be understood as a form of lossy compression. Errors in the modelling process mean that we cannot hope for perfect lossless predictions. However, by choosing expressive binary encodings we can model the task accurately and thereby minimise the loss. Expressing encodings are often large ones and therefore the rate of compression is low. This has a direct bearing on training time, as a larger binary encoding will require more constituent models to be trained. At the other extreme, very simple encodings can provide high compression rates, however this comes at the cost of a higher loss in accuracy. Ideally, we wish to find a compromise between loss and compression rate (training time).

This chapter shows how classification using ECOC can be extended to structured labelling tasks such as sequence labelling. We encode each label in the label sequence into binary using an ECC. Each constituent model is a CRF, which models the joint distribution over a given bit – a binary structured labelling task. The predictions of these constituent CRFs are combined in decoding to produce a predicted label sequence which is resilient to errors by individual constituent models. This approach is particularly useful in that it allows the powerful ensemble method of ECOCs to be applied to CRF training. The constituent models are individually cheap to train because they model only binary problems. Coupled with a small code – one that encodes each label in only a few bits – this results in significant reductions in the cost of training. Many powerful ECCs have a high compression rate and therefore produce short binary encodings. These short codes allow for fast training, with considerably better scaling characteristics than standard CRF training. In addition, good short codes can be found with low loss, i.e., they can achieve similar accuracy to the standardly trained CRF.

This chapter is structured as follows. Section 6.1 provides the background by reviewing previous work on error-correcting codes, initially in the context of communication over a noisy channel, and subsequently with regard to classification. The subsequent
sections present novel contributions: Section 6.2 shows how ECOCs can be applied to sequence labelling with CRFs, presenting methods for training and decoding. Section 6.3 presents experimental results on each of the three benchmark tasks. A detailed analysis of questions raised from the earlier results, including further experimentation is presented in Section 6.4. Section 6.5 describes some potential problems with ECOC for CRFs and proposes some possible ways in which these could be addressed. Finally, Section 6.6 presents the conclusions of this chapter.

6.1 Background

Error-correcting codes have been long used for reliable and efficient communication over noisy channels. These codes trade some throughput over a transmission medium for an increased robustness to errors. This allows us to make efficient use of even the most unreliable communication media, without suffering from frequent data corruption. More recently, error-correcting codes have been applied to classification problems (Dietterich and Bakiri, 1995). This provides a number of advantages to a learning system, most notably providing a high robustness to errors. We briefly review error-correcting codes for communication and for classification before showing how CRFs can also make use of error-correcting codes.

6.1.1 Error-Correcting Codes for Communication

Error-correcting codes are traditionally used to minimise transmission errors over a noisy channel. This section reviews work dating back to the 1950s based on MacKay (2003, Chapter 1). There are many common situations where we wish to communicate over an imperfect channel; e.g., sending data over a telephone line, via a satellite, or simply writing and reading data on computer storage devices such as RAM, hard drives or CD-ROMs. In all of these cases there is some small probability that parts of the data may be corrupted by physical phenomena such as background noise, de-magnetism or scratches. We wish to insure against such data corruption and thereby avoid miscommunication between the sender and receiver. This is most easily achieved by adding redundancy to the information transmitted, which we call the source message. The redundant data is engineered such that partial corruption of the transmitted message – both the source and redundant parts – may be detected by the receiver. The receiver might be able to correct these errors by calculating the most probable valid message and data corruption pattern that could lead to the received data. When a solution is found which is considerably more probable than the next-best, this is solution is taken as the intended message. This process can be used to reduce the chance of miscommunication to negligibly close to zero.

Error-correcting codes (ECCs) formalise this method of adding redundancy, while allowing efficient encoding (computing the redundant data) and decoding (recovering
Chapter 6. Scaling CRFs using Error-Correcting Output Coding

Figure 6.1. Pictorial representation of the $H(7, 4)$ code, reproduced from MacKay (2003, Figure 1.13)

the intended message). For example, the Hamming $H(7, 4)$ code is a simple error-correcting code which takes blocks of $K = 4$ source bits and transmits $N = 7$ bits. This is a type of block code, as it divides the source message into a series of equal sized blocks. The code is illustrated in Figure 6.1, where $s = (s_1, s_2, s_3, s_4)$ is the source block, and $t_5, t_6$ and $t_7$ are the three redundant bits; together these form the transmitted message for the block $t = (s_1, s_2, s_3, s_4, t_5, t_6, t_7)$. The three redundant bits perform parity checks over three source bits, shown by the circles in Figure 6.1. These bits are set such that the sum over the four bits in each circle is even, e.g., $(s_1 + s_2 + s_3 + t_5) \mod 2 = 0$. The values for the parity bits can be calculated for every possible source block (any sequence of four bits), yielding a set of $2^4 = 16$ code words. For example, the code word for $s = 0011$ is $t = 0011100$. The resulting code words all differ from one another in at least three bits. This property means that if a code word, $t$, is corrupted with a single error, then the resulting message, $r$, remains closer to $t$ than any other code word. The closeness here is measured in number of different bits, called the Hamming distance. This enables the receiver to correct any single bit error. Formally, the error-correcting capacity is equal to $\left\lfloor \frac{H_{\text{min}} - 1}{2} \right\rfloor$, where $H_{\text{min}}$ is the minimum Hamming distance between any two code words. The $H(7, 4)$ code provides a transmission rate of $\frac{4}{7}$ with an error-correcting capacity of 1 bit.

There are many different types of error-correcting code, all of which seek to achieve a high transmission rate while also having a high error-correcting capacity. This allows the best utilisation of the channel, while remaining immune to transmission errors. These are competing aims and therefore a compromise must be found which is suitable to the user’s requirements.

6.1.2 Error-Correcting Output Coding for Classification

We now turn to the use of error-correcting codes in machine learning for multiclass classification (where there are $k > 2$ classes). Dietterich and Bakiri (1995) introduced a method for their use in machine learning, which they call error-correcting output coding (ECOC). This technique replaces the noisy communication channel with a learning algorithm and a training sample. The learning algorithm is used to separately model each bit of the output representation (the encoding of the classification label) and is used to predict these output
representations for novel instances. Flaws in the algorithm, its feature representation and noise in the training sample corrupt parts of the output representation, which can then be corrected in the standard way, e.g., using minimum distance resolution (Dietterich and Bakiri, 1995).

In practical terms, ECOC is a very effective way to render a multiclass classification task into a number of binary classification problems. This is particularly useful as it allows binary learning algorithms (e.g., connectionist and kernel algorithms) to be used on multiclass problems. Most commonly, these binary algorithms are used for multiclass classification using a one-vs-rest setup: where $k$ classifiers are trained with each classifier learning to discriminate between a single label and all other labels. The label predicted with the highest confidence is then chosen as predicted label. Dietterich and Bakiri (1995) showed how ECOC could be used to provide alternative binary decompositions which improve predictive accuracy. In addition to simply enabling a binary decomposition, Dietterich and Bakiri show how the technique can also yield improvements in accuracy over standard multiclass algorithms, due in part to its robustness to error.

ECOC requires a block code, $C$, a $M \times N$ matrix of \{0,1\}, where $M$ is the number of classes and $N$ is the number of columns, also called the length of the code. An example code for a four class classification problem is shown in Figure 6.2(a). This shows how each of the four class labels are mapped into binary strings of length $N = 8$; these encodings are referred to as codewords.

The training process learns a separate classifier for each column of the coding matrix (the individual classifiers will be referred to as constituent models). These constituent models learn to classify a single bit of the output representation. For instance, the first constituent model distinguishes between the labels person or location versus organisation or other, and the second constituent model distinguishes between person, location or other versus organisation. These two label subsets correspond to labels with codewords which have either zero or one in their first bit position, respectively. More formally, the constituent model for column $j$ learns to distinguish between the set of labels indexed by $\{i \mid C_{ij} = 1\}$ and its complement. This is illustrated in Figure 6.3, where the labels selected by first three constituent models from Figure 6.2(a) are shown with circles. Constituent models three and five in Figure 6.2(a) (shown in italics) model identical concepts, as they have the same column 0001. The eighth (last) constituent model has column 0000 (shown in bold) and therefore models nothing. It is also worth noting that this definition implies a symmetry: a column and its complement (flipping every bit) induce the same label subsets, and therefore the same classifier. For this reason the first and seventh constituent model are logically equivalent (codewords 0011 and 1100). A good code defines diverse sub-tasks, with each constituent model discriminating between very different subsets of the labels, and therefore resulting in low error correlation between the constituent models. A code with a high degree of separation between columns (af-
Chapter 6. Scaling CRFs using Error-Correcting Output Coding

<table>
<thead>
<tr>
<th>label</th>
<th>code word</th>
<th>label</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>01000110</td>
<td>person</td>
<td>0001001</td>
</tr>
<tr>
<td>location</td>
<td>01010010</td>
<td>location</td>
<td>0010010</td>
</tr>
<tr>
<td>organisation</td>
<td>10010100</td>
<td>organisation</td>
<td>0100100</td>
</tr>
<tr>
<td>other</td>
<td>11101000</td>
<td>other</td>
<td>1000111</td>
</tr>
</tbody>
</table>

(a) (b)

Figure 6.2. Example error-correcting output codes for \( M = 4 \) classes with \( N = 8 \) and \( N = 7 \) columns. The left code is considerably poorer than the right: the left contains trivial and duplicate columns, and has a minimum row separation of two bits, while the right has no redundant columns and a row separation of four bits.

\[ \text{person} = 001 \quad \text{organisation} = 010 \quad \text{location} = 100 \quad \text{other} = 111 \]

Figure 6.3. Graphical depiction of part of the code in Figure 6.2(a), showing the first three constituent models. The constituent models are shown as circles, indicating the labels selected for by that classifier.

Training proceeds with each constituent model trained on the same data set, using the binary label mapping defined by its column. We denote the set of constituent models as \( \Gamma = \{\gamma^1, \gamma^2, \ldots, \gamma^N\} \). These can be used for prediction as follows. Firstly classify a novel instance \( x \) with each constituent model, yielding an \( N \)-bit vector \( \Gamma(x) = \{\gamma^1(x), \gamma^2(x), \ldots, \gamma^N(x)\} \). Now compare this vector to the codewords for each label. The vector may not exactly match any of the labels due to individual errors in the constituent model predictions, and thus we chose the label \( i \) which minimises the distance \( \Delta(\Gamma(x), C_i) \). Typically this is the Hamming distance. For example, for the code in Figure 6.2, if the constituent models predicted 0100111, we would chose the label person, which has the closest codeword 0100100, differing in only two bits (compared to four, five and five bits for the remaining labels, respectively). This label is most consistent with the constituent models’ predictions. In this manner, the system can recover from individual prediction errors by the constituent models, provided the code has a good error-correcting capacity.

The constituent models may instead predict confidence measures or probabilities, rather than definite class assignments. In this case the scores can be normalised into the
interval $[0,1]$, and as before, we form a vector of the constituent models’ predictions. To resolve this vector into a label we use an alternative distance measure, such as the $L_1$ distance, $L_1(\Gamma(x),C_i) = \sum_{n=1}^{N} |\gamma_n(x) - C_{i,n}|$, which measures the distance between two real-valued vectors.

Dietterich and Bakiri (1995) applied ECOC to a number of common classification problems, using a variety of codes and two different classification algorithms. They found that applying ECOC to back-propagation and decision tree classifiers often yielded significant improvements in accuracy. Berger (1999) and Ghani (2000) applied ECOC to text classification using a Naive Bayes classifier, where they also observed some significant improvements in accuracy.

### 6.2 Error-Correcting Output Coding for CRFs

We now introduce a method which allows ECOC to be used with CRFs for sequence labelling. Sequence labelling is the natural setting for CRFs, where they are used to annotate each word in a sequence, although more generally CRFs can be applied to any structured labelling task, as described in Section 3.4.

Since the time (and often space) complexity of linear-chain CRF estimation is dominated by the square of the number of labels, it follows that reducing the number of labels will significantly reduce the complexity. ECOC recasts a multiclass problem into a set of binary problems, each of which is of lesser complexity than the full multiclass problem. Interestingly, training a set of binary CRF classifiers can be overall much more efficient than training a full multiclass model. This is because error-correcting CRF training reduces the $L^2$ complexity term to a constant. Decoding first predicts these binary labels from which it then recovers the ‘encoded’ label. This motivates the use of ECOC as an alternative method for training a CRF, thereby reducing the training time while also possibly benefiting from the improved generalisation of ECOC methods (Dietterich and Bakiri, 1995).

#### 6.2.1 Error-correcting CRF Training

Error-correcting output coding can also be applied in a straightforward manner to sequence labelling algorithms, such as CRFs, which are capable of multiclass labelling. ECOCs can be used with CRFs in a similar manner to that for classifiers, as described in Section 6.1.2. A block error-correcting code is used, with a row for each label while the number of columns is allowed to vary. A series of CRFs (constituent models) are trained, each on a relabelled variant of the training data. The relabelling for each constituent model maps the labels into binary space using the relevant column of the coding matrix, such that label $i$ is taken as a positive for the $n^{th}$ model example if $C_{i,n} = 1$, and negative otherwise. Thereby each constituent model describes the following joint distribution over
binary labellings:

\[ p_n(b|o) = \frac{1}{Z_n(o)} \exp\left( \sum_k \lambda_{n,k} F_{n,k}(b, o) \right) \]  

(6.1)

where \( b \) is the relabelled binary label sequence for constituent model \( n \):

\[ b = C_n(s) = (C_{s_1,n}, C_{s_2,n}, \ldots, C_{s_T,n}) \]  

(6.2)

\( \lambda_{n,k} \) are the parameters for constituent model \( n \), \( F_{n,k}(b, o) \) is an aggregate feature function over all cliques the graph and \( Z_n(o) \) is the normalisation function, which sums out the numerator in (6.1) over all binary labellings, \( b \).

The log-likelihood of a supervised training sample can be expressed using (6.1), and this quantity can be maximised in the standard manner (see Section 3.6). Each constituent model must be independently trained in this manner.

This is not the only way to model the distributions over binary sequences, as required by the ECOC method. A naive approach would be to model each bit with a separate classifier, however in order to represent the Markov dependencies between labels, we must model groups of bits. In our approach a binary CRF models the distribution over labellings for a given bit of each label’s encoding, which can readily account for such Markov dependencies.

The constituent model joint probability in (6.1) describes a standard CRF conditional distribution, where the label set for each node is \( \{0, 1\} \). This allows features over cliques which combine contextual predicates over arbitrary aspects of \( o \) and a function of the binary labelling of the clique. In a linear chain, the pairwise cliques can therefore detect the labellings \( \{00, 01, 10, 11\} \). This effectively ‘ties’ many of the features of the multiclass CRF; i.e., pairs of features which both detect clique labellings with the same binary encoding.

Training with a binary label set reduces the time complexity for each training iteration to \( O(NTF^2C) \), where \( N \) is the number of labelling training instances, \( T \) is the average size of these graphs, \( F \) is the average number of active features and \( C \) is the maximal clique size. This compares with the complexity of multiclass training of \( O(NTFLC) \). For chains the maximal clique size is two, and therefore the final \( 2^C \) term in the complexity of the binary model is replaced by a constant. Provided the code is relatively short (i.e., there are few constituent models), this translates into considerable time and space savings. Linear blocks usually encode the label into \( \lceil \log_2 L \rceil \) bits, and then provide a number of parity checks over these bits (MacKay, 2003). The number of redundant parity checks is typically low, and is often proportional to the number of information bits. This leads to codes with length \( O(\log L) \). These short codes are ideal for scaling purposes, providing overall time complexity of ECOC training which is sub-linear in the size of the label set. However, these short codes may not be optimal for ECOC modelling, where error rates and error correlations are a complex function of the code design.
6.2.2 Code Design

Traditional codes from communication theory do not necessarily make good codes for ECOC. Communications codes attempt to minimise the code length (and therefore maximise transmission rate) for a given error-correcting capacity by ensuring a good row separation. A good row separation is also desirable for ECOC, where it also provides good error-correcting capacity during decoding. However, in ECOC the number of rows, or code words, is not always a power of two, as is assumed for communication codes. This means that only a subset of the code words can be used, which often reduces the error-correcting capacity of the code. In communication individual bit errors are commonly assumed to be independent. However, the errors made by individual constituent models are often highly correlated: the constituent models are all trained on the same training sample and often model quite similar tasks. This can lead to a clumpy error distribution, where many errors tend to occur together, and therefore the errors cannot be easily corrected by the code. The experiments in Section 6.4.2 found a high error correlation, especially between very similar constituent models. For the error independence assumption to apply, the code must be designed to ensure diversity between constituent models. One way to achieve this is to use a code with well separated columns, measured in Hamming distance after considering logical inversion (Dietterich and Bakiri, 1995). Another way to ensure diversity is to ensure the constituent models use different feature sets (Ricci and Aha, 1998). The column separation criterion is unique to ECOC, and while traditional codes guarantee good row separation, and therefore error-correcting capacity, no guarantees are made about the properties of their columns.

Previous work on ECOC for classification used block codes motivated from coding theory, such as BCH codes, and also exhaustive and random codes (Dietterich and Bakiri, 1995; Berger, 1999; Ghani, 2000). Many communication codes contain identical columns, or columns which differ in only a few bit positions. For this reason, these previous ECOC studies also used other types of code. Exhaustive codes have been used where feasible (for small numbers of classes). These codes contain every unique column, after removing the trivial columns (all zeros or all ones), yielding $2^M - 1$ columns. This code has pairs of columns which differ in only a single bit, and therefore the error correlation is significant. The code in Figure 6.2(b) is the exhaustive code for $M = 4$, with the full $N = 2^3 - 1 = 7$ columns. Any columns added to this code must be either trivial or equivalent to an existing column. For larger $M$, where the exponential length of the exhaustive code rendered them impractical, random codes were used, which sampled columns from the exhaustive code. Random codes were shown to have good expected column and row separation for large $M$ (Berger, 1999). Dietterich and Bakiri (1995) also used a randomised hill climbing method to find codes with both good row and good column separation.
Chapter 6. Scaling CRFs using Error-Correcting Output Coding

Figure 6.4. Example error-correcting output coding decoding using the standalone and marginals methods for a sequence of length four, using seven constituent models. The predictions of the constituent models are shown as column vectors where the standalone method uses Viterbi predictions and the marginals method uses the marginal probability of a 1 label. The labels have been resolved against the error-correcting code in Figure 6.2(b).

6.2.3 Error-correcting CRF Decoding

Training an error-correcting ensemble of CRFs is a logical extension of the ECOC classifier method to sequence labellers, however decoding can be more complex. This is largely due to the structured labellings modelled by the CRFs. This study has used three different decoding strategies, which are described below.

**Standalone**  The standalone method requires each constituent model to find the Viterbi path for a given sequence, yielding a string of zeros and ones, one per constituent model. These predictions are combined into a matrix where the predictions of each constituent model are columns. An example prediction matrix is shown in Figure 6.4(a), containing the predictions of seven constituent models on a graph with four nodes. The rows are resolved to labels by comparing each row’s bit string to each of the label codewords, and the label is chosen with the minimum Hamming distance. For the example in Figure 6.4(a) this results in the first row predicting an organisation label when using the error-correcting code from Figure 6.2(b). This label’s codeword differs from the bit-string in only two bit positions, while the person and location codewords differ in four. Note that in this instance there is a tie, as the other label’s codeword is equidistant at two bits. Ties are broken arbitrarily.

The standalone method allows for errors to be corrected at each site, however it discards information about the uncertainty of each constituent model, instead only considering the most probable paths. CRFs are probabilistic models, and their uncertainty over the labelling can be useful for decoding; this is ignored by the standalone method. Moreover, the error-correction decisions for each label are made independently. It might be better instead to jointly make error-corrections over the entire labelling.

**Marginals**  The marginals method of decoding uses the marginal probability distribution at each position in the sequence instead of the Viterbi paths. This distribution is easily computed using the sum-product (forward-backward) algorithm. The decoding
6.2. Error-Correcting Output Coding for CRFs

proceeds as before, however instead of a bit string we have a vector of probabilities, i.e., 
\((p_n(S_1 = 1), \ldots, p_n(S_T = 1))\). This vector is compared to each of the label codes using the \(L_1\) distance, and the closest label is chosen. The method is illustrated in Figure 6.4(b), where the probability vectors are shown as columns, and label resolution is performed on each row.

The marginals decoding method was also used in Dietterich and Bakiri (1995). This method incorporates the uncertainty of the binary models, however it does so at the expense of the path information in the sequence. In addition, it performs label resolution independently, rather than jointly.

**Product** Both of the standalone or marginals decoding methods perform label resolution at each site in the graph independently. There may be some gain in jointly predicting the full labelling, and therefore finding a full consistent labelling. The **product** decoding method addresses this problem. It does so by finding the state sequence which maximises the following distribution:

\[
p(s|o) = \sum_b p(s, b|o) = \sum_b p(s|b)p(b|o)
\]  

(6.3)

where \(b\) is a binary string which encodes a full labelling. To allow tractable search, we simplify (6.3) by assuming that the sum over \(b\) ranges only over the space of encodings for labellings, i.e., non-codewords are assigned zero probability. Therefore, the distribution \(p(s|b)\) can be assumed deterministic, where \(p(s|b) = 1 \) if \(b\) is the encoding of \(s\) and zero otherwise. Applying these assumptions to (6.3) yields:

\[
p(s|o) = p(C(s)|o)
\]  

(6.4)

where \(C(s)\) is the binary encoding of the label sequence using the code \(C\). This encoding forms a two dimensional binary matrix, with a row for each encoded label and a column for each constituent model.

In order to calculate the probability of a bit sequence in (6.4), we treat each constituent model as an independent predictor of its bits in the sequence, such that (6.4) can be re-expressed as the product \(p(C(s)|o) = \prod_{n=1}^{N} p(C_n(s)|o)\), where \(C_n(s)\) takes the \(n^{th}\) bit of each label’s encoding in the sequence (the \(n^{th}\) column of the \(C(s)\) matrix). When combined with (6.4), this yields the following probability distribution over labellings:

\[
p(s|o) = \prod_{n=1}^{N} p_n(C_n(s)|o)
\]  

(6.5)

\[
= \frac{1}{Z'(o)} \exp \left( \sum_n \sum_k \lambda_{n,k} \sum F_n,k(C_n(s), o) \right)
\]  

(6.6)

where the constituent model probability distributions, \(p_n\), are defined in (6.1), and \(Z'(o)\) is the partition function, which sums out the numerator in (6.6).
Equation (6.6) has the form of a standard multiclass CRF, although with a non-conventional set of features. The probability of a full multiclass labelling, $s$, can be maximised using the max-product (Viterbi) algorithm. Note that the partition function, $Z'(o)$, need never be evaluated during this maximisation. The product decoding uses an equivalent formulation to a uniformly weighted logarithmic opinion pool of CRFs (Smith et al., 2005), an instance of the product of experts method which averages an ensemble of distributions using a geometric mean (Heskes, 1998; Hinton, 1999). This method is particularly compelling for combining log-linear distributions in an ensemble, because the product distribution has the same log-linear form.

Of the three decoding methods, standalone and marginals have the lowest complexity, requiring sum-product or max-product inference over the binary labellings for each constituent model followed by a label resolution step. Product decoding, however, requires Viterbi decoding with the full label set, and with many features: the union of the features of each constituent model. When the label set is very large, the cost of full multiclass decoding – with time complexity proportional to the square of the label set for chains – can be quite computationally demanding. We have used all three methods for decoding and compare their relative efficiency and accuracy.

### 6.2.4 Expected Results

Before presenting the experiments, we will first state some our initial expectations with regard to the accuracy and efficiency of ECOC training and decoding. Firstly, we expect that the overall training time will be proportional to the code length. A long code has many columns and therefore requires the training of many constituent models. The training time for each constituent model will depend on the difficulty of its task: for simple tasks with few decision boundaries, the training process should converge in fewer iterations than for more complex tasks.

The design of the error-correcting code will be a critical factor in the accuracy of the ensemble. The accuracy should correlate with the code’s row and column separations. We also predict that column separation is a good indicator of error correlation. The code’s error-correcting capacity will greatly affect the standalone and marginals decoding methods, which explicitly perform error-correction. The effect may be less clear with product decoding, which does not strictly perform error-correction, but instead uses the code for smoothing.

When using very long codes, such as large random codes or the exhaustive code, there will be high error correlation. This will limit the accuracy of the method. If we start with a small code and progressively enlarge it with extra columns, we should see an asymptotic accuracy profile: accuracy improvements will be initially quite large but will abate for longer codes. This is a result of the positive effect of better modelling decision boundaries which is countered by increasing error correlation between constituent models.
Finally, we expect that the ECOC ensemble will provide additional regularisation over standard training. The ensemble redundantly models many of the decision boundaries, and therefore will reduce the effect of noise. This effect could lead to better overall accuracy, or at least comparable accuracy to standard training.

6.3 Experiments

The ECOC training and decoding methods were tested on the tasks of named entity recognition (NER), part-of-speech (POS) tagging and joint POS tagging and noun-phrase chunking (NPC). These tasks are all described in Chapter 4, along with the feature representation and graphical structure used for each task. Of these tasks, NER represents those tasks on which CRFs can be simply and efficiently applied, while POS tagging and POS/NPC are both very difficult, with extremely long training times (which is also coupled with high memory demands).

The results are shown in Tables 6.1, 6.2 and 6.4, which compare the baseline multiclass CRF to the ECOC CRF on the three tasks. These results tables show the training times for the multiclass and ECOC CRF and the decoding times and accuracy for each decoding method. We now present the results on these tasks, flagging some findings which are to be analysed in more detail in Section 6.4.

In each task the results from two different codes are presented. The first code is a one-vs-rest code, in which each constituent model describes the distribution over a single label. This code is the \( L \times L \) identity matrix. This code should result in fast training times for two reasons: (1) the code is quite short (linear in the label set), and (2) the code specifies particularly simple decision boundaries. The second code was chosen to be much longer, to contrast with the one-vs-rest code. As the NER task had a very small label set (8 labels), it was possible to use an exhaustive code with the full complement of \( 2^8 - 1 = 127 \) unique columns. This code contains every unique column, and therefore represents the largest code. (Its exponential size in the label set means that the exhaustive code can only be used for tasks with tiny label sets, and is used here for illustrative purposes.) The POS and POS/NPC tasks had much larger label sets, and therefore a random code with 100 columns was used instead. This length was determined empirically as a good compromise between efficiency and accuracy, as shown in Section 6.4.2.5. This code was randomly initialised and constrained to avoid any equivalent rows or columns.

**NER** The NER task had the smallest label set of the three tasks, with only 8 labels. The results for this task are shown in Table 6.1, showing both MLE and MAP \( F_1 \) scores. MAP training used a Gaussian prior with zero mean and variance of 10 for each constituent model. The simple one-vs-rest model trains in a total time of about a quarter of the baseline multiclass time, while the exhaustive is six times slower than the baseline. The
<table>
<thead>
<tr>
<th>Model</th>
<th>Train (s)</th>
<th>Decoding</th>
<th>Dev. $F_1$</th>
<th>Test $F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Decoding Time (s)</td>
<td>MLE</td>
<td>MAP</td>
</tr>
<tr>
<td>Multiclass</td>
<td>5,516</td>
<td>-</td>
<td>19</td>
<td>87.33</td>
</tr>
<tr>
<td>Exhaustive code</td>
<td>32,827</td>
<td>standalone 960*</td>
<td>88.20</td>
<td>89.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 1,069*</td>
<td>88.39</td>
<td>89.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 1,389*</td>
<td>89.12</td>
<td>89.94</td>
</tr>
<tr>
<td>One-vs-rest code</td>
<td>1,388</td>
<td>standalone 147</td>
<td>81.74</td>
<td>82.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 129</td>
<td>87.32</td>
<td>89.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 148</td>
<td>88.34</td>
<td>89.68</td>
</tr>
</tbody>
</table>

Table 6.1. Results for named-entity recognition, showing training and decoding times and the $F_1$ scores on the development and testing sets. All ECOC constituent models were regularised with a zero-mean Gaussian prior with $\sigma^2 = 10$.

*These results are significantly different to the baseline multiclass prediction using the matched-pairs test at $p < 0.05$, as described in Section 4.4.3.

The decoding code is impractical for all but tasks with extremely small label sets, as it incurs not only a high training time but also a high decoding time. The small label set means that the baseline multiclass training is quite fast; it is surprising that the short one-vs-rest code can do better.

The decoding times of the ECOC ensembles are all considerably greater than the baseline, reflecting the increased overhead of combining the predictions over the ECOC ensemble. The large exhaustive ECOC ensemble required parallel decoding to share the memory burden over two machines (about 5Gb). There is an obvious optimisation which was not implemented and may reduce the memory demands: the features for each constituent model were instantiated independently on each instance, despite the fact that they all applied the same observation tests. If this redundant step was performed only once, the decoding time and memory demands would both improve.

These results show that error-correcting CRF training achieves quite similar accuracy to the multiclass CRF. Marginals and product decoding were statistically tied, but both significant outperformed standalone decoding. This suggests that there is uncertainty in the predictions of each constituent model, which is being utilised effectively by the marginals and product methods. Both ECOC ensembles showed some significant improvements over the standard multiclass model when trained with no prior (MLE). When using product decoding, the feature set of the ECOC models and the multiclass models define equivalent CRFs with different weight vectors. In this setting ECOC training decomposes standard multiclass training into a series of separate training steps. The ECOC-learned parameters fit the training set less well because they do not maximise the

---

1 As described in Section 4.4.1 we use the term *accuracy* in its general sense of test set performance, which in this case is measured by the $F_1$ score.
6.3. Experiments

full data likelihood, i.e., the model is implicitly regularised. The net effect is provides a better generalisation capability, which confirms our earlier prediction in Section 6.2.4.

The one-vs-rest ECOC CRF is particularly sensitive to the decoding method, producing strong results only with the product decoding. This is a consequence of it having no error-correcting capacity; its strong accuracy with product decoding comes as a surprise. This result is analysed in detail in Section 6.4.2.1.

The final MAP results are also worthy of note. These show that with product decoding, both the one-vs-rest and exhaustive code significantly outperform the baseline multiclass CRF. This pleasing result can be explained by three factors: 1) more effective regularisation in training, 2) a reduction in bias and/or variance due to the ensemble setup (Kong and Dietterich, 1995); or 3) a more expressive feature set. The first and second reasons stem from the use of a variety of diverse independently trained models of the data. With the exhaustive ECOC the decision boundaries are learnt many times (by different constituent models), and therefore modelling errors in individual constituent models can be smoothed by others in the ensemble. With the one-vs-rest ECOC the decision boundaries are each learnt only twice each; however, even in this case the ensemble method still improves the accuracy. The last reason – that the ECOC CRF uses a more diverse feature set – requires further explanation. The product CRF used for decoding employs as its feature set the union of the features in each constituent model. These constituent models tie together features which are separated in the multiclass CRF. For instance, the contextual predicate $o_t = 'Mr.$ might be combined with $s_t = \text{person}$ and $s_t = \text{other}$ to produce two features in the multiclass CRF. In an ECOC CRF, this predicate can be included in each constituent model with a different label condition, such as $o_t = 'Mr.' \land s_t \in \{\text{person}, \text{location}\}$. These tied versions of the multiclass features may provide some smoothing of the ECOC ensemble.

The exhaustive code was shown here for illustrative purposes – it is too large for practical use, except for tasks with very small label sets. Many columns in this code were unnecessary, yielding only a slight gain in accuracy over much simpler codes while incurring a very large increase in training time. Therefore by selecting a subset of the exhaustive code, it may be possible to reduce the training time while preserving the strong accuracy. A variety of shorter codes are applied to NER in Section 6.4.2; some of these maintain a high accuracy while training in a very short time. The exhaustive code is interesting in itself, as this code is the largest possible random code, and therefore provides an approximate ceiling on the expected accuracy. In terms of the compression analogy, the exhaustive code is the most ‘loss-less’, while shorter codes will be more lossy but will also be faster to train. The much shorter one-vs-rest code still managed

---

2The marginals results for both the exhaustive and one-vs-rest codes significantly outperform the baseline when measuring accuracy, despite having lower accuracy. The significance testing use per-site errors (i.e., accuracy), leading to this abnormality. In the clear majority of cases accuracy and $F_1$ score induce identical rankings.
Table 6.2. Part-of-speech tagging results on the small training sample

<table>
<thead>
<tr>
<th>Model</th>
<th>Train (s)</th>
<th>Decoding</th>
<th>Decoding Time (s)</th>
<th>Acc. Devel</th>
<th>Acc. Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiclass</td>
<td>61,939</td>
<td></td>
<td>186</td>
<td><strong>95.99</strong></td>
<td><strong>95.99</strong></td>
</tr>
<tr>
<td>Random code 100</td>
<td>9,296</td>
<td>standalone</td>
<td>1093</td>
<td>95.34</td>
<td>95.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals</td>
<td>2782</td>
<td>95.41</td>
<td>95.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product</td>
<td>19,695</td>
<td>95.45</td>
<td>95.30</td>
</tr>
<tr>
<td>One-vs-rest code</td>
<td>1,113</td>
<td>standalone</td>
<td>391</td>
<td>94.16</td>
<td>94.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals</td>
<td>423</td>
<td>95.74</td>
<td>95.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product</td>
<td>8,310</td>
<td>95.74</td>
<td>95.66</td>
</tr>
</tbody>
</table>

strong results despite its comparatively small size and lack of error-correcting capacity. This suggests that other short codes may exist which provide strong accuracy and short training times.

**POS tagging** The POS tagging results are shown in Table 6.2, trained on the 5,000 sentence set. These results compare the baseline multiclass CRF to the ECOC method using a random code of length 100 and a one-vs-rest code (with 45 columns – one per label). The constituent models were trained using a zero mean Gaussian prior with variance of 10. The baseline multiclass model uses a zero-mean Gaussian prior with variance 5, the setting with the best results on the development set.

The training times for the ECOC methods are 2-15% of the training time for the baseline. These are substantial reductions; the one-vs-rest code reduces the training time from almost a day down to twenty minutes. The relative improvement in ECOC training time over the baseline is larger than seen in the NER task due to the much larger POS label set. The time cost of standard training varies as the square of the number of labels. In contrast, the cost of ECOC training varies with the length of the error-correcting code. The one-vs-rest code has one column for each label, and therefore the cost of one-vs-rest ECOC training is linear in the number of labels. As described in Section 6.2.1, there are classes of smaller codes which have length sub-linear in the size of the label set, which present even better scaling characteristics.

The decoding times are slower than those for the multiclass baseline, with best times from the one-vs-rest code with standalone and marginals decodings. The increased decoding times could be ameliorated to a small degree by code optimisation. The product decoding is particularly slow because it must perform Viterbi inference over the full multiclass lattice, as does the multiclass CRF, but the ECOC model has many more dense features. Therefore the dot products will be slower, which will increase decoding time.

The POS accuracy results all significantly under-perform relative to the baseline (multiclass). Despite this the results are still respectable relative to the alternative training methods presented in Section 5.1, particularly for the one-vs-rest code with product de-
6.3. Experiments

coding. The under-performance may be due to the lack of intensive prior hyper-parameter search for each of the ECOC constituent models. In these experiments, all constituent models use the same prior. Decoding accuracy may be improved by fitting the prior to each constituent model individually. However, ensemble methods benefit from diversity between the constituent models; the diversity is often reduced by rigorous smoothing, and therefore smoothing can harm the ensemble accuracy (Brown et al., 2005).

To test how smoothing affects the constituent models accuracy, we individually fitted a zero-mean Gaussian prior to each of the models in the one-vs-rest ensemble. This resulted in a result of 95.81%, which is marginally better than 95.74% using a fixed variance of 10. We also tried fitting a single prior to the whole ensemble, which gave the same improvement in accuracy (with a variance of 5). This was surprising, as we expected individual regularisation to improve accuracy. The finding suggests that many of the constituent model prediction errors which are due to over-fitting can already be corrected automatically during decoding.

The POS accuracy results show a similar trend to the NER results: the ranking of decoding methods remain the same, although the accuracy difference between the three methods is reduced. This could be due to a reduction in constituent model uncertainty (and therefore the gain from marginals or product decoding is lessened), or simply a result of the use of accuracy as the performance measure in place of NER’s more punishing $F_1$ score. The one-vs-rest ensemble is more accurate than the random one, which is surprising considering the random code is much larger, and has an error-correcting capacity of 16 bits compared to zero for the one-vs-rest code. This is consistent with the strong one-vs-rest results in NER task, and is analysed further in Section 6.4.2.1. The random code here is much shorter than the exhaustive code, which would have $1.8 \times 10^{13}$ columns. We do not advocate such an intractable code, however we propose that larger random codes or more carefully crafted short codes might yield stronger accuracy.

Full POS Tagging The scaling benefits of the ECOC technique also allowed training on the full POS training set of about 45,000 sentences. This is the first serious application of CRFs to POS tagging, due largely to the high cost of standard training. The results on this data set are shown in Table 6.3. Here the random code had 200 columns and an error-correcting capacity of 38 bits. Note that even with such a large random code, the one-vs-rest code still achieves the top performance. Two algebraic codes were also used: the Hamming code $H(10,6)$ with an error-correcting capacity of one bit, and the BCH code $B(31,6)$ with an error-correcting capacity of seven bits. While the very short $H(10,6)$ code trains faster that the one-vs-rest, its performance is considerably lower. This is unsurprising because the Hamming code has many fewer columns, coupled with a low error-correcting capacity. The BCH code fares better, with reasonable performance, and an acceptable training time. Here, once again the one-vs-rest method surprisingly
### Table 6.3. POS tagging accuracy on the full training sample.

*Run on a cluster of many CPUs to reduce per-machine memory consumption. Here, the total elapsed time over all CPUs is reported. †These results are significantly different to the multiclass baseline at $p < 0.05$."

<table>
<thead>
<tr>
<th>Model</th>
<th>Train (s)</th>
<th>Decoding Time (s)</th>
<th>Acc. Devel.</th>
<th>Acc. Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiclass</td>
<td>2,305,160</td>
<td>-</td>
<td>96.73</td>
<td>97.03</td>
</tr>
<tr>
<td>Random code 200</td>
<td>437,834</td>
<td>standalone 3,588*</td>
<td>96.51†</td>
<td>96.63‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 3,668*</td>
<td>96.56†</td>
<td>96.70‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 41,517*</td>
<td>96.59†</td>
<td>96.72‡</td>
</tr>
<tr>
<td>One-vs-rest code</td>
<td>21,429</td>
<td>standalone 512</td>
<td>95.66†</td>
<td>95.91‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 630</td>
<td>96.76</td>
<td>96.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 8,028</td>
<td>96.76</td>
<td>96.93‡</td>
</tr>
<tr>
<td>$H(10,6)$ code</td>
<td>22,162</td>
<td>standalone 127</td>
<td>94.88†</td>
<td>95.05‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 133</td>
<td>95.58†</td>
<td>95.78‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 1,724</td>
<td>95.77†</td>
<td>96.01‡</td>
</tr>
<tr>
<td>BCH(31,6) code</td>
<td>72,081</td>
<td>standalone 424</td>
<td>96.05†</td>
<td>96.11‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 442</td>
<td>96.23†</td>
<td>96.25‡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 5,159</td>
<td>96.30†</td>
<td>96.25‡</td>
</tr>
</tbody>
</table>

outperforms the more robust BCH code.

These timing and performance results compare favourably to the multiclass CRF, which required many gigabytes of memory and a cluster of 40 CPUs to train. In contrast, training each of the constituent models required at most 835Mb of RAM, which is well within the capabilities of a single machine. The accuracy of the one-vs-rest code is insignificantly different to the multiclass when using marginals decoding, despite taking an order of magnitude less time to train.

### POS tagging and NPC

The results on the task of simultaneously POS tagging and noun phrase chunking (NPC) are shown in Table 6.4. This shows a 100 column random code and a one-vs-rest code. As the task has two layers of annotation – 3 chunk tags and 45 POS tags – the codes were designed to cover both layers. This was achieved for the one-vs-rest code by first creating a one-vs-rest code for each layer, and then combining every pair of columns from the two codes to form the final code. This resulted in 135 constituent models: one for every pair of labels in $S_{\text{POS}} \times S_{\text{NPC}}$, where $S$ are the two label sets. This manner of construction can yield very large one-vs-rest codes: its size is exponential in the number of layers. For this reason, random or algebraic codes would be recommended instead for general tractability.

The constituent models were trained on the relabelled training data, where all POS labels are mapped using the POS portion of the constituent model’s column, and the NPC labels are mapped using the NPC portion. This is similar, but not identical, to the
result of projecting the two layers into a single chain, in which case the label set is the Cartesian product $S_{POS} \times S_{NPC}$, and the one-vs-rest code would select only one of these composite labels. In contrast, the mapping applies independently to the nodes in each layer (e.g., select all NN, not just those which co-occur with inside), and therefore allows richer modelling.

Each constituent model was presented with two-layer binary-labelled lattices for training. As these structures were cyclic, exact inference could not be used directly. The training time was slower using approximate loopy BP than using forward-backward on the projected lattices, which formed chains. The converged log-likelihood and the parameter values were near identical for the two approaches. Therefore, reasons of speed, the projection technique was used, which produced chains with 4 labels at each node.

The times in Table 6.4 show that ECOC training is still considerably faster than multiclass training with loopy BP. Attempts to train the multiclass model using projection onto a chain were thwarted due to the massive training time and high memory requirements. This model would take roughly 3 weeks to train and require about 50Gb of RAM. This contrasts with a memory requirement of 2.7Gb for loopy BP, and around 1Gb for the ECOC constituent models. The decoding times are much slower than the baseline, which again is due to the very large numbers of constituent models and unoptimised routines. Loopy BP was used for multiclass and product decoding, and diverged on average only 0.7% of the time.

The performance of these models is well below that of the baseline multiclass CRF. This suggests that ECOC may be less well suited to lattice based tasks, or that other means of ECOC training and decoding are required. This is a surprising result, as we would expect that the technique would work equally well for any graphical structure. Better codes may be required, although preliminary experimentation with (short) Hamming and BCH codes resulted in similar performance results. A possible reason for this result is given in Section 6.5.1, which also suggests various means by which the problem might be addressed.

<table>
<thead>
<tr>
<th>Model</th>
<th>Train (s)</th>
<th>Decoding Time (s)</th>
<th>Acc. Devel.</th>
<th>Acc. Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiclass</td>
<td>1,177,747</td>
<td>-</td>
<td>94.09</td>
<td>93.36</td>
</tr>
<tr>
<td>Random code 100</td>
<td>296,597</td>
<td>standalone 2,054</td>
<td>90.96</td>
<td>90.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 2,874</td>
<td>91.38</td>
<td>90.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 18,600</td>
<td>91.73</td>
<td>90.90</td>
</tr>
<tr>
<td>One-vs-rest code</td>
<td>232,723</td>
<td>standalone 3,032</td>
<td>91.07</td>
<td>90.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>marginals 4,488</td>
<td>91.51</td>
<td>90.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product 25,714</td>
<td>91.71</td>
<td>90.74</td>
</tr>
</tbody>
</table>

Table 6.4. Joint POS and NPC tagging results for ECOC training. *Trained over 4 CPUs.
6.3.1 Summary

The results on the three experiments showed some general trends and also raised a number of questions. Firstly, ECOC training was often much faster than regular training, depending on the choice of error-correcting code. Even the most simple code (one-vs-rest) produced very strong performance with a fast training cycle. The one-vs-rest code often out-performed codes with much higher row and column separations. This surprising result may be due to the one-vs-rest code defining much simpler binary tasks than defined by more complex codes.

Product decoding performed better than marginals, while standalone decoding was uniformly poor. This confirms earlier predictions, and suggests that there is information in the uncertainty of constituent models and that jointly resolving the labels during decoding can make better use of sequencing models predictions. ECOC training was relatively insensitive to constituent model regularisation, in comparison to standard training. This was attributed to rigorous smoothing harming the overall ensemble performance despite improving constituent model predictions.

The overall performance of ECOC training and decoding improved over standard training for NER but under-performed for the POS and POS/NPC tasks. The performance degradation for POS tagging was quite small, however for POS/NPC the loss was considerable.

6.4 Analysis

We now present further analysis of the questions raised by the previous experiments. This section presents learning curves for ECOC training, showing how the method performs on small training samples. This is followed by an analysis of different codes, which provides a possible explanation as why the one-vs-rest code performs well despite it not being able to correct any errors. We also evaluate a number of traditional codes, hand-tailored codes and random codes.

6.4.1 Small Training Samples

Both the POS and POS/NPC tasks had very small training corpora (5,000 and approximately 7,000 sentences – less than half the size of the NER task); the observed under-performance may be limited to only small training samples. Therefore we now assess the quality of ECOC on different sizes of training samples. Figure 6.5 shows learning curves for NER and POS, which compare the timing and performance of the one-vs-rest code to the multiclass. The training sample is randomly sub-sampled without replacement from the full training set, starting with 1,000 sentences and then repeatedly doubling in size. All models are trained using a zero-mean Gaussian with a fixed variance of 10. Note the
Figure 6.5. Learning curves for NER and POS tagging, comparing the one-vs-rest ECOC CRF with the multiclass. Note here that the accuracy grows smoothly with more data, regardless of the decoding method, and that the accuracy of product decoding is almost identical to that of the multiclass. The training time of the one-vs-rest code grows at a much slower rate than multiclass training time.

The learning curve in Figure 6.5(a) shows that all four curves exhibit increasing accuracy with each doubling of the training sample. The same behaviour is seen for POS tagging in Figure 6.5(c), except that here the magnitude of the increases are getting smaller each time. The POS tagger is closing in on an asymptotic ceiling on accuracy. The NER tagger is still well short of the ceiling, so we are not yet seeing this effect. The product decoding method produces near-identical accuracy to the multiclass, while the marginals is close behind and standalone a little further away. The marginals and standalone curves have steeper gradients than the product and multiclass curves, suggesting that with a large enough training sample the four curves will converge. Therefore, the choice of decoding method becomes immaterial with a representative enough training sample. This finding also suggests that the ECOC CRF is a consistent estimator of the multiclass CRF.

Figures 6.5(b) and 6.5(d) show the training time for the one-vs-rest ECOC CRF and the multiclass, on a log-log scale. Both the ECOC and multiclass curves are roughly linear in each task and therefore the training time grows polynomially with the training...
Chapter 6. Scaling CRFs using Error-Correcting Output Coding

<table>
<thead>
<tr>
<th>label</th>
<th>code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-LOC</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>B-MISC</td>
<td>0 1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>B-ORG</td>
<td>0 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>I-LOC</td>
<td>0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>I-MISC</td>
<td>0 0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>I-ORG</td>
<td>0 0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>I-PER</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>O</td>
<td>0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

(a) One-vs-rest code

<table>
<thead>
<tr>
<th>label</th>
<th>code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-LOC</td>
<td>1 1 1 0 0 0 0 1</td>
</tr>
<tr>
<td>B-MISC</td>
<td>1 1 0 0 1 1 0 0</td>
</tr>
<tr>
<td>B-ORG</td>
<td>1 0 1 1 1 0 0 0</td>
</tr>
<tr>
<td>I-LOC</td>
<td>1 0 0 1 0 1 1 1</td>
</tr>
<tr>
<td>I-MISC</td>
<td>0 1 1 1 0 1 0 1</td>
</tr>
<tr>
<td>I-ORG</td>
<td>0 1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>I-PER</td>
<td>0 0 1 0 1 1 1 1</td>
</tr>
<tr>
<td>O</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

(b) $H^*(7,3)$ code

Figure 6.6. Two error-correcting codes, with 0 and 1 bit error-correcting capacity, respectively.

sample. Fitting polynomial curves to the data results in powers of 1.48 and 1.35 for the one-vs-rest model on the two tasks, and 1.56 and 1.55 for the multiclass. This shows that the multiclass has poorer scaling characteristics. For both tasks, the multiclass curve has a higher y-intercept and therefore proportionally higher training time than the one-vs-rest ECOC CRF.

Together the graphs in Figure 6.5 show that the ECOC CRF has a similar learning curve to the multiclass CRF. This is coupled with a similar empirical complexity of the training time.

6.4.2 Properties of Good Codes

We now turn to the issue of what properties are present in a good code, and how we might go about finding codes for which we expect strong accuracy. A good code should have a good error-correcting capacity while also inducing constituent models with low error correlation. We now attempt to analyse these characteristics further, and apply these findings in the construction of high quality codes.

6.4.2.1 Why Does the One-vs-rest Code Perform So Well?

An constituent model can model a few decision boundaries relatively easily, but modelling a more complex set of boundaries is usually more difficult and therefore leads to more prediction errors by the constituent model. For this reason, we may wish to rely more heavily on ‘simple’ models, such as those in the one-vs-rest code, which each model a single label. The one-vs-rest code used for the NER task is shown in Figure 6.6(a), where the rows show the code word for each label, and the columns define the constituent constituent models. As described earlier, these models learn to distinguish between the set of labels marked with a 1 and set marked with a 0. Therefore each constituent model learns the boundary between a single label and all other labels.
If we use only very simple constituent models, the ensemble would make very few individual bit errors: a very desirable property, presuming that these errors can be corrected. One would expect that the use of more complex constituent models would lead to more individual errors in the ensemble. However, this alone cannot be used to justify the sole use of very simple constituent models. When individual errors do occur some codes are better than others at recovering the correct labels – those codes with higher error-correcting capacities can still resolve the correct label in the presence of bit errors. The one-vs-rest code is very bad at correcting errors as its code words only differ from each other in two bit positions. Therefore after flipping a single bit on a valid code word (simulating a single prediction error), the resulting bit-string cannot be unambiguously resolved back to the original code word. Resolving multiple bit-errors is even more difficult for this code.

The low error-correcting capacity of the one-vs-rest code explains why it performs so poorly with standalone decoding, as seen in Table 6.1. When using the other decoding methods the uncertainty in the constituent model predictions allows for more individual errors to be more easily corrected or smoothed away. This suggests that there are many instances where there is ambiguity in the posterior distribution of the individual constituent models and the Viterbi path is prematurely removing this ambiguity.

So how does the code’s error-correcting capacity relate to the complexity of its constituent models and therefore their prediction errors? To assess this we compare two similarly-sized codes, the one-vs-rest code and the $H^*(7,3)$ code, shown in Figure 6.6. The latter code is a modified version of the $H(6,3)$ Hamming code, supplemented with an extra parity bit. The first three columns of the Hamming code encode the information bits – the binary representation of the label – while the remaining 4 columns provide redundancy in the form of parity checks. Each of these parity checks encode the parity (whether the sum is even or odd) over different combinations of the information bits. Because of this carefully engineered redundancy, the codewords of the $H^*(7,3)$ code differ from all others codewords by four bit flips. Therefore, it is capable of correcting a single bit error.\(^3\) This contrasts with the one-vs-rest code’s separation of two bits and correction capacity of zero bits.

The columns of the $H^*(7,3)$ code are evenly populated with ones and zeros. Each constituent model in the Hamming code must distinguish between four labels and their complement set. For example, the model for the final column must distinguish between [B-LOC, I-LOC, I-ORG, I-PER] and [B-MISC, B-ORG, I-MISC, O], i.e., 16 different decision boundaries between pairs of labels. This number of decision boundaries is maximised when the sets are of equal size because the two sets partition the full label space. This re-

\(^3\)In traditional ECCs the $H^*(7,3)$ code offers no improvements over the $H(6,3)$ code because they both have the same error correcting capacity. However, in ECOC the extra parity bit does afford a greater correction capacity when decoding using the marginals or product methods. Moreover, the extra bit adds another orthogonal information source (column) which has low error-correlation with the other columns.
Chapter 6. Scaling CRFs using Error-Correcting Output Coding

Figure 6.7. Comparing column density and the number of decision boundaries in an error-correcting code. Each node depicts a label and the heavy edges depict decision boundaries. A column with a density $d = 1$ (left) models four decision boundaries (shown as solid edges), while a column with $d = 2$ (right) models six boundaries.

The relationship is illustrated in Figure 6.7 which shows the decision boundaries when selecting one or two labels from a set of five.

In general, the modelling task increases in difficulty with greater numbers of decision boundaries. This generalisation will not always hold: we would expect that certain combinations of boundaries could be effectively modelled together, such as the boundaries between begin and inside labels for NER and the boundaries between nominal tags and verbal tags in POS tagging. We define the \textit{density} of a column as follows:

$$d(C_n) = \min \left( \sum_i \delta(C_{i,n},0), \sum_i \delta(C_{i,n},1) \right)$$  \hspace{1cm} (6.7)

In general, the complexity is roughly related to the density of the column: a column with equal numbers of zeros and ones is likely to be complex, with many decision boundaries, while columns with mostly zeros or mostly ones is probably much simpler, with many fewer decision boundaries. This conjecture relating the column density of the code to the modelling complexity is supported by observed predictive accuracy of constituent models, as well as in the increased training time of each constituent model, as illustrated in Figure 6.8. Figure 6.8 also compares the column density to the error rate for each constituent model, which confirms the intuition that constituent models modelling more decision boundaries tend to make more prediction errors.

Figure 6.9 shows the average values of the Pearson correlation coefficient between the errors by pairs of constituent models from the exhaustive code.\footnote{The number of pairs of constituent models was extremely large ($127^2$) and therefore 1000 pairs were selected at random for Figure 6.9.} The error correlation is overall positive and often large, and therefore when one constituent model makes an error during decoding, it is likely that many other constituent models will also make an error at the same site. This suggests that large error-correcting codes will not achieve their theoretical error-correcting capacity due to a bursty error distribution. This also explains why accuracy does not continue to increase for arbitrarily long codes. Furthermore, where there are only small differences between the columns for pairs of constituent models (i.e.,
the inter-column distance is low), there is much higher error correlation than between constituent models with very different columns. This confirms the link between column separation and error correlation.

Figure 6.9 (right) shows that the number of shared decision boundaries can also explain the increase in error correlation. This is a more direct measure of similarity between the models’ tasks than column separation. For example, two columns of the one-vs-rest code will differ in two bits (a fairly poor column separation) but will share only one decision boundary. However, these pairs of constituent models typically have a very low error correlation as they model very different concepts (observed to be around 0.1 for the NER task). This is better explained in terms of the number of shared decision boundaries than by the column separation. The number of unique decision boundaries over the pair of models (i.e., those modelled by one model by not the other) also explains the error correlation well.

In summary, when selecting a code we seek a compromise between two factors: the simplicity of each column and the diversity between pairs of columns. Low density columns lead to fewer individual prediction errors and a diverse set of columns will reduce error-correlation, thereby increasing the capacity of the code to correct errors. In addition, simpler columns define constituent models which are faster to train. The one-vs-rest code relies entirely on simple columns, while communication codes typically use quite complex (dense) columns. The best codes must mix the two styles of column sensibly.
Figure 6.9. Error error-correlation between models constituent models, plotted against column separation (left) and the number of shared decision boundaries (right). These graphs show error correlation between pair of constituent models tends to lessen as their inter-column separation increases, or when they share fewer decision boundaries.

6.4.2.2 Hand Tailored Codes

Until now we have restricted ourselves to automatically generated codes, such as the one-vs-all code and random codes. This raises the question of whether it is possible to hand engineer codes which rival these automatic codes. Hand tailored codes allow us to specify how the labelling task should be decomposed into simple and logical steps. In this way domain knowledge can be represented into the coding matrix. However, it is often difficult to specify redundant steps in this decomposition and thereby produce a code with a good error-correcting capacity.

Ideally we wish to construct codes which are small and also result in strong accuracy. For this reason we start with the one-vs-rest code for NER and supplement it with a few additional columns suitable for the task. Four codes were engineered in this manner and are described below.

**OVR + EO** This code adds three new columns which each detect a single entity type, $X$, combining the labels $B-X$ and $I-X$. In this way the begin and inside labels for the same entity type can be modelled together. Although there are four entity types, only three had begin tags in the training set, and therefore only three new columns were required.

**OVR + BI** This code adds two new columns which detect (1) whether the label is a begin label, $B-*$, and (2) whether the label is an inside label, $I-*$. This allows modelling of transitions between two adjacent entities of any type where the begin tag is used.

**OVR + BI + EO** This code adds to the one-vs-rest the five new columns taken from the EO and BI codes above.
### Table 6.5

Timing and accuracy using alternative codes for NER. The results shown are development $F_1$ scores. $H_r^{\text{min}}$ shows the code’s minimum Hamming distance between rows, $x = r$, or columns, $x = c$ (taking into account each column’s complement). Size shows the number of columns in the code and therefore the number of binary constituent models. The difference between the standalone and the marginals or product results are significant at $p < 0.05$ except for Exhaustive. The marginals and product results are insignificantly different in all cases except OVR +BI and OVR +EO.

<table>
<thead>
<tr>
<th>Coding</th>
<th>Size</th>
<th>$H_r^{\text{min}}$</th>
<th>$H_c^{\text{min}}$</th>
<th>Train (s)</th>
<th>Stand.</th>
<th>Marg.</th>
<th>Prod.</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-vs-rest</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>1,388</td>
<td>82.85</td>
<td>89.19</td>
<td>89.68</td>
</tr>
<tr>
<td>Pairs</td>
<td>28</td>
<td>12</td>
<td>2</td>
<td>6,546</td>
<td>89.11</td>
<td>89.65</td>
<td>90.04</td>
</tr>
<tr>
<td>Exhaustive</td>
<td>127</td>
<td>64</td>
<td>1</td>
<td>32,827</td>
<td>89.22</td>
<td>89.53</td>
<td>89.94</td>
</tr>
<tr>
<td>BI +EO +O</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1,497</td>
<td>85.08</td>
<td>88.79</td>
<td>89.50</td>
</tr>
<tr>
<td>OVR +BI</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>1,733</td>
<td>84.91</td>
<td>88.92</td>
<td>89.52</td>
</tr>
<tr>
<td>OVR +EO</td>
<td>11</td>
<td>2</td>
<td>1</td>
<td>2,080</td>
<td>82.07</td>
<td>88.52</td>
<td>89.35</td>
</tr>
<tr>
<td>OVR +BI +EO</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>2,427</td>
<td>85.26</td>
<td>87.70</td>
<td>89.20</td>
</tr>
<tr>
<td>$H(6,3)$</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>1,678</td>
<td>87.96</td>
<td>88.86</td>
<td>89.65</td>
</tr>
<tr>
<td>$H^+(7,3)$</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>1,925</td>
<td>87.73</td>
<td>89.35</td>
<td>89.93</td>
</tr>
<tr>
<td>BCH(11,3)</td>
<td>11 (7)</td>
<td>5</td>
<td>0</td>
<td>2,789</td>
<td>88.16</td>
<td>89.23</td>
<td>89.82</td>
</tr>
</tbody>
</table>

**BI +EO +O** This code combines the BI and EO columns (described above) with a single column to detect the label 0. This columns are sufficient to fully specify the label, and therefore the one-vs-rest columns are not included. This code will not be robust to errors, as it uses such a minimal set of columns with no redundancy.

The results for these codes (and others) are shown in Table 6.5. This table shows the codes’ row and column separations, as well as their training times and development accuracy. A number of automatically generated codes are also presented for comparison, including the one-vs-rest and exhaustive codes. The results are also shown graphically in Figure 6.10, which plots the training time against accuracy for each decoding method. All the engineered codes have a fairly poor row and column separation, which is even worse than the one-vs-rest due to increased similarity between their columns. The accuracy of these codes looks reasonable in comparison to the one-vs-rest code for standalone decoding, however they are uniformly worse for marginals and product decoding. When compared to the other codes in the table, the accuracy of the engineered codes is quite poor. This highlights the difficulty of engineering an error-correcting code.

#### 6.4.2.3 Algebraic Codes

Table 6.5 and Figure 6.10 show results for a range of alternative codes, including the pairs code, some Hamming codes and a BCH code. The pairs code is a natural extension to the one-vs-rest code, containing a column for every unique pair of labels. This results in $\frac{L(L-1)}{2}$ columns: quadratic in the label set, and therefore it cannot provide any real scaling
Figure 6.10. Accuracy vs. time trade-off for various error-correcting codes in NER, using the three decoding methods. Note that one-vs-rest code (OvR) performs very poorly for standalone decoding, but relatively well for the other two decoding methods. Also note that the hand-tailored codes uniformly under-perform the conventional error-correcting codes.
improvements over multiclass training. However, it does achieve the best accuracy on the development set, exceeding even the exhaustive code. This is most probably due to its reduced error-correlation while still having a very good error-correcting capacity.

The $H(6,3)$ code is a Hamming code with six columns, of which three are information bits and therefore it encodes 8 different codewords. The three redundant bits of the $H(6,3)$ code check for parity over the three combinations of two source bits and each source bit participates in two parity checks. This code has a minimum row separation of 3 bits and can correct a single bit error. This error-correcting capacity is evident from its accuracy, where with standalone decoding the $H(6,3)$ code soundly exceeds the one-vs-rest code and all the hand-tailored codes. In general Table 6.5 shows a strong relationship between the row separation (and therefore error-correcting capacity) and the standalone decoding accuracy. Note that the codes with a row separation of 2 bits or fewer score 85 or lower, those with a row separation of 3-5 score around 88 and those with a greater separation perform the best, with scores around 89. This relationship is also observed, albeit less strongly, for the other decoding methods, where the one-vs-rest code is an outlier.

The $H^+(7,3)$ code is the $H(6,3)$ with an extra parity check over all three source bits, thereby increasing the row separation of the code. This does not afford any greater error-correcting capacity, and therefore does not significantly affect the standalone accuracy. However, it does result in much better accuracy using marginals and product decoding. This is because the new column enlarged the code without worsening the minimum column distance and hence the error correlation. Therefore more largely independent information is made available to the error-correcting ensemble during decoding, which results in better accuracy. Incidentally, the $H^+(7,3)$ is equivalent to the Hadamard matrix after removing a trivial column of zeros. Hadamard matrices are symmetrical square matrices which have optimal row and column separation, and are used in Reed-Muller codes (MacWilliams and Sloane, 1977). These separation properties are ideal for ECOC, although the symmetry constraint means that only a single Hadamard matrix is appropriate for a given number of labels. These matrices have only been proven to exist for sizes which are powers of two, and not for arbitrary sizes.

The accuracy of a BCH code was also included in the comparison. These codes can provide near-optimal error-correcting capability (MacWilliams and Sloane, 1977), however they provide no guarantee of good column separation. The BCH(11,3) code had four duplicate columns and therefore a column separation of zero. The training time shown is that for only the seven unique constituent models. While the code provided additional row separation, this came at the expense of perfectly correlated columns and therefore its slightly worse accuracy when compared to the $H^+(7,3)$ comes as no surprise.

In summary, the algebraic codes performed very well until they were sized so large as to contain duplicate columns. The small Hamming based codes had strong row and column separation, which lead to strong accuracy. This can be seen in Figure 6.10, where
for short training times the $H^*(7,4)$ achieves the strongest accuracy almost uniformly over decoding methods. This exceeds the one-vs-rest and most other algebraic codes, as well as the hand tailored codes. For longer training time, the pairs code produces the best accuracy, although this is not a solution for larger label sets because the code length is quadratic in the size of the label set. In contrast, the Hamming code is instead sized proportional to the logarithm of the label set size and is therefore ideal for scaling.

### 6.4.2.4 Row Ordering

One-vs-rest and exhaustive codes have a symmetrical structure, such that the allocation of labels to codewords is irrelevant. However this is often not the case for algebraic codes. A different allocation (and therefore a reordering of the rows in the code) does not change the row or column separation, but it does affect the label partitions modelled by the constituent models. Ideally ECOC should be robust to ordering of the rows, which will be the case if the set of partitions specified by the code are of equivalent overall difficulty and therefore lead to similar error rates (and error correlation). Figure 6.11 shows the effect of randomly reordering the rows of the $H^*(7,3)$ code on the development $F_1$ score over five trials. Here we see that the variation in accuracy is quite small, particularly so for marginals and product decoding. This closely parallels Dietterich and Bakiri’s (1995) findings for ECOC classification, where that the row ordering was observed to have only had a very mild and insignificant effect on accuracy.

### 6.4.2.5 Random Codes

So far we have analysed exhaustive codes and structured shorter codes, such as the one-vs-all code and algebraic codes. An alternative class of codes exists which is particularly relevant for very large label sets where the exhaustive code (or even the one-vs-all code)
is impractical. With these larger label sets, the columns must be selected with care to maximise the row and column separation. As demonstrated above, some algebraic codes feature good column separation, but this is usually incidental and cannot be relied upon. An alternative is to use a random code, where the code matrix elements are randomly initialised to either zero or one. For these codes the probability of poor separation diminishes quickly as the number of columns increases (Berger, 1999).

Figure 6.12 shows the development accuracy using standalone decoding for NER as a function of code length. The ‘random’ method selects a random sample of columns from the exhaustive code, thereby eliminating the possibility of duplicate columns. The ‘random with replacement’ method samples the columns with replacement and can therefore obtain duplicate columns with perfectly correlated errors. Both random curves exhibit a similar accuracy profile: the accuracy increases quickly while the code is small, but the gains from increasing the code size lessen as the code gets larger. In Figure 6.12 the accuracy has reached a plateau at 50 bits. Ideally with no error correlation the accuracy would continue to rise. The effect of error correlation increases with the code length, which therefore leads to the accuracy plateau. It is interesting to observe that accuracy does not peak, and then degrade with longer codes. This suggests that any increase in error correlation is countered by the increased error-correcting capacity for sufficiently large codes. This accuracy profile is not unique to NER or the standalone decoding method, as demonstrated in Figure 6.13 which shows the development accuracy profiles for POS tagging with each decoding method averaged over three trials. For short codes there is considerable variability in accuracy, while for longer ones this variability is lessened. This is a result of shorter codes not evenly modelling all decision boundaries. This effect is simply a function of their size: in the limit (the exhaustive code) all boundaries will be equally represented.

### 6.4.2.6 Minimum Loss Bound Codes

Error-correcting codes can also reflect a skew in the label distribution. Ideally the code should minimise the confusability of commonly occurring labels more so than rare labels. In NER, for example, the B-LOC and B-ORG labels are very rare and therefore there it is less useful to model the boundary between these two labels compared to boundaries involving more common labels. The label O is very common and therefore the code should distinguish between O and other labels, common or rare. This ensures that the code focuses on limiting common errors, that is those errors which involve commonly occurring labels. Assuming that errors made by the constituent models are independent, the probability of a single constituent model error, $q$, as a function of the code $C$ can be bounded by:

$$q(C) \leq 1 - \sum_x p(x) \sum_{i=0}^{\frac{h(C)}{i}} \binom{|C|}{i} f^i(1-f)^{|C|-i}$$

(6.8)
Figure 6.12. NER accuracy for standalone decoding with random codes of increasing size. The accuracy of a minimum loss code and a greedy oracle are also shown.

Figure 6.13. Random codes for POS tagging, showing the average development accuracy versus code length.
where \( p(s) \) is the marginal probability of the label \( s \), \( h(C,s) \) is the minimum Hamming distance between the codeword for \( s \) and all other codewords in the code \( C \), \( f \) is the probability of an individual constituent model error and \( |C| \) is the code's length. There is a much greater correlation between the bound in (6.8) and the accuracy than observed between the minimum row or column separation and accuracy, when measured over a set of random codes for NER.

Figure 6.12 shows the accuracy achieved by selecting the code with the minimum loss bound from 10,000 random codes of each length, where the constituent model error rate, \( f \), was estimated on the development set. This curve exceeds the accuracy of the random codes, more quickly reaching the accuracy plateau. The assumptions underlying the minimum loss bound in (6.8) are increasingly unjustified as the code is enlarged, leading to the cap on accuracy. For comparison, an oracle was also created which attempted to find the best possible code at each given size. It did this by starting with an empty code and greedily adding a column at a time, choosing the column which maximises the development \( F_1 \) score. This approximates the best possible code, avoiding the exponential search required to find the best code. The minimum loss bound is not as effective as the oracle; potentially other styles of codes could close this gap, and thereby increase the accuracy and reduce the cost of ECOC training.

### 6.4.3 Non-uniform Voting Weights

Now that we have analysed a number of different classes of codes, we consider an alternative decoding setup which can be used to discover which columns of a code are the most useful. In the previous experiments, all constituent models were given an equal vote in decoding. That is, during decoding the prediction of each binary model contributed equally: each bit error was penalised uniformly in the Hamming or \( L_1 \) distance calculations for the standalone and marginals methods, and each model was combined equally in a geometric mean for the product method. Instead of equal voting we use an unequal voting ensemble, where the each constituent model can be amplified or dampened according to a set of weights. This codifies an intuition that some constituent models will be more useful than others and therefore in order to best model the data these models should have a greater relative vote. The following description of product model CRFs follows Smith et al. (2005), who form ensembles over multiclass CRFs to improve accuracy. Here this same technique is applied to the product formulation of the ECOC CRF.

The constituent model weights are learnt from data by maximising the data likelihood. We use the product formulation from (6.6) supplemented with a weight for each
constituent model, $\alpha_n$, which is then renormalised:

$$p_{\text{LOP}}(\mathbf{s}|\mathbf{o}) = \frac{1}{Z_{\text{LOP}}(\mathbf{o})} \prod_{n=1}^{N} p_n(b_n(C,s)|\mathbf{o})^{\alpha_n} \quad \text{s.t.} \quad \sum_{n=1}^{N} \alpha_n = 1 \land \alpha_n \geq 0, \forall n \quad (6.9)$$

$$= \frac{1}{Z_{\text{LOP}'}(\mathbf{o})} \exp \sum_{n=1}^{N} \alpha_n \sum_{k} \lambda_{n,k} F_{n,k}(b_n(C,s),\mathbf{o}) \quad (6.10)$$

where $Z_{\text{LOP}}$ and $Z_{\text{LOP}'}$ are the partition functions, which sum the numerator of (6.9) and (6.10) respectively for all labellings $\mathbf{s}$. This formulation allows the distributions of some models to be sharpened with high weights, while others are flattened towards the uniform distribution with low weights. When the weights are uniform this distribution is proportional to (6.6) which is equivalent for decoding.

Equation (6.10) defines a logarithmic opinion pool (LOP) over CRFs (Heskes, 1998; Smith et al., 2005), which itself is a CRF with the standard log-linear form. We can use this LOP-CRF to model the log-likelihood of iid fully observed data:

$$L_{\text{LOP}} = \sum_{\mathbf{o},\mathbf{s}} \tilde{p}(\mathbf{o},\mathbf{s}) \sum_{n=1}^{N} \alpha_n \sum_{k} \lambda_{n,k} F_{n,k}(b_n(C,s),\mathbf{o}) - \log Z_{\text{LOP}'}(\mathbf{s}) \quad (6.11)$$

which can be maximised in the standard way using L-BFGS, as described in Section 3.6. In contrast to the standard CRF training which requires derivatives with respect to the model parameters, here we need the derivatives with respect to each weight, $\alpha_n$. These derivatives are complicated by the normalisation and positivity constraints in (6.9).

These constraints are represented using a soft-max transformation which expresses the constrained weights, $\alpha$, as a function of unconstrained weights, $\gamma$:

$$\alpha_n = \frac{\exp \gamma_n}{\sum_{n'=1}^{N} \exp \gamma_{n'}} \quad (6.12)$$

The exponential ensures positivity, while the denominator ensures the values sum to one. The soft-max transform in (6.12) is used to re-express (6.11) in terms of the unconstrained weights, $\gamma$. The resulting log-likelihood may be maximised using unconstrained L-BFGS to find the optimal $\gamma$ values, from which the constrained weights can be directly evaluated. The derivatives are as follows:

$$\frac{\partial L_{\text{LOP}}}{\partial \gamma_{n'}} = \sum_{\mathbf{o},\mathbf{s}} \tilde{p}(\mathbf{o},\mathbf{s}) \sum_{n=1}^{N} \Theta_n(C,s,\mathbf{o}) \{ \alpha_n \delta(n,n') - \alpha_n \alpha_{n'} \}$$

$$- \sum_{\mathbf{o}} \tilde{p}(\mathbf{o}) \sum_{\mathbf{s}'} p_{\text{LOP}}(\mathbf{s}'|\mathbf{o}) \sum_{n=1}^{N} \Theta_n(C,s',\mathbf{o}) \{ \alpha_n \delta(n,n') - \alpha_n \alpha_{n'} \} \quad (6.13)$$

where $\Theta_n(C,s,\mathbf{o}) = \sum_{k} \lambda_{n,k} F_{n,k}(b_n(C,s),\mathbf{o})$ are the per-constituent model labelling scores, and $\gamma$s in the right hand side have been rewritten as $\alpha$s for better readability. This retains the familiar form of the CRF gradient of empirical count less expected count, where in this case the quantity being counted is:

$$\sum_{n=1}^{N} \Theta_n(C,s,\mathbf{o}) \{ \alpha_n \delta(n,n') - \alpha_n \alpha_{n'} \}$$
This is primarily the constituent model scores, but complicated by the weight constraints.

This method was used to train a LOP-CRF on the task of simplified POS tagging using an exhaustive error-correcting code. This task uses a simplified tag set with 5 labels denoting adjective, adverb, noun, verb and other parts-of-speech (articles, punctuation, conjunctions, etc.). This mapping was performed so as to simplify the task, allowing the very computationally demanding LOP model to train in a reasonable time with an exhaustive code. Recall that the exhaustive code requires $2^L - 1$ unique columns which is impractical for large label sets. For this task the exhaustive code has 15 columns. The constituent models were first trained on 4,000 sentences from the Penn Treebank III (Marcus et al., 1994) after mapping into the simplified label set. The LOP weights were then learnt on the same training set using L-BFGS to maximise the log-likelihood with no prior over the weights. This configuration resulted in an accuracy of 96.50% compared to the uniform’s 96.48%. The real outcome of this experiment are the converged weights, which we can use to inform our code design.

The final weights are shown in Table 6.6. There are two main findings from this experiment. Firstly, note that the weights are zero for all columns which select a single label. These columns are those used by the one-vs-rest code. The normalisation condition from (6.9) requires the constituent models to compete for weight, such that only the constituent models which best aid in modelling the data are assigned weight. This competition has forced all of the weight onto the complex constituent models which model more decision boundaries. This reaffirms the earlier assertion that codes with ‘high density’ columns (i.e., columns with similar numbers of zeros as ones) are often more effective than very simple codes, despite the increased complexity in the constituent models’ modelling tasks and correspondingly higher constituent model error rates.

Secondly, the non-zero weights in Table 6.6 are also quite skewed. This skew roughly reflects the label distribution in the training set. The labels are distributed as follows: $O > N > V \gg J > R$. Those models covering decision boundaries between the less frequent labels have been allocated the lowest weights (e.g., bit 8 = [J, R] and bit 13 = [J, V]), while those covering boundaries involving common labels are assigned high weights. The
highest weight is for bit $5 = \{N, O\}$, which selects for the two most common labels.

Both of these findings can influence code design. The skew in the weight distribution suggests that the code should reflect the label distribution, such that the resulting product model can more accurately model common labels than rare labels. This lends credibility to the minimum loss bound codes described in Section 6.4.2.5. The zero weight assignments confirm that despite the strong accuracy of the one-vs-rest code, other codes can provide better modelling power. These codes have higher density columns, covering more decision boundaries.

## 6.5 Possible Extensions

Now that we have established the properties of good codes and investigated a number of means of finding such codes, we return to the results of the main experiments from Section 6.3. We observed that error-correcting CRFs resulted in substantial savings in the training time over the baseline multiclass CRF, but did not always achieve competitive accuracy. The method outperformed the baseline for NER, but underperformed for POS tagging and for joint POS tagging and NPC. We now seek to identify the reasons behind these results and suggest ways in which the method can be improved.

### 6.5.1 Independence Assumptions

The accuracy degradation can be explained with reference to the constituent model training configuration in (6.1). The $n^{th}$ constituent model is trained to maximise the likelihood of sequences of binary labels, where each binary label is produced by encoding the original label and taking the $n^{th}$ output bit. In this way, one constituent model is trained for each column of the coding matrix. This requires some selective independence assumptions over bits in the labelling’s encoding, which may be unwarranted for certain tasks. Traditional ECOC for classification assumes that the bits of the output representation are independent random variables. This motivates the decomposition of multiclass training in a set of independent training steps. In ECOC sequence modelling we assume that bits in the output representation for a single label are independent, but that there is a (possible) dependence between the same bit of each label’s encoding over a label sequence. If we were to assume that all bits are independent, then the ensemble would be unable to model time dependence and instead would make independent classifications for each label at each time. The modelling of time dependence is imperative for sequence labelling tasks. The dependencies also motivate the use of a CRF to model the joint distribution over binary labellings. However, these independence assumptions may not be appropriate to all tasks.

The practical upshot from these assumptions is that each constituent model has heavily constrained features over edge labellings. These features can only test for membership...
of each of the labels in a set. For edges between pairs of nodes this results in four separate label conditions: into (0,1), out of (1,0), between (1,1) and remaining outside the label set (0,0). This allows the ECOC CRF to easily model chunking style tasks, which feature sequences of identical labels where consecutive words are all members of the same chunk. For these tasks a single constituent model can model transitions into a given chunk label, between this label and out of the label. This also holds across groups of labels which follow a bursty distribution and can therefore be modelled accurately by a single constituent model. For tasks with richer state transitions (e.g., POS tagging and the joint tagging task) these features are relatively inexpressive and do not easily summarise the full transition information inherent in the data. For such tasks it is more difficult to justify the selective independence assumptions made by ECOC training. This means that in order to accurately model such tasks, either many constituent models or more training data are required, or else accuracy will suffer.

This problem could be addressed by making different independence assumptions between the output representation for different parts of each labelling. Each constituent model could instead model binary labellings over different bits in the output representation, rather than being constrained to only a single bit. This would relax the 1:1 correspondence between code columns and constituent models and therefore the number of constituent models and the modelling task of each constituent model would need to be carefully considered. Other learning algorithms could be used in place of CRFs to model the binary distributions. With very different independence assumptions, full joint inference over binary labellings may no longer be required and therefore other methods, such as maximum entropy models, could be used instead. As stated earlier, in order to model time dependence the constituent models must model the dependence between some bits in the labelling’s encoding. This represents a worthy avenue for further research.

### 6.5.2 Convolutional Error-Correcting Output Coding

Using other types of error-correcting codes present another possible avenue for improving the accuracy of ECOC for sequence models. The method presented earlier in Section 6.2 used block codes to independently encode each label in the sequence. We could extend this configuration to encode two or more labels in each block, thereby allowing each constituent model to model richer transitions. However, the use of multi-label blocks would impose an asymmetry on the model: label \(n\)-grams which cross a block boundary would be modelled very differently to those that fit wholly within a block. This would result in different degrees of error-correction over intra- versus inter-block transitions.

We suggest that convolutional codes (MacKay, 2003) could be used to encode the label sequence. These codes can encode label \(n\)-grams without the asymmetry of block codes. Convolution codes encode a stream of data using a linear filter with a memory (a shift register) and linear output functions (MacKay, 2003). The label sequence would form the
input data stream, after being rendered into binary using, for example, a simple block code. The filter (encoder) stores a finite history of past source bits in its shift register and uses these bits along with the next source bit as input to some linear functions (e.g., parity checks). The function outputs form the transmitted message and the number of functions determines the transmission rate. This filter is equivalent to sliding a fixed size window over the source bit sequence and calculating at each position an encoding of these bits. The bit stream formed by these encodings is the transmitted message. Upon receipt of this message, which may be corrupted, the receiver must determine the original sequence of source bits. This is done probabilistically, by deducing the most probable way in which the encoder could have created a message for transmission which was then randomly corrupted to produce the received message.

Each constituent model describes the distribution over a subset of the bits in the encoded label sequence in the training data. In decoding each constituent model predicts the values of their bits. These predicted values (or confidence estimates) form the observed ‘received message’. Inference could be used to decode the most probable label sequence resulting in the constituent models’ predictions, using loopy belief propagation. While the original block code approach limited the constituent models to model functions of single labels, while this convolutional approach allows the constituent models to model functions over \( n \)-grams of labels. Therefore the convolutional approach should allow constituent models to more adequately model label transitions, although this would incur a higher training and decoding time.

6.6 Conclusion

This chapter showed how error-correcting output coding (ECOC) can be used with CRFs to efficiently perform sequence labelling. ECOC is a technique used in classification to render a multiclass problem into a series of binary problems (Dietterich and Bakiri, 1995). The binary models vote to decode an unlabelled instance, from which the label can be determined. This chapter showed how ECOC can be applied to sequence labelling, decomposing a multiclass labelling problem into a series of binary labelling problems, where each is modelled by a CRF. Independently training the binary CRFs is overall much more efficient than training a single CRF on the full multiclass problem. We presented three different decoding methods which use the binary CRFs to predict a full multiclass labelling. This method yields similar accuracy to standard training, but often trains in only a fraction of the time.

ECOC training allows CRFs to be applied to larger problems and those with larger label sets than were previously possible. On standard tasks ECOC requires much less time and memory to train. The training time is a function of the length of the code and the complexity of columns (which determines the set of binary tasks). The one-vs-rest
code is a particularly simple code which trains a single model for each label. Despite its simplicity the code produces quite respectable accuracy. Other codes which are sized proportional to the label set or to the logarithm of the label set can also result in strong accuracy. These small codes afford ECOC training improved scalability over standard CRF training.

The accuracy of the approach relies heavily on the choice of error-correcting code. This code should induce constituent models which are diverse, having low error correlation. This can be achieved by ensuring a high column separation (or by other means, such as feature selection). In addition, the code should also have a high error-correcting capacity, through a high row separation. This provides the ensemble with robustness to constituent model prediction errors. Codes can be engineered to provide such properties using algebraic codes from coding theory or else by using random codes. This chapter presented a number of techniques for creating powerful short codes.

Despite the reduced cost, even binary inference is intractable for general graphs, which may have large cycles or large cliques. Therefore ECOC training may not be appropriate by itself in these situations. The following chapter presents an alternative method for scaling CRFs which takes a more direct approach to reducing the cost of training and decoding. This alternative method reduces the cost of inference and therefore maximum likelihood training and decoding. This can allow tractable inference in cyclic graphs with large cliques.
This chapter introduces an alternative method for scaling Conditional Random Fields (CRFs) to large tasks. As described in Chapter 3 and demonstrated in Chapter 5, CRFs scale poorly to tasks with a large number of states or with richly connected graphs. This is a consequence of the time complexity of inference, which is quadratic in the label set for linear chains – the most commonly used graphical structure, corresponding to a first order Markov assumption. For graphs with larger cliques (e.g., a higher order chain), the complexity is raised to an even higher power. Performing supervised training of a CRF requires repeated sum-product inference over each graph. Therefore for tasks with a large numbers of label and/or large cliques, the cost of training a CRF is often prohibitively expensive. Decoding is also expensive as it requires max-product inference, however this is only applied once per graph.

Chapter 6 presented a method by which CRFs could be trained in a fraction of the time required for standard training. This approach used error-correcting output coding to decompose a full multiclass sequence labelling task into a series of independent binary sequence labelling tasks. This had the net effect of reducing the cost of training, as each binary model was cheap to train, while achieving similar accuracy to standard training.

This chapter presents a novel approach to reducing the training time and improving its asymptotic complexity. This method is similar in nature to standard CRF training in that it does not use an ensemble decomposition like the ECOC approach. The method uses a novel mechanism that reduces the complexity of inference and therefore training and decoding complexity. This is achieved by exploiting regularities in the features to optimise inference. These regularities can be described (or engineered) by feature constraints. Typical features have the form $f_k(c, s_c, o)$, where $c$ is a clique, $s_c$ is a clique labelling and $o$ are the observations. We constrain the labellings considered in each feature function such that the functions can only detect membership in a set of labellings rather than detecting a specific labelling. The constrained features can be considered as
Chapter 7. Efficient Inference using Tied Potentials

Figure 7.1. Venn diagram showing the set of labellings.

disjunctive features over different labellings. They do not add any power to the model, but rather constrain the model to have the same parameter value for all the features which are now tied. The sets of labellings are chosen a priori and partition the full labelling space. These constraints lead to many labellings being detected en masse, with the features unable to discriminate between each component labelling. As such the clique potentials share the same tied structure, with shared potentials for all labellings in each labelling set. For this reason we refer to the method using the term tied potentials. When one of these labelling sets is considerably larger than the other sets, this asymmetry can be leveraged to improve the efficiency of inference. The tied potentials method is orthogonal to many other alternative training methods, such as error-correcting output coding, piecewise training and perceptron training (Sutton and McCallum, 2005; Freund and Schapire, 1999): these methods could feasibly be combined for further reductions in training time.

In typical NLP tasks only a small fraction of the full set of labellings are attested in the training data, and even amongst these, the majority are quite uncommon. This is illustrated in Figure 7.1, which shows a Venn diagram over the set of labellings. The unseen labellings will often contain a subset of illegal labellings, i.e., ungrammatical labellings or labellings which violate the tagging guidelines. We might choose to partition the labelling space to collapse distinctions within these sets of uncommon, illegal or otherwise unseen sets. For example, in a first-order chain CRF we might constrain the features over label bigrams to collapse the distinction between uncommon bigrams. This may sacrifice some modelling power, although when used in combination with features over label unigrams, the sacrifice is often insignificant.

Feature tying has two beneficial effects: the parameter space of the model (and thus the propensity to over-fit) is reduced, and the cost of inference is reduced – potentially also reducing the asymptotic complexity. In practice, CRFs are typically used with extremely large feature sets and therefore must be regularised heavily to limit over-fitting of the training sample. The tied-potentials method limits the expressiveness of the feature set,
and therefore can also reduce the extent over-fitting. The sum-product algorithm (Pearl, 1988) can exploit the tied potentials for a significant reduction in runtime, therefore allowing faster training. The complexity of optimised sum-product algorithm is reduced from $O(LC)$, where $L$ is the number of labels and $C$ the maximal clique size, to $O(S)$, where $S$ is the number of selected labellings. In cases where $S < O(LC)$ this results in a reduction in complexity. Similarly, the cost of decoding (the max-product algorithm; Pearl (1988)) can also be considerably reduced. On standard natural language tasks this is shown to reduce CRF training time three-fold without significant loss in accuracy. More importantly, this method allows for inference in very richly connected graphs for which many current methods – exact or approximate – are intractable.

The method exploits regularities in the feature set during exact sum- and max-product inference. This contrasts with other approaches to speed inference in CRFs, which typically use either approximate inference or optimisation of an alternative loss function. In many tasks there are relatively few useful clique labellings and these can be easily enumerated. As such, tying the potentials of all remaining labellings should not reduce the modelling ability of the CRF, and indeed may make better use of sparse data.

This chapter is organised as follows: Section 7.1 describes the feature constraints and how these can be exploited in the sum-product and max-product algorithms. Experimental results are given in Section 7.2, showing that the approximations can improve training and decoding time on the benchmark tasks, without loss of accuracy. Further experiments show that richly connected graphical structures can be used to tractably perform more complex tasks, as illustrated for semantic role labelling. Section 7.3 presents related work and conclusions are presented in Section 7.4.

### 7.1 Tied Potentials

Chapter 3 described CRFs and their inference operations following Lafferty et al. (2001). The two fundamental tasks for CRFs are training and decoding, which both require belief propagation (BP) inference (Pearl, 1988). Sum-product BP is used to find marginal distributions over cliques, which are required to calculate the log-likelihood objective function in training (see Sections 2.3.1 and 3.6). Max-product BP is used directly for decoding, where we require the maximising configuration over hidden variables given the observations (see Sections 2.3.2 and 3.5.2).

We propose the use of tied potentials to simplify both the sum- and max- product algorithms, and thus training and decoding. Firstly we make the standard assumption that all the features have the form $f_k(c, s_c, o) = g_k(c, o)h_k(s_c)$, where $g$ detects features of the observation and $h$ detects features of the labelling. See Sections 3.2 and 3.3.2 for a full description of feature functions. The example from (3.16) is reproduced here for
convenience:

\[ f_{38}(s_c, o_c, c) = \llbracket o_c = (\text{put, up}) \rrbracket \cdot \llbracket s_c = (\text{VBN, RP}) \rrbracket \] (7.1)

The function \( g_{38} = \llbracket o_c = (\text{put, up}) \rrbracket \) could be reused with other \( h \) functions to provide features combining the same contextual predicate (observation test) with different labellings.

**Feature Constraints** We now partition the full labelling space, \( S \), into a number of disjoint sets:

\[ S = \bigcup_{i=1}^{N} S_i \]

The labelling space is the Cartesian product of the label space at each node, e.g. for a factor over three nodes, we must consider the labelling space \( S = \Omega^3 \) where \( \Omega \) is the label set. We constrain the functions \( h_k \) to detect only whether the labelling is in one of these sets, i.e., the \( h_k \) functions are limited to testing set membership rather than testing for exact label configurations. Therefore the feature in (7.1) could only be included if one of the labelling sets contained only the single labelling, i.e. \( S_i = \{(\text{VBN, RP})\} \). If there was no such set, then the contextual predicate must be combined with a different labelling test, \( h_i \), to qualify for inclusion.

The label sets are chosen *a priori*, and typically conflate (tie) sets of label configurations which are to be modelled together rather than individually. Without loss of generality, we assume the last of these sets, \( S_N \), is the largest, which we call the *remaining labellings*: these are often the labellings considered illegal or those which are of little use. In contrast, the labellings in the other sets are referred to as the *selected labellings*.

Many situations arise in NLP where the remaining set is much larger than the selected set. This occurs when using a small labelled training set in which only a small proportion of the possible labellings are observed in the sample. In this case the model cannot learn to distinguish between the unseen labellings, but only to prefer those attested labellings in preference. For this reason the model need not model the unseen labellings precisely, and this can be exploited by tying together their features. The distribution of labellings is often highly skewed, as illustrated in Figure 7.2 which shows the counts for label bigrams in NER. Of the 64 possible bigrams, only 33 were attested, and of these attested bigrams only 11 occurred commonly when using a count threshold of 1000. This distribution suggests that conflating distinctions among these many infrequent bigrams may not harm modelling accuracy, while drastically reducing the size of the selected set.

In a chunking task, we might be interested only in whether the labels belong to a single chunk type or to none at all. The remaining labellings constitute spans over chunk boundaries. This partition of the label space has \( L \) selected labellings and \( L_C - L \) remaining labellings – an overwhelming majority when using large cliques, \( C \). Often there are implicit constraints over valid labellings which can be encode by feature constraints. For instance when chunk tagging with an IOB2 label strategy (Tjong Kim Sang and Veenstra,
7.1. Tied Potentials

We should not transition to a begin tag except from an inside tag of the same chunk type. Where the labellings are strictly illegal, they can be assigned a zero valued potential, and therefore will not be considered during inference. However, in many cases the boundary between legal and illegal labellings is not clear-cut, and therefore excluding these labellings from inference may be detrimental to accuracy.

**Factor Potentials** Recall that CRFs describe a distribution using a globally normalised product of clique potentials, as shown in (3.13). The potential functions, \( \psi_f(s_f, o) \), can be restated using the feature constraints:

\[
\psi_f(s_f, o) = \exp \sum_k \lambda_k g_k(c, o) h_k(s_f) = \exp \sum_k \lambda_k g_k(c, o) h_k(s_f) = \exp \sum_i \sum_{k \in K_i} \lambda_k g_k(c, o) \mathbb{1}[s_f \in S_i] \tag{7.2}
\]

where \( K_i \) indexes the features which detect a labelling in \( S_i \). The labelling sets \( S_i \) partition the full labelling space, and therefore the labelling test \( \mathbb{1}[s_f \in S_i] \) in (7.3) evaluates to one for only one labelling set, \( S_i \). This leads to the following structure of the potential function:

\[
\psi_f(s_f, o) = \sum_i \omega_i \mathbb{1}[s_f \in S_i] \quad \text{where} \quad \omega_i = \exp \sum_{k \in K_i} \lambda_k g_k(f, o) \tag{7.4}
\]

These potential functions allow us to avoid explicit enumeration of the remaining labellings during inference. This provides a reduction in complexity, insofar as the set of
selected labellings is smaller than the set of all possible labellings.

The potentials obey the ‘mostly-constant’ property of Siddiqi and Moore (2005), who present a similar approach for approximate inference in Hidden Markov Models. They encode the state transition matrix by preserving the top $K$ transitions from a given state, while sharing the remaining probability mass evenly between all other transitions from that state. We now show how the tied potentials can be exploited in the sum- and max-product algorithms.

### 7.1.1 Sum-product

Sum-product BP requires messages to be sent in both directions over all edges in the graph, and subsequently calculates the beliefs (marginal distributions) from these messages (see Section 2.3.1). These beliefs are required during CRF training to calculate the log-likelihood objective and its derivatives, as described in Section 3.6. We present belief propagation for factor graphs (Kschischang et al., 2001), and therefore consider messages sent between nodes and factors (see Section 2.3). In the proposed model the messages sent from nodes to factors (2.17) remain unmodified and are reproduced here for convenience:

$$m_{n_i \rightarrow f}(s_i) \propto \prod_{f' \in N(n_i) \setminus f} m_{f' \rightarrow n_i}(s_i) \quad (7.5)$$

The messages from factors to nodes (2.18) have a time complexity of $O(LC)$, where $C$ is the factor size. These messages can be computed more efficiently by exploiting the tied potentials:

$$m_{f \rightarrow n_i}(s_i) \propto \sum_{s_{j'}:s_{j'} = s_i} \psi_f(s'_{j'}, o)M(s_f) \quad (7.6)$$

$$= \sum_{s_{j'}:s_{j'} = s_i} \sum_{j=1}^{N} \omega_j \mathbb{1}[s'_j \in S_j]M(s'_j) = \sum_{j=1}^{N} \sum_{s'_j \in S_j: s'_j = s_i} \omega_j M(s'_j) \quad (7.7)$$

$$= \omega_N \sum_{s_{j'}:s_{j'} = s_i} M(s'_j) + \sum_{j=1}^{N-1} \sum_{s'_j \in S_j: s'_j = s_i} (\omega_j - \omega_N)M(s'_j) \quad (7.8)$$

$$= \omega_N + \sum_{j=1}^{N-1} (\omega_j - \omega_N) \sum_{s'_j \in S_j: s'_j = s_i} M(s'_j) \quad (7.9)$$

where $M(s_f) = \prod_{f' \in N(f) \setminus f} m_{n_i \rightarrow f}(s_i)$ is the product of incoming messages. Equation (7.7) incorporates the tied potential structure from (7.4) into the message formulation, and then simplifies the summations to reflect that each factor labelling is a member of one selected set. The second step in (7.8) adds the default potential, $\omega_N$, for every possible label configuration, which is offset by the potential difference, $\omega_j - \omega_N$, for every selected

---

1We can apply feature tying independently to different factors in a graph. For instance, in a chain with factors over label unigrams and bigrams, we need only tie the features over the bigrams. This optimises the most expensive message without dramatically affecting the modelling power.
configuration. The final result is shown in (7.9), which makes an additional assumption that each message vector is normalised to one, leading to the left most summation in (7.8) simplifying to one. This result can be verified by re-stating the sum over incoming messages using a standard dynamic programming technique:

\[
\sum_{s'_{i} \in s_i} M(s'_j) = \sum_{s'_{i} \in s_i, n_i \in N(f) \cap n_i} m_{n_i \rightarrow f}(s'_j) = \prod_{n_i \in N(f) \cap n_i} \sum_{s'_{i} \in s_i} m_{n_i \rightarrow f}(s'_j) \quad (7.10)
\]

This decomposition is possible because each factor in the product references independent labels \(s'_j\) in the labelling \(s'_f\). The righter-most sum over each element in a message vector is one by the normalisation assumption and therefore the product is also one. This normalisation assumption is unrestrictive, as belief propagation usually requires some form of normalisation to avoid numerical overflow.\(^2\)

Equation (7.9) avoids the summation over the full space of labellings: instead it sums over only the selected labellings. This expression has a time complexity linear in the total number of selected labellings \(S\), and we obtain an overall time complexity of sum-product inference of \(O(TS)\) regardless of the factor size, where \(T\) is the number of edges in the graph.\(^3\)

The marginals can then be calculated using (2.19) and (2.20), which are reproduced below:

\[
p(s_i) \propto \prod_{f \in N(n_i)} m_{f \rightarrow n_i}(s_i) \quad (7.11)
\]

\[
p(s_f) \propto \psi_f(s_f) \prod_{n_i \in N(f)} m_{n_i \rightarrow f}(s_i) \quad (7.12)
\]

The node marginals (7.11) are easy to calculate, although the factor marginals (7.11) can prove more difficult. To calculate the normalisation constant we must sum the numerator of (7.12) for each possible factor labelling, \(s_f\), of which there are exponentially many. Fortunately, the normalisation constant can be calculated efficiently using the same trick from (7.9) to exploit the tied potential values:

\[
\sum_{s_f} \psi_f(s_f, o) M(s_f) = \omega_N + \sum_{j=1}^{N-1} (\omega_j - \omega_N) \sum_{s_f \in S_f} M(s_f) \quad (7.13)
\]

where the notation \(M(s_f) = \prod_{n_i \in N(f)} m_{n_i \rightarrow f}(s_j)\) is recycled from before, but without excluding \(n_i\) from the summation. This allows the efficient calculation of the normalisation constant for the distribution without explicitly enumerating every labelling. The same technique can be used to efficiently find the sum of beliefs over the set of remaining

\(^2\)For numerical reasons this algorithm was implemented in log-space, which required special handling of the subtraction in (7.9), because negative numbers cannot be represented. The full result will always be positive, and therefore the subtraction is performed as the final step to avoid negative intermediate results.

\(^3\)Larger factors will require more computation, as the number of messages in the product \(M(s_f)\) will increase. However, this effect is linear in the factor size, not exponential.
labellings, $\sum_{s_f \in S_f} b(s_f)$, without enumerating these labellings. The complexity of belief calculation is $O(S)$.

This technique allows the marginals to be calculated efficiently for use in evaluating the CRF’s log-likelihood and gradient during L-BFGS training. The cost of sum-product inference, as used to find these expected marginals, forms the bulk of the training cost. Therefore these optimisations allow for faster training and importantly also yield a lower time complexity when $S < O(L^C)$.

### 7.1.2 Max-product

The decoding time with a CRF is often slower than desirable, especially in tasks with many labels or large factors. The max-product algorithm is used for decoding, as described in Sections 2.3.2 and 3.5.2. As with the sum-product algorithm, the most expensive messages are those going from factor to node (2.26). These messages can also exploit the tied potentials:

$$m_{f \rightarrow n_i}(s_i) = \max_{s_f \in S_f, s_i = s_f} \psi_f(s_f, o) M(s_f)$$

(7.14)

$$= \max_j \omega_j \max_{s_f \in S_f, s_j = s_f} M(s_f)$$

(7.15)

using the definition of $M(s_f)$ from (7.9). This reduces to a maximum over any selected or remaining labelling. The same trick used in the sum-product method in Section 7.1.1 cannot be used for this maximisation.

In order to calculate this maximum, we use a dynamic program over the set of selected labellings. This aims to find the labelling with the maximum score in an inductive manner, building labellings left to right until each labelling thus produced is either a member of a selected set, or else can never lead to a labelling in a selected set. In this way the search process avoids the full space of remaining labellings. This is achieved by constructing a trie over the selected labellings, as illustrated in Figure 7.3. A trie is a type of search tree for strings which represents keys as paths from the root to a leaf node, where each edge specifies parts of the string. The example trie shows three selected sets of labellings and therefore four distinct potentials – one for each selected set and one for the remaining set. The paths leading to remaining labellings are omitted from the trie, denoted by dotted arcs. By traversing this trie, we can maximise over both the selected labellings and the remaining labellings. Clearly this trie is considerably smaller than the full trie for every possible labelling of $\{A, B, C\}^3$, which would have 11 internal nodes and 27 leaves. Note also that the trie could be constructed for any node order. For example, if the trie in Figure 7.3 corresponded to a factor over the nodes $n_3, n_4, n_5$, an alternative equivalent trie could instead be used for the ordering $n_4, n_3, n_5$. This alternative trie would have two root transitions for labels B and C. These alternative tries may have different numbers of internal nodes (due to earlier or later exclusions of remaining labellings) and therefore different search costs.
Maximisation uses a preorder traversal, storing at each node a score: the product of incoming messages. As we descend via an edge, we multiply the current score by the message $m_{n_i \to f}(s_i)$ where $n_i$ is the next node in the factor graph and $s_i$ is the edge label. In the example in Figure 7.3 the partial path AC would have score $m_{n_3 \to f}(A) \times m_{n_4 \to f}(C)$, where the first two nodes in the factor are $n_3$ and $n_4$. Once we reach a leaf we have found a selected labelling; its score is given by the product of its potential $\omega_i$ and the leaf score. At each stage in our traversal we may traverse off the trie by following a dotted arc, after which no continuation of the path can yield a selected labelling. Therefore we can maximise the score for subsequent states without constraints. This maximum is simply $\omega_N$ if we ensure that message vectors are normalised such that the maximum message value is 1. This yields a maximum score for the remaining labellings (with a given prefix) of $\omega_N$ times the score of the external node reached after traversing the dotted edge. The scoring function is shown as pseudo-code in Figure 7.4 using a recursive formulation.

The trie reduces the search space from the full set of labellings to the number of nodes in the trie with an additional fringe. We can reduce this further by traversing the trie in a best-first manner, using $A^*$ search (Russell and Norvig, 2003) to guide us towards branches of the trie in which we expect high scores. As a simple heuristic we use the product of the node score and the maximum potential value, which gives an upper bound on the score for the subtree.\(^4\) This takes a very small number of steps, as it allows a better traversal order: promising subtrees are preferentially searched before less promising ones. This avoids exhaustive traversal of the entire trie by cleverly directing the search process without resorting to approximation. Furthermore, when there are only a handful of selected labellings, this decoding technique involves markedly fewer calculations than would be required to enumerate the full set of labellings.

The complexity of this decoding process stems from the number of internal and external nodes in the trie. In a general trie $L$ nodes are reachable from the root, including both the internal and external nodes. From the $B_1 \leq L$ internal nodes another $B_1L$ nodes

\(^4\)The maximum messages are normalised to one, so the score for descendant nodes can never exceed the current score.
define score(p):
    if p is a leaf then
        j ← potential_index(p)
        return ω_j
    else if p is in fringe then
        return ω_N
    else
        v ← 0
        for all (c,s) ∈ outgoing_edges(p) do
            n ← node(c)
            v ← max(v, m_{n→f}(s) × score(c))
        end for
        return v
    end if

Figure 7.4. Pseudo-code for the decoding algorithm which recursively calculates the score for each node in the trie. Here ‘fringe’ refers to the external nodes reachable by traversing a dotted arc, the incoming and outgoing edge functions return the nodes reachable by a transition and the transition’s label, and the node function returns the node in the factor graph corresponding to the depth in the trie.

are reachable, and at the next level $B_1B_2L$ nodes may be reached, etc. This leads to a complexity of $O(L^C)$, i.e. the same complexity as standard max-product decoding. This would occur where the selected labels covered every possible prefix, and each remaining labelling only differed from a selected labellings only on their final label. In practise the selected labellings will often not generate these worse case trie, and the heuristic will direct the search towards the best labelling without exhaustively visiting all nodes in the tree.

This decoding technique does however have a drawback. In standard max-product decoding after message passing is complete the maximising configuration is found in an iterative manner. This requires finding the maximising label at a single node, and then tracing back to its neighbouring nodes their maximising labels given the label at the first node. The process continues to ground labels in the maximising configuration given those labels which have already been grounded. This allows decoding to find a maximising configuration when there are many such configurations. This process is described in Section 3.5.2 in relation to the Viterbi algorithm.

Using the tied decoding method, the back-pointers are not always deterministic, and therefore cannot be traversed to yield a maximising configuration. That is, this method finds the set of transitions which maximises the score at a node with a given label. When this set contains many labellings, the back-pointers will be ambiguous. The solution to this problem is to use independent maximisation of each label at each node in the graph. This ensures that each label is part of a maximising configuration, although the full labelling
7.1. Tied Potentials

may not be a maximising configuration itself. For practical purposes the distribution of a trained CRF is usually peaked about a single label sequence, and therefore this subtle distinction is immaterial. Empirical predictions for POS tagging were identical using per-node maximisation and standard exact decoding. This was verified with a variety of trained models.

7.1.3 Using the Trie for Sum-Product Inference

The same trie used in decoding could also be used for sum-product inference. In the sum-product algorithm the product of the message values, $M(s_f)$, must be evaluated for each selected labelling. When pairs of the selected labellings have the same prefix, the intermediate products needed to evaluate $M$ for one labelling can be reused in the calculation of $M$ for the other. In this way the number of multiplications required could be reduced. There would only be substantial savings when the number of internal nodes in the tree (and thus multiplications required) is considerably fewer than the number of selected labellings — i.e. where many paths have common suffixes. This situation occurs for large factors with many nodes coupled with a large label set and many selected labellings.

The trie structure suggests a further generalisation of the sum-product optimisation: allowing the choice of the remaining labelling set to differ between subtrees. This is illustrated in Figure 7.5, which shows two tries for the space of binary label trigrams, where 0 and 1 transitions are shown as downward and upward slanting arrows, respectively. The left-hand tree shows the current sum-product method, where the remaining set is chosen as the most common, in this case number 3. The dotted arc shows this choice, and indicates that its potential is added for all paths starting at the root. This potential, $\omega_3$, is then subtracted from the potential for each path in remaining sets (terminating at the unshaded leaf nodes). The shaded nodes in the tree need not be considered, as they only form part of labellings in the remaining set. These remaining labellings have no contribution to the summation — i.e. the potential difference at the leaves are all zero, $\omega_3 - \omega_3 = 0$.

Figure 7.5 (b) shows a novel configuration with two different remaining sets, chosen to minimise the size of the remaining sets in each subtree off the root. Here the potential for labelling one, $\omega_1$, is added to all paths which start with label 1, and $\omega_3$ is added to all paths which start with label 0. These two sums over all paths in a subtree both result in a value of one, using the same reasoning used in (7.10). Therefore this extension also presents an efficiency gain by again avoiding explicit consideration of the remaining labellings. The potentials for the leaf nodes in the two subtrees must subtract the relevant tied potential value. For example the top \( \oplus \) leaf will use $\omega_2 - \omega_1$ as its potential value, while the lower \( \oplus \) leaf uses $\omega_2 - \omega_3$. In this way only three full selected paths need to be evaluated, instead of the four in the left tree. For larger examples the difference between the two methods will
Figure 7.5. Example tries for sum-product inference over binary label trigrams. The left tree shows a single remaining set while the right tree has two remaining sets. These lead to four and three selected labellings respectively, shown as unshaded leaf nodes.

often be greater, justifying the added complication. In the following experiments, only a single remaining set was used, although using many may have provided an efficiency gain.

7.2 Experiments

We now present experimental results on the three standard tasks set out in Chapter 4: part-of-speech (POS) tagging, named entity recognition (NER) and joint POS tagging and noun phrase chunking (NPC). For these tasks features over rare labellings (those beneath a frequency threshold) were tied and training and decoding were performed using the optimised inference algorithms described in Section 7.1. We present the majority of results using a very simple method of feature tying which tied all the labellings below a given threshold. Ideally feature tying should be motivated by some domain or task knowledge. For this reason other more expressive methods of feature tying were also applied to NER and POS tagging, in an attempt to integrate domain knowledge into the model. These are presented in Section 7.2.4.

Of the three tasks, we start with POS tagging, which has a large label set and therefore a rich space of label transitions (bigrams). In this task standard CRF training is only just possible with modern hardware, and we show how this can be made considerably faster. We also go on to show how this method can be used to perform semantic role labelling (SRL) over novel richly connected cyclic graphs with many large factors. For such graphs even loopy belief propagation is impossible due to the large factor size. We show that feature tying can be used to make training and decoding possible for such graphs.
7.2. Experiments

7.2.1 Part-of-speech Tagging

The first experiment entailed tagging words with part-of-speech labels, which was modelled with a chain CRF using the data set and feature templates described in Section 4.1. In this task there are 43,189 training sentences, a total of 1,023,863 tokens and 45 labels. The first timing experiments used only the first 5,000 training sentences, while the accuracy experiments used the full set.

Figure 7.6 shows a histogram which displays the number of label transitions (bigrams) at or above each frequency threshold on the small POS training set. Of the possible $45^2 = 2025$ transitions, only 1027 were attested in the sample. Interestingly, the clear majority of transitions are quite uncommon, with about half of the bigrams occurring fewer than ten times. This augurs well for the tied feature optimisation, as excluding rare transitions will drastically reduce the number of selected transitions modelled and correspondingly reduce the cost of inference.

Our working hypothesis is that although these infrequent transitions constitute the majority of transition types, excluding these transitions will not harm model accuracy. ‘Excluding’ here means tying their features in the set of remaining labels, such that each of these transitions no longer has its own specific feature. These tied transition features are used alongside regular dense single node (unigram) features; the unigram features allow the model to disambiguate between the various tied bigrams.

7.2.1.1 Timing

Five sets of selected pair-wise labellings were derived from the reduced training set: those transitions occurring at or above a frequency threshold. This threshold was set to select between 5% – 65% of the attested transitions and tie the remaining uncommon transition labellings. Each selected transition labelling was modelled with a separate
Table 7.1. POS tagging results for the small training sample. Accuracy is shown in percent, and the MAP models use a zero-mean Gaussian prior with a variance of 10. †These results are significantly different to the baseline accuracy using the matched pairs test at $p < 0.05$.

The converged log-likelihood values for the models trained with each transition threshold were very similar, as are the accuracy results. Significant reductions in accuracy were only observed for the very coarse transition thresholds. Clearly little modelling power was sacrificed by tying the rare transitions. This is illustrated in the MLE results where only the most coarse threshold significantly degrades performance.

Figure 7.8 shows the MAP accuracy results plotted against the number of selected transitions. The development accuracy increases with the number of selected transitions while the test results peak early, at around 300 transitions. Although these differences are not significant, the test results suggest that the feature tying might limit over-fitting of the training set.

---

5. The baseline results here differ from the results in Chapter 5 due to the accidental incorporation of a new feature template in these experiments. This template combined the word shape of the current token (Collins, 2002b) with the POS tag, and resulted in a 0.1% improvement in accuracy.

6. Statistical significance was measured using the matched pairs test at $p < 0.05$, as described in Section 4.4.3. This test is used throughout the chapter.
Figure 7.7. POS tagging training and decoding times plotted against the number of selected transitions. Training and decoding times are shown on the left and right axes, respectively. Observe the linear increase in time for both curves. These models were trained on the small training set.

Figure 7.8. POS tagging accuracy versus the number of selected transitions. Note that pruning the space of transitions is initially quite detrimental, but quickly levels off. The best test result is quite heavily pruned. These models were trained on the small training sample.
### 7.2.1.2 Accuracy

The tied models were then applied to the full training set, using thresholds of 1000, 500 and 100. Lowering this threshold considerably increased the number of selected labellings (and thus the run-time requirements) for no tangible gain; therefore we stopped at 100. As described in Section 5.1 the standard CRF was also trained on the full task for comparison. The accuracy of these models on the development and test sets are shown in Table 7.2. A Gaussian prior was used with a zero mean and unit variance except in the case of the baseline, which had a variance of 10. Once again, we see that the accuracy is quite similar across all thresholds (the results are not significantly different), indicating that a simple frequency based selection policy is adequate. As expected, the training time (shown for a single evaluation of the log-likelihood objective) increases with the number of selected labellings. The total training times were 6, 10, 14 and 29 days for the four models respectively. Our best accuracy is extremely competitive with not only the full dense CRF, but with other more complex models (e.g., Toutanova et al. (2003), although we cannot compare the results directly as they have used a different training and testing split).

### 7.2.2 Named Entity Recognition

The NER task is described in Section 4.3; results for the task are shown in Table 7.3. This task only had 8 labels, and therefore 64 possible transitions. Of these, only 33 were attested in the training set, which left little room to manoeuvre for feature tying. The same frequency threshold approach that was applied to POS tagging was also used here, for three different thresholds. The development and test $F_1$ scores are roughly equivalent to the baseline – there were no significant differences in accuracy. As earlier, the ML results often improve a little over the baseline, as a result of implicit smoothing in the tied models. Training and decoding time increases linearly with the number of selected transitions, although there is little gain to be had for a task with so few labels. Note that the threshold 1 tied model should be equivalent to the baseline model, although it differs very slightly in accuracy.\footnote{As described in Section 4.4.1 we use the term \textit{accuracy} in its general sense of test set performance, which in this case is measured by the $F_1$ score.} This difference is probably due to subtle numerical differences
7.2. Experiments

<table>
<thead>
<tr>
<th>Threshold</th>
<th>No. sel.</th>
<th>Train (s)</th>
<th>Decode (s)</th>
<th>Dev. MLE</th>
<th>Dev. MAP</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>12</td>
<td>3905</td>
<td>13.1</td>
<td>87.49</td>
<td>89.84</td>
<td>82.98</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>4808</td>
<td>14.9</td>
<td>87.55</td>
<td>89.86</td>
<td>82.94</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>5897</td>
<td>15.4</td>
<td>87.33</td>
<td>89.95</td>
<td>82.91</td>
</tr>
<tr>
<td>Baseline (dense)</td>
<td>5516</td>
<td>35</td>
<td>87.33</td>
<td>89.93</td>
<td>82.91</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3. NER results, showing the $F_1$ scores in percent. MAP models all used a zero-mean Gaussian prior with a variance of 100. No results are significantly different to the baseline, using the matched pairs test at $p < 0.05$.

in the software implementation. Note also that the training time for the tied model with threshold of one exceeds the time required by the baseline model, despite the baseline considering approximately twice as many labellings in each transition. This reflects the added cost of enumerating the selected labellings in the tied model, which cannot be as easily optimised as the full dense enumeration.

7.2.3 Joint Part-of-Speech Tagging and Noun-Phrase Chunking

The joint POS tagging and NPC task differs from the previous tasks in that it features two distinct layers of annotation and a lattice graph. This task is described in Section 4.2. Features are defined over adjacent pairs of POS tags, adjacent pairs of NPC tags and over the POS and NPC tag pairs. These three types of ‘transition’ have different event spaces, and therefore were tied separately. The NPC pairs had very few labellings ($3^2 = 9$), and therefore they were not tied. The other two transition types were tied using the same frequency threshold, for five different frequency values, as shown in Table 7.4. For reference, the dense model has $45^2 = 2025$ POS transitions and $45 \times 3 = 135$ POS/NPC transitions.

The training and decoding times both increase with the increasing numbers of selected transitions. The training times for lower thresholds exceed that of the baseline model, notably so for the tied model over all transitions, which takes 50% longer to train.

The joint accuracy increases as the threshold is lowered. Overall, even the results using a coarse transition threshold of 100 provide insignificantly different test accuracy, while training in half the time.

7.2.4 Alternative Methods for Feature Tying

Now that we have seen the results on the three standard tasks using a simple method of feature tying, we investigate how one might tie features in a more informed manner. The earlier experiments used a simple frequency threshold to determine which transitions should be modelled individually while all rare transitions below the threshold were modelled together. This pruning was demonstrated to yield linear training time in the number of selected transitions, while often only slightly harming accuracy. In some
instances, this method very slightly increased accuracy over the baseline, which was attributed to a smoothing effect.

Here we attempt to tie features in an informed manner in order to achieve accuracy improvements over the baseline model. These improvements could arise from two effects. 1) Tied features will reduce the parameter space and therefore the model’s propensity to over-fit the training sample. This reduction in dimensionality may provide a more effective means of regularisation than a simple prior alone. 2) The tied features will have denser counts than their non-tied variants. Therefore the effect of idiosyncrasies from sparse counts derived from small training samples will be reduced. In other words, this may reduce the variance of the learning algorithm (the component of the prediction error which is due to noise in the training sample and random behaviour of the learning algorithm (Kohavi and Wolpert, 1996)).

**Named Entity Recognition** The named-entity task used IOB style labels (Tjong Kim Sang and Veenstra, 1999), which is described in Section 4.2. The IOB label set is highly redundant, in that begin and inside tags have a similar meaning. This suggest a manner of feature tying which equates transitions from begin or inside tags of a given type. For example, the transitions from B-LOC to I-LOC and I-LOC to I-LOC both correspond to continuing a location chunk, and therefore tying these features might improve the model. This method of feature tying resulted in 43 different transitions and 28 unique potentials. Training with a zero-mean Gaussian prior with variance of 100 yielded a development $F_1$ score of 89.92, which was slightly lower the best results in Table 7.3, albeit insignificantly so. One would expect that this method of feature tying should increase accuracy. The inconclusive result is probably due to the small size of the label set relative to the training set size: this leads to quite reliable counts over transitions and therefore feature tying has little room for improvement. Moreover, the begin (B) tags were used only when necessary to delineate separate chunks – i.e. when two chunks of the same type are adjacent in the text. Therefore the begin tags are quite rare and a tying method which relies on counts

<table>
<thead>
<tr>
<th>Threshold</th>
<th>POS sel.</th>
<th>POS/NPC sel.</th>
<th>Train (s)</th>
<th>Decode (s)</th>
<th>Dev.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>74</td>
<td>35</td>
<td>601424</td>
<td>271</td>
<td>93.55†</td>
<td>92.81†</td>
</tr>
<tr>
<td>100</td>
<td>254</td>
<td>60</td>
<td>552980</td>
<td>360</td>
<td>93.84†</td>
<td>93.21</td>
</tr>
<tr>
<td>50</td>
<td>333</td>
<td>67</td>
<td>737164</td>
<td>394</td>
<td>93.97†</td>
<td>93.26</td>
</tr>
<tr>
<td>10</td>
<td>593</td>
<td>10</td>
<td>1301084</td>
<td>472</td>
<td>94.04</td>
<td>93.34</td>
</tr>
<tr>
<td>1</td>
<td>1057</td>
<td>118</td>
<td>1842928</td>
<td>411</td>
<td>94.04</td>
<td>93.36</td>
</tr>
<tr>
<td>Baseline (dense)</td>
<td></td>
<td></td>
<td>1177747</td>
<td>74</td>
<td>94.09</td>
<td>93.36</td>
</tr>
</tbody>
</table>

Table 7.4. Joint POS/NPC results, showing the joint accuracy in percent. All MAP models use a zero-mean Gaussian prior with a variance of 10. All training and decoding was performed over cluster of four CPUs (times adjusted appropriately). †These results are significantly different to the baseline joint accuracy using the matched pairs test at $p < 0.05$.  

154 Chapter 7. Efficient Inference using Tied Potentials
from these tags in the training set will therefore have little effect. For chunk data labelled with begin tags at the start of each chunk, or with end tags signifying the last word in a chunk, this domain-specific feature tying method should be more effective.

### Part-of-Speech Tagging

An alternative method of feature tying was also used for POS tagging. For this task, the author did not have sufficient domain expertise to propose a linguistically plausible manner of tying. Instead an automatic clustering approach was used to find groups of similar POS transitions and the features of these clustered transitions were tied. Thereby the counts for many individual features were aggregated into tied features with less sparse and potentially more reliable counts.

The clustering was performed using the CLUTO clustering toolkit,\(^8\) by means of repeated bisection. This technique starts with one large cluster which is split in two whereafter its children are recursively split until there are the desired number of clusters. Each split is chosen to maximise the \(I_2\) criterion function:

\[
\sum_i \sqrt{\sum_{v,u \in S_i} \text{sim}(v, u)}
\]

where \(i\) indexes the clusters, \(S_i\), the inner summation ranges over every pair of instances within a cluster, and \(\text{sim}(v, u)\) measures the similarity of instances \(v\) and \(u\). We use the cosine distance to measure similarity between transition types, which are each represented as a vector. Each vector value is a count of the number of times a given tag occurs to the left or right of the transition at indices \(\pm 1, \pm 2\) and \(\pm 3\). Transitions with similar distributional neighbours will have similar vectors and therefore a low angle and a high cosine value.

This clustering was run to induce 50, 100, 200, 300, 500, 700 and 900 separate clusters on the small POS tagging task. A sample of the resulting clustering is shown 7.5, which illustrates the homogeneity within each cluster. The last cluster contains mostly punctuation, while the first is composed of foreign words and the third contains verbal adjunctive operators.

The decoding accuracy results are shown in Figure 7.9 along with the accuracy when using a transition count threshold. This shows that the clustering approach is inferior to the simple threshold method, particularly when there are a small number of clusters.

---

\(^8\)http://glaros.dtc.umn.edu/gkhome/cluto
Figure 7.9. Accuracy for feature tying using transition clusters on the small POS tagging task. The ‘thresholded’ curve displays the results from Section 7.2.1 which uses a frequency based threshold for feature tying.

For the three smallest sizes of 50, 100 and 200 clusters, the performance degradation is statistically significant at $p < 0.05$, while the other points on the two curves are insignificantly different. This negative result can be explained in terms of the quality of the clustering: perhaps a more linguistically informed means of tying would improve the results. Alternatively, it might be explained by the fact that the threshold method produces singleton feature groups for the common transitions, while lumping together the remaining features. In contrast, the clustering method induces similar sized clusters with few singletons. A clustering which yielded more singletons might be more appropriate for this task. Tasks with sparser counts over labellings might benefit more from this clustering technique.

7.2.5 Semantic Role Labelling

The previous experiments all used graphs with small factors, as is commonplace for most CRF applications. The largest factors in these graphs link pairs of nodes, and therefore the complexity of standard inference is quadratic in the label set. The second main experiment demonstrates how tied-potentials can be used to allow efficient inference on graphs with very large factors. For such graphs the complexity of standard inference is much greater: $L^C$, where $L$ is the number of labels and $C$ is the maximum factor size. This complexity applies to exact inference in trees, as well as for a single round of loopy belief propagation.
in cyclic graphs. The exponential form of the complexity leads to intractable inference in general graphs, which may have very large factors.

This experiment uses a different natural language task, semantic role labelling (SRL; Palmer et al. (2005)): the task of identifying which groups of words act as arguments to a given verbal predicate and what role they fill – i.e. agent, patient, etc. This process allows the semantic content of the proposition to be expressed in predicate logic. The labelling task requires locating the argument word-spans and labelling these arguments with their role with respect to the predicate. The groups of words which form arguments are usually syntactic constituents, and as such the SRL task can be viewed as annotating the syntactic parse tree. An example sentence is given in Figure 7.10 showing its parse and semantic role labels for the verb sold. Here the yield of the first NP is assigned label A0 and the third NP is assigned label A1. These labels must be interpreted with respect to the Propbank entry for the commercial sense of the verb to sell, as shown in Figure 7.11. In the example sentence, the luxury auto maker is the seller and the things sold are 1,214 cars. The remaining arguments are unspecified in the sentence.

This task will benefit from rich and non-standard graphical structures. A standard application of a CRF might model this task as a chain with a node for each word, each being assigned a chunk-style argument label. The nodes would be connected using a Markov condition which would preclude the model from making use of long distance features, e.g., referencing words in non-adjacent constituents. In addition, the Markov condition will often also disallow features over all labels in a argument. For example, in order to include a feature over the labelling of “the luxury auto maker” we would need a 3rd order Markov order. For these reasons, we would like features (and therefore factors) over larger spans of text, where these spans correspond to constituent yields in the parse tree. Standard inference in these graphs is intractable due to the unbounded factor sizes.

Figure 7.10. Syntax tree labelled for semantic roles with respect to the predicate sell. The subscripts show the role labels, and the dotted and dashed edges are those which are pruned from the tree.
Roleset sell.01 "commerce: seller":
  Arg0: Seller
  Arg1: Thing Sold
  Arg2: Buyer
  Arg3: Price Paid
  Arg4: Benefactive

Figure 7.11. Proposition-bank entry for the verb ‘to sell’.

We use the SRL task description from the CoNLL 2005 shared task (Carreras and Márquez, 2005), which is based on the semantic role markup used in the Proposition Bank (Palmer et al., 2005). This corpus supplements the Wall Street Journal component of the Penn Treebank with coarse semantic role labels. This data set assigns to each syntactic constituent zero or one argument label for each content verb in the sentence. These labels signal core arguments, which have a specific semantic meaning for the predicate; adjunctive arguments, which take on general meaning for most verbs; and referential arguments, which themselves reference other predicate arguments.

**Verb specific arguments** Arguments with a specific semantic meaning for a verb are labelled $A0$-$A5$. The semantics of the roles corresponding to these numbered arguments are defined in the Proposition Bank frame for the predicate verb, but in general $A0$ maps to agent and $A1$ to patient or theme.

**Adjunctive arguments** General arguments that any verb may take are labelled $\text{ArgM-X}$, where $X$ is a function tag. There are a number of possible functions, such as location ($\text{LOC}$) and cause ($\text{CAU}$), which are described in (Palmer et al., 2005).

**Argument references** Predicate arguments that reference other predicate arguments.

For example, a phrase that referenced argument 1 would be labelled $R$-$A1$.

**Predicate verb** The predicate verb that defines the proposition being labelled is tagged $V$.

In this experiment, we only predict the core verb-specific arguments ($A0$-$A5$). These arguments form the main SRL task, while the remaining labels are non-essential to the task. We adopt the CoNLL 2005 task specification (Carreras and Márquez, 2005), where a predicted syntactic parse tree is provided.

We treat the task as one of labelling each word with a role label which indicates that the word is part of a constituent with this role. The syntax tree specifies the set of syntactic constituents, and each is used as a factor, connected to the node for all words in its yield. Figure 7.12 shows an example sentence in (a) and some induced syntactic factors in the factor graph in (b). The features for a factor were constrained to select only homogeneous
7.2. Experiments

labellings over its nodes. That is, only labellings entirely consisting of one label were selected, and the features for the remaining labellings were tied together. Therefore only \( L \) labellings (\( L \) is the number of labels) are detected instead of an exponential number in \( L \). These features can bias the model towards choosing uniform argument labels for all nodes in an argument constituent, while still allowing overlapping constituents to compete for dominance over the labelling.

For the sake of reducing unnecessary complexity, the factor graph was shrunk by merging adjacent nodes which were both members of the same set of syntactic factors, forcing them to be labelled together. This process is shown in Figure 7.12 (c) where the nodes for ‘the’ and ‘man’ are merged (both words belong to the same NP and VP constituents); as are ‘a’ and ‘telescope’. When the syntax tree is correct, this collapsing will have little effect, although in the presence of parse errors, it may conflate pairs of nodes with different role labels. This problem occurred very infrequently in the training sample; when this happened the new merged node was given a NULL label.

In addition, we introduced adjacency factors linking adjacent constituents (e.g., linking ‘I’ and ‘the’ to join NP\(_1\) and NP\(_2\)), representing a first order Markov assumption between adjacent argument candidates. These factors link the left-most node of each constituent with the right-most node of the last complete constituent. The features over adjacency factors were dense, considering all possible label pairs. Figure 7.12 (d) shows the four new adjacency factors with dashed edges and shaded factors (squares). Note that these factors link words in non-adjacent constituents, allowing the model to more precisely model long distance effects. Specifically this allows the model to disfavour labellings with repeated arguments (e.g., two A0 arguments).

The observation features were based on those in Pradhan et al. (2005), which included
syntactic paths, head words, syntactic category, etc. The feature templates used are summarised in Table 7.6. Some of the contextual features require further explanation. The verb voice feature uses a heuristic to detect the active or passive voice: simply testing for the verb tag VBN and searching for a form of ‘to be’ in the VP constituent. A passive verb typically has the agent and patient (A0 and A1) arguments in a different order to an active verb. The verb path feature traces a path through the syntax tree from the verb to current constituent or word. This path is simplified by taking only the first character from each category label, e.g., NNS becomes N, and PP becomes P. The path feature is the individual best feature despite often having extremely sparse counts. The coarse version provides a useful extension of the feature with denser counts while introducing only a small amount of noise.

These features are applied to single node factors combined with the node label, syntax node factors combined with a label (or a single ‘non-homogeneous’ value) and adjacency factors combined with a pair of labels. The features were attested on the training sample, as described in Chapter 4.

We also used simple pruning (Xue and Palmer, 2004) to remove irrelevant nodes from the parse tree before creating the factor graph. This pruning heuristic discarded those

---

**Table 7.6.** Features templates used for semantic role labelling. The final three column show the single node factors, syntax factors and adjacency factors. These are applied to each word, each constituent and pairs of constituents respectively. The adjacency factors conjoin the contextual features at either end of the transition.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>syntactic category ((s))</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>position relative to verb ((r)): left, right, is/contains</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>first/last word ((w_L, w_R)), category ((s_L, s_R)) and conj.</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>verb lemma ((v))</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>(v) conjunctions: with (s, w_L, w_R) and (r)</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>verb voice</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>verb path</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>coarse verb path</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>head word ((w_H)), category ((s_H)) and conj.</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>parent category ((s_P))</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>parent head word ((w_{P,H})), category ((s_{P,H})) and conj.</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>left sibling category ((s_&lt;))</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>left sibling head word ((w_{&lt;,H})), category ((s_{&lt;,H})) and conj.</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>right sibling category ((s_&gt;))</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>right sibling head word ((w_{&gt;,H})), category ((s_{&gt;,H})) and conj.</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>
Table 7.7. Semantic role labelling development results.

<table>
<thead>
<tr>
<th></th>
<th>Prec.</th>
<th>Recall</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>79.77</td>
<td>73.60</td>
<td>76.56</td>
</tr>
<tr>
<td>A0</td>
<td>85.09</td>
<td>80.63</td>
<td>82.80</td>
</tr>
<tr>
<td>A1</td>
<td>78.43</td>
<td>74.18</td>
<td>76.24</td>
</tr>
<tr>
<td>A2</td>
<td>68.86</td>
<td>55.87</td>
<td>61.69</td>
</tr>
<tr>
<td>A3</td>
<td>68.00</td>
<td>44.74</td>
<td>53.97</td>
</tr>
<tr>
<td>A4</td>
<td>82.22</td>
<td>56.92</td>
<td>67.27</td>
</tr>
<tr>
<td>A5</td>
<td>100.00</td>
<td>50.00</td>
<td>66.67</td>
</tr>
</tbody>
</table>

nodes in the parse tree which are either on the path from the verb to the root, or far away from that path. This simplified the structure of the factor graph, limiting both the size of the factors and the number of cycles, which is crucial for efficient and accurate belief propagation. This pruning heuristic only discarded a small proportion of argument constituents, and thus should not adversely affect accuracy. Most often this detrimental effect was caused by parse errors. In these cases the best features of the CRF will be incorrect or else swamped by noise, and therefore the CRF would most likely make incorrect predictions even without pruning. Figure 7.10 shows the effect of this pruning, where the dashed edges in the syntax tree are pruned. Note also that the constituents containing the verb (on the path from the verb to the root) are also excluded from the syntax tree, as these should not be classified as arguments.

This model was trained on the proposition-bank corpus (Palmer et al., 2005), using the CoNLL 2005 shared task dataset (Carreras and Márquez, 2005), considering only the core argument labels (A0 – A5), which roughly map to the roles of agent, patient, theme, etc. Sections 2-21 were used for training, and section 24 for development. The training set consisted of 39,832 sentences yielding 90,750 propositions and nearly 1 million tokens. The data was annotated with predicted part-of-speech tags and syntactic parses (Collins, 1999) as given in the CoNLL 2005 shared task (Carreras and Márquez, 2005). The accuracy on the development set is shown in Table 7.7. The precision and recall is quite high for A0 and A1, but deteriorates for many of the higher numbered arguments. This reflects the comparative rarity of these later labels and also their verb sense specific meanings: the features in the model assume that the argument labels are used with a consistent global meaning. Moreover the model does not attempt verb sense disambiguation or utilise the Propbank frames – with highly accurate sense disambiguation, more expressive features could be included leading to better modelling accuracy. These results can be roughly compared to (Roth and Yih, 2005), who applied (constrained) CRFs to the SRL argument labelling task. They report an $F_1$ result of 74.49, although on a different version of the data set.
Chapter 7. Efficient Inference using Tied Potentials

7.3 Related Work

Of the many approximate inference techniques which have been used to improve the efficiency of estimation of CRFs or related graphical models, the approach of Siddiqi and Moore (2005) is most similar to the tied-potentials method. Their method allows for fast inference in Hidden Markov Models, and was described on page 142. Siddiqi and Moore’s model requires the specification of the number of dense outgoing transitions, $K$, while our model instead requires explicit enumeration of these configurations. Therefore our model finds exact solutions to a constrained problem, in contrast to their approach which is an approximation. We show how this transition matrix structure can readily be exploited in general (exact or loopy) belief propagation for factor graphs, as opposed to forward-backward and Viterbi inference in chains.

CRFs have also been applied using a sparse transition matrix, where transitions unseen in the training sample are assumed impossible. This is implemented by fixing their potentials to zero (Charles Sutton, personal communication). While this approach can yield similar complexity improvements to our technique, the proliferation of zeros severely limits its usefulness in practise. While this approach also decreases the time complexity of the two algorithms, it often eliminates all possible interpretations for a novel instance. This is because novel transitions cannot be proposed during decoding and therefore the model may not be able to posit even a single analysis for such a sentence. This results in poorer empirical accuracy when compared to the unconstrained model, which often uses these unattested transitions in decoding. For instance, in POS tagging only 939 of the 967 transition types in the Penn Treebank test set are also in the training set. Using a smaller 5,000 sentence sample of the training set results in an average coverage of only 881. If the unseen transitions are rendered illegal then the model will be forced to make many errors on the test set. In contrast, tying the majority of potentials to a non-zero constant avoids the proliferation of zeros, while still realising the complexity improvements. The tied potentials technique combines the efficiency gains from the sparse transition matrix (here not being zeros, but a constant value) with the benefit of allowing the model to perform inference over the remaining labellings.

Similar feature based optimisation has been used for training log-linear language models (Wu and Khudanpur, 2000). For this task, the label space is extremely large (the vocabulary of words), and the optimisations presented allow efficient grouping of labels with similar feature sets. Our method also groups features, but in the context of belief propagation for globally normalised log-linear models.

Semi-Markov CRFs (Sarawagi and Cohen, 2004) are an extension to CRFs which combine a standard linear chain CRF with the ability to treat a contiguous nodes in the graph as a single node. This allows them to perform both a segmentation (a form of graph selection) and label prediction simultaneously. This configuration closely parallels the graph used for semantic role labelling in Section 7.2.5, which seeks to find a segmentation.
and labelling for a sequence of words. The semi-Markov CRF performs inference over a space of graphs specifying the different segmentations with each graph parameterised by a different set of features. This contrast with the tied model used here for SRL which uses a single graph (often with a high Markov order, defined by the syntactic constituents) in which the different segmentations are implied by the graph labelling and many of the features are shared between different segmentations. Critically, semi-Markov CRFs allow for exact forward-backward and Viterbi style inference without incurring a large increase in complexity compared inference in a linear chain CRF. This contrasts with the tied CRF used for SRL, which required the slower and approximate loopy belief propagation. Integrating these approaches by incorporating tied features into the semi-Markov CRFs framework might allow for considerably more efficient inference for segmentation tasks.

7.4 Conclusion

This chapter introduced a method of tying the features (or equivalently, the parameters) in a CRF. These constraints lead to a predictable structure in the clique potentials, in that many labellings must share the same potential value. Extensions to the sum- and max-product algorithms were presented which used this tied-potential structure to avoid explicit enumeration of the full space of labellings. This resulted in the following main findings:

1. Inference can be performed considerably faster, often with a reduced time complexity. The complexity of standard training is $O(LC)$ where $L$ is the number of labels and $C$ is the maximum clique size. With the tied approximations this becomes linear in the number of selected labellings, $O(S)$. The selected labellings are those with dense entries in the potential matrix, while the remaining labellings are those majority labellings which share the same potential value. For many tasks we expect that there will be many fewer selected labellings than the full set, i.e., $S \ll LC$. The approximations presented avoid considering individual remaining labellings when enumerating the labellings in belief propagation, leading to faster inference.

2. These approximations can reduce training and decoding time for many currently difficult tasks. This was shown for NER, POS tagging and joint POS tagging and NPC. This method allowed training of a CRF for POS tagging on the full training set, which was not possible for the standard training algorithm due to high time and memory requirements. Moreover, the accuracy of the tied model rivals the state-of-the-art.

3. Inference remains tractable for graphs with considerably larger cliques. This allows CRFs to be applied to previously impossible tasks, as shown for semantic role labelling with a very densely connected graph. This also allows the use of more
richly connected graphs for tasks in which standard CRF training is currently only possible using a simple chain.

4. There are many different ways in which features might be tied, and some of these were explored. The tying not only affects the complexity of inference but can affect decoding accuracy. The most naive method used a frequency threshold to exclude rare transitions, which was shown to reduce training time with little detrimental affect on accuracy. Feature tying can also allow the user to encode domain knowledge appropriate to the task, with similar results.

In summary, this method allows CRFs to be applied more efficiently to current tasks, with reduced complexity of training and decoding. Furthermore, it also allows the application of CRFs to tasks with considerably larger factors, in which many other training methods would prove intractable.
Chapter 8

Conclusions

This thesis investigates the efficiency and scaling aspects of Conditional Random Fields (CRFs; Lafferty et al. (2001)) for natural language processing (NLP). CRFs have proven very successful for many NLP tasks, however they suffer from two main failings. The first is that they often over-fit the training sample as a consequence of their expressive power. This problem can be reduced by using an appropriate prior or by other means of smoothing (Smith and Osborne, 2005). The second problem is that training a CRF is often very expensive, especially so for large tasks, such as those with densely connected graphs, many features, many labels or a large training set. Even for tasks with modestly sized label sets, such as part-of-speech tagging, training will often take weeks of processor time. This efficiency problem prevents the application of CRFs to many NLP tasks, or else necessitates the use of coarse approximations, which typically compromise accuracy.

This thesis focuses on the problem of training efficiency for CRFs, and makes three main contributions. Firstly, we provide an empirical analysis of training complexity, using standard training and various popular alternative training methods. Secondly, we present a novel method for CRF training using error-correcting output codes (ECOC), which decomposes training into a series of much simpler independent training tasks. Overall, ECOC often results in much faster training times, while achieving similar accuracy to standard training. The third contribution is another alternative method for training which imposes constraints on the model’s features. These feature constraints ensure a regularity in the potential functions, which can be exploited for faster sum- and max-product belief propagation inference. This leads to much faster training than standard methods for only a very minor reduction in accuracy. We now elaborate on each of these contributions in turn.

The experimental results for standard training show that the asymptotic time complexity of training in a linear chain CRF is quadratic in the size of the label set, linear in the number of features and almost quadratic in the size of the training sample. We also show that the cost of inference in cyclic graphs, such as lattice structured dynamic CRFs (Sutton et al., 2004), is even greater. This motivates the use of alternative training
methods that are more efficient and scale to larger tasks. Pseudolikelihood (Besag, 1975) and piecewise training (Sutton and McCallum, 2005) are both approximations which substantially reduce the training time. However, these methods can also significantly reduce the accuracy. We present a novel modification of the pseudolikelihood method which decomposes a lattice graph into separate layers using a pseudolikelihood assumption. This method results in significantly better accuracy than regular node-based pseudolikelihood and presents a promising method for approximate training of dynamic CRFs. Our experimentation also confirmed that averaged perceptron training (Collins, 2002a) is a particularly useful technique, often training in a short time and resulting in only a small accuracy loss relative to standard training.

The second major contribution of this work is a novel training method which uses error-correcting output coding (ECOC; Dietterich and Bakiri (1995)) to decompose training for a multiclass CRF (which models a task with three or more labels), into training a series of CRFs, where each models a binary labelling task. Training a series of binary CRFs on these sub-tasks can be overall much more efficient than training a full multiclass CRF. Decoding then combines the predictions of these binary models, either in a majority vote or using a product model formulation. This decoding process is the ‘error-correction’ step, where the final labelling decisions are robust to errors made by individual binary CRFs. The performance of the ECOC method relies on the error-correcting code, which provides the mapping between the full label space and the binary space. A good code specifies a diverse set of sub-tasks, and therefore results in a low error correlation between the constituent models. It also must ensure that the decision boundaries between labels are well represented, allowing the ensemble to make robust labelling decisions. These properties must be balanced with the code’s size: a code which specifies many sub-tasks will require a longer training time than a smaller code. We present results for a number of different types of codes: block codes used in communication, random codes, exhaustive codes, hand-engineered codes, and codes designed to reflect any bias in the label distribution. Large codes typically produce the best accuracy, which is quite similar to that of standard training. However, moving to a smaller code only incurs a small loss in accuracy whilst resulting in a large reduction in training time. Many small codes produce good accuracy coupled with short training times. These codes are often sized proportional to the label set or the logarithm of the label set. This translates into a complexity improvement over standard training, which has a quadratic complexity in the number of labels.

The third and final contribution is another new training method, tied potential training. This method imposes constraints on the features used in a CRF such that sets of labellings share the same feature values. This results in potential values which are identical (tied) for these labellings, and this structure can be exploited during sum- and max-product inference. The inference operations avoid the need to consider every labelling in a tied
set of labellings and thus if many labellings are tied, inference is considerably faster. This results in a complexity of training and decoding which is linear in the number of selected (non-tied) potentials. We show that even using very simple heuristics to constrain the features can result in much faster training and decoding, with only a mild negative effect on accuracy. Tying rare labellings, which occur only very few times in the training sample, greatly reduces the number of selected labellings. This in turn reduces the training time, and, to a lesser extent, the accuracy, allowing us to trade a small loss in accuracy for a large reduction in the cost of training. This has a similar effect to the choice of code in the ECOC method. Tying can also encode domain knowledge, and thereby reduce the training time without incurring an accuracy penalty. Most importantly, tied potential training allows for efficient inference on graphs with arbitrarily large cliques, as long as the set of selected labellings remains small. These graphs allow features to represent long distance phenomena, which are common in NLP, but are often poorly modelled.

8.1 Issues for Further Research

Many of the findings in this thesis are more generally applicable to other fields and applications. In addition, there are many possible extensions which could further improve the methods presented here. We now list some of these avenues:

1. Both of the novel training methods presented in the thesis could be combined with other alternative training methods. This might further reduce the cost of training, although it could also potentially compound the errors made by the individual methods. For instance, the ECOC method could use any of the alternative training methods presented in Chapter 5 to train each constituent model, reducing the overall cost of training. The tied potentials method could be used with piecewise training (Sutton and McCallum, 2005), where it would reduce the number of labellings considered, or perceptron training (Collins, 2002a), where it would reduce the cost of decoding each training instance. Moreover, the ECOC method could be used together with the tied potentials method, which would be particularly applicable to graphs with very large cliques, where even the cost of binary inference is impractical.

2. CRFs often over-fit the training sample; this effect can be reduced by using a prior, however other means of smoothing can often further reduce the problem (Smith and Osborne, 2005). The two training methods presented in this thesis could be used to smooth the distribution, and thereby improve accuracy. Some of the results from using these methods improved on the accuracy of standard training, which illustrates that these methods do already provide some regularisation. The accuracy of the ECOC method could be improved further by using more powerful codes, feature selection in each constituent model to promote diversity (Ricci and Aha,
1998), or using a weighted opinion pool to learn different voting weights for each constituent model (see Section 6.4.3). The tied potential method could also achieve further accuracy improvements, by using other means of feature tying such as those presented in Section 7.2.4.

3. The complexity analysis of standard training in Chapter 5 shows that standard training is polynomial in the size of the training set, i.e., CRF training will be exceedingly slow for problems with large amounts of training data. This provides a compelling motivation for using stochastic gradient methods for training (Vishwanathan et al., 2006). These are online optimisation methods which evaluate the gradient of the log-likelihood objective on a single training example (or on small batches) in order to approximate the true gradient over the full training set. These methods scale sub-linearly with the amount of training data, and therefore should converge much faster than batch gradient descent methods (e.g., L-BFGS) on large data sets.

4. ECOC training for CRFs could be improved in a number of ways. Section 6.5 describes how the manner of decomposition can encode a bias which makes it well suited to segmenting or chunking style tasks, but less appropriate for tasks with very rich and varied transitions. This problem could be addressed by using a different decomposition into constituent models (as suggested in Section 6.5.1), or by using other types of error-correcting codes, such as convolutional codes (as described in Section 6.5.2). These approaches would allow the constituent models to describe better the distribution over label sequences. In addition, ECOC CRF training could be improved by implementing methods known to improve ECOC classifiers, such as using better decoding methods (Kong and Diettrich, 1997), or explicitly promoting diversity between the models in the ensemble (Ricci and Aha, 1998).

5. In addition, many of the findings relating to ECOC for CRFs in Chapter 6 might also apply to ECOC for classification. For example, the product decoding method described in Section 6.2.3 could also be applied to ensembles of probabilistic classifiers. Moreover, these models could be individually weighted using a logarithmic opinion pool, as described in Section 6.4.3. For classification there would be a much lower cost of learning the constituent model weights than for CRFs, where this requires considerable computing resources and time. The analysis of error correlation in Section 6.4.2.1 would probably also hold for classifiers, in which case finding codes which minimise the number of shared decision boundaries would prove more effective than maximising column separation. The minimum loss bound code in Section 6.4.2.6 could also be applied to classification tasks with skewed label distributions.

6. Finally, ECOC could be applied to other structured labelling models, such as Hid-
den Markov Models or Markov Random Fields. This would involve only minimal changes to the approach described in Chapter 6, and could yield increases in accuracy and/or efficiency over the standard training methods.

8.2 Summary

In summary, this thesis motivates the need for alternative efficient methods for training CRFs, and presents two such methods. These new approaches compare favourably to existing alternative training methods, in terms of training time and accuracy. Together, these methods allow CRFs to be more efficiently applied to existing tasks, and most importantly allow the model to be scaled to new, more complex tasks. This will allow the many benefits of CRFs to be realised on a wider range of NLP tasks, while also supporting richer and more expressive feature sets and therefore more accurate modelling of language phenomena.
Bibliography


Empirical Methods in Natural Language Processing (EMNLP-02), page 18, Philadelphia, USA.


