Author/s:
Nguyen, XV;Monazam Erfani, S;Paisitkriangkrai, S;Bailey, J;Leckie, C;Ramamohanarao, K

Title:
Training Robust Models with Random Projection

Date:
2016

Citation:
Nguyen, X. V., Monazam Erfani, S., Paisitkriangkrai, S., Bailey, J., Leckie, C. & Ramamohanarao, K. (2016). Training Robust Models with Random Projection. International Conference on Pattern Recognition (ICPR), 0, pp.531-536. IEEE. https://doi.org/10.1109/ICPR.2016.7899688.

Persistent Link:
https://hdl.handle.net/11343/282461

Training Robust Models Using Random Projection

Nguyen Xuan Vinh Sarah Erfani Sakrapee Paisitkriangkrai
James Bailey Christopher Leckie Kotagiri Ramamohanarao
Department of Computing and Information Systems
The University of Melbourne, VIC 3010, Australia. Correspondence Email: vinh.nguyen@unimelb.edu.au

***Abstract*—Regularization plays an important role in machine learning systems. We propose a novel methodology for model regularization using random projection. We demonstrate the technique on neural networks, since such models usually comprise a very large number of parameters, calling for strong regularizers. It has been shown recently that neural networks are sensitive to two kinds of samples: (i) adversarial samples, which are generated by imperceptible perturbations of previously correctly-classified samples—yet the network will misclassify them; and (ii) fooling samples, which are completely unrecognizable, yet the network will classify them with extremely high confidence. In this paper, we show how robust neural networks can be trained using random projection. We show that while random projection acts as a strong regularizer, boosting model accuracy similar to other regularizers, such as weight decay and dropout, it is far more robust to adversarial noise and fooling samples. We further show that random projection also helps to improve the robustness of traditional classifiers, such as Random Forrest and Gradient Boosting Machines.**

Introduction

Regularization prevents machine learning systems from overfitting the training data. This is especially critical in the case of modern neural networks (NNs), which can comprise millions of parameters, usually much more than the available training data, thus requiring strong regularizers. While neural networks have recently received renewed interest due to their state-of-the-art performance in a range of challenging pattern recognition tasks, recent studies have shown their vulnerable properties w.r.t. *adversarial noise* and *fooling samples*.

Szegedy et al. [1] and Goodfellow et al. [2] show that it is possible to cause a NN to misclassify an image by applying a certain carefully-designed, imperceptible perturbation. In Figure 1, we give an example of this phenomenon, where we generated *adversarial samples* by applying a small amount of adversarial noise (imperceptible to humans) to the original hand digit MNIST data. Briefly, a neural network initially correctly classifies all the original digits (top row). Adding a small amount of adversarial noise (second row, intensity enhanced by a factor of 10 for visibility), the same network mis-classifies all the resulting visually-identical images (third row). As a dual problem to adversarial samples, Nguyen et al. [3] recently showed that for a given NN, it is possible to find a set of fooling samples that are completely unrecognizable, yet the network will classify them with extremely high confidence. We give an example in Figure 2, where for a NN trained on MNIST, we employed genetic algorithms to generate a set of fooling examples, which the network classifies with very high confidence (i.e., with maximum probability of 100% given to the predicted class label). More details on these experiments are given in the experimental evaluation section.

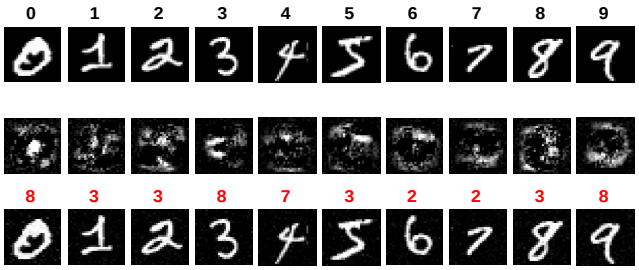


Figure 1. Adversarial samples. Top row: original images correctly classified. Second row: Adversarial perturbation (intensity enhanced by a factor of 10 for visibility). Third row: Adversarial samples misclassified (red labels).

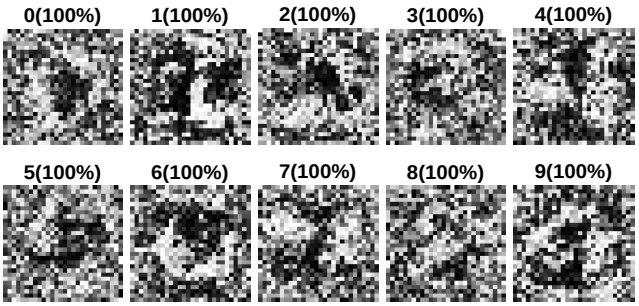


Figure 2. Fooling samples of digits from 0-9. All samples are classified with absolute confidence, yet are difficult for a human to recognise.

The practical implications of adversarial and fooling samples are far-reaching. We take self-driving vehicles as an example, in which NNs are a major component of the car's vision system. What would happen if some adversary modifies a STOP road sign, so that it looks identical to humans, yet the car mis-interprets it and thus fails to stop? And what would happen if some adversary places a fake road sign containing images similar to ones in Fig. 2, which does not make sense to humans, yet the car confidently interprets it as something else, such as a speed sign? These practical scenarios demonstrate the needs of having machine learning systems that are robust to adversarial and fooling samples.

In this paper, we show how robust NNs can be trained using a novel type of regularizer. Strong regularizers play a critical role in the success of machine learning systems, especially for modern NNs where the number of free parameters can far exceed the number of training samples. Regularization on NNs can be realized via L1, L2 (a.k.a. weight decay), gradient

and Hessian regularization [4]. One of the most successful recently invented regularization schemes is Dropout [5], which can be regarded as an approximate way to perform model averaging over a very large number of models, each being trained most likely for only one iteration. Alternatively, it can also be regarded as an implicit L2 regularization scheme [6] or noise injection [7]. At training time, the Dropout regularizer randomly eliminates 50% of the nodes in the hidden layers, which is believed to prevent complex co-adaptation of neurons while encouraging them to detect features that are generally useful in a wide variety of network topologies. DropConnect [8] generalizes dropout to dropping arbitrary sets of links (rather than dropping the entire set of links associated with the drop-out nodes). As we will experimentally demonstrate later, current regularizers for NNs are not robust against adversarial noise and fooling samples.

The novel regularizer we propose herein, named *the random projection regularizer*, as the name suggests, involves using random projection [9]. Random projection has been widely used across different domains as an efficient and effective method for dimensionality reduction. Herein, we exploit random projection from a non-conventional perspective, not for dimensionality reduction, but rather as a means to augment the training data set. More specifically, from an original training data set, we generate multiple projected versions of the data, all used to train a *single* NN. We show that the random projection regularizer boosts the accuracy of NNs similar to other regularization methods while *being far more robust to both adversarial noise and fooling samples*.

THE RANDOM PROJECTION REGULARIZER

We first start by reviewing the random projection lemma and elaborating the intuition behind the random projection regularizer. Random projection specifies how to efficiently project a high dimensional data set onto a lower dimensional space while faithfully preserving the topology of the original data to a large extent. Technically, the particular variant of the random projection lemma that we employ is stated as follows.

Lemma 1. *[9] Let an arbitrary set of n points in $\mathbb{R}^d$ be represented as an $n \times d$ matrix $\boldsymbol{X}$. Given $\epsilon, \beta > 0$, let $k_0 = (4+2\beta)\log n/(\epsilon^2/2 - \epsilon^3/3)$. For $k \geq k_0$, let $\boldsymbol{P}$ be a $d \times k$ random matrix with $\boldsymbol{P}(i,j)$ being independent random variables drawn from the following probability distribution*

$$\boldsymbol{P}(i,j) = \sqrt{3} \times \begin{cases} +1 & \textit{with probability } 1/6, \\ 0 & \textit{with probability } 2/3, \\ -1 & \textit{with probability } 1/6. \end{cases}$$

Let $\boldsymbol{X}' = \boldsymbol{XP}/\sqrt{k}$, and let $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ map the i-th row of $\boldsymbol{X}$ to the i-th row of $\boldsymbol{X}'$. With probability at least $1 - n^{-\beta}$, for all rows u, v in $\boldsymbol{X}$:

$$(1-\epsilon)\|u-v\|^2 \geq \|f(u) - f(v)\|^2 \geq (1+\epsilon)\|u-v\|^2$$

Although random projection has been almost exclusively employed as a tool for dimensionality reduction, herein, we are interested in random projection from a different angle: *to augment the data set used for training a NN with multiple perturbed versions of the original data*. With this vision in mind, we first briefly review data augmentation as a regularization strategy for machine learning.

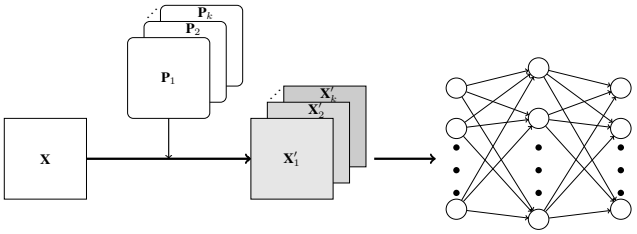


Figure 3. A schematic description of the proposed method. Projected batches of data $\mathbf{X}_i$'s are synthesized on-the-fly using a pre-determined set of random projection matrices $\mathbf{P}_i$'s and fed into a single model (a NN in this case).

Data Augmentation: Arguably, the most effective way of augmenting the training data is to collect more real life data. Unfortunately, this is not always feasible. There are several cost-effective methods to generate synthetic training data, which can be broadly classified as follows:

Domain-independent data augmentation: It has long been known that adding a small amount of noise to the training data improves the generalization ability of the model [10]. A previous result by Bishop [4] has shown that training with noise is equivalent to adding a regularization term to the objective function. For the sum-of-squares loss, the regularization term belongs to the class of generalized Tikhonov regularizers. Therefore, direct minimization of the regularized error function provides a practical alternative to training with noise. Similarly, an earlier work [11] showed that adding noise to inputs, outputs and weights encourages the NN output to be a smooth function of the input or its weights, respectively.

Domain-specific data augmentation: Various domain-specific methods for data augmentation have been exploited. In speech recognition for example, a cheap way to synthesize more training data is via adding background noise (e.g., restaurant noise, street noise) and varying the speaker's speed [12]. In image recognition, it is customary to crop, scale, flip, shear, perturb color channel intensity or take random patches of the input images [13].

Random Projection as a Data Augmentation Scheme: The approach involves training a single model, e.g., a NN, on an augmented data set comprising k projected versions of the original data $\mathbf{X}$, generated using a pre-defined set of random projection matrices $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$, as illustrated in Figure 3.

Note that, since our primary goal here is not dimensionality reduction, *we keep the original dimensionality of the data*. Also, we set $\mathbf{P}_1 = \mathbf{I}_{d \times d}$, so that an original version of the data is always kept. All the projected versions $\mathbf{X}'_1, \mathbf{X}'_2, \ldots, \mathbf{X}'_k$ are concatenated to form a new training data set of kn instances in $\mathbb{R}^d$ and fed into the neural network. At test time, predictions from k projected versions of the test data are averaged to produce the final prediction. More formally, let $f(\mathbf{x}, \mathbf{W})$ be the transfer function of a NN with a specified architecture

having weights $\mathbf{W}$, then the transfer function of the same NN using RP-data augmentation can be written as:

$$f_{RP} = \frac{1}{k}\sum_{i=1}^{k} f(\mathbf{x}\mathbf{P}_i, \mathbf{W}) \quad (1)$$

Using random projection to augment the training data is a radically different approach from existing data augmentation methodologies. It can be seen that existing data augmentation methods share a common characteristic: they create slightly perturbed versions that are close to the original data point, either in $\mathbb{R}^d$ (as by adding random Gaussian noise), or in the intrinsic, usually low-dimensional, manifold on which the data actually resides (as by flipping or cropping the image). Random projection, on the other hand, produces slightly perturbed versions of the original dataset as a whole, but not minor perturbations of each individual data point. Each of these projected versions, while being topologically similar to the original dataset, can potentially reside on different manifolds in the same $\mathbb{R}^d$ space. Note that since we have exercised no control over where the randomly projected data will reside, it might so happen that any two of these projected versions overlap and potentially contaminate each other's class structure. We resolve this situation by simply adding an additional 'index' feature, which is assigned the value i/k for samples coming from the i-th projected version. This ensures that different projected versions of the data lie on different linearly-separable manifolds.

Note that our proposed model is not an ensemble of neural networks: it involves training only *a single neural network*, which is fundamentally different from training k separate networks from k projections of the data, then performing model ensembling, e.g., via majority voting. We refer to this model as Ensemble(k). For the latter model, each individual neural network is still trained on the same amount of data as the original data, thus is still prone to over-fitting. The idea of training ensemble models with random projection in fact has been previously explored in the context of random forests [14] and linear discriminant analysis [15]. The regularization effect of random projection on an ensemble of Fisher linear discriminant classifiers has also been analyzed in [15].

Parameter Sharing: Our approach, on the other hand, can be seen as a way to perform ensembling and regularization with a single learner. Observing that the k networks trained on k projections of the same data should have more similarity than difference, enforcing weight-sharing between networks could provide an effective regularization scheme. By training only a single network using all k projections, we naturally achieve weight-sharing across tasks and this brings about the desired regularization effect. Note that modern neural networks often contain many more parameters than available data, making them highly expressive and ensuring that all tasks can be accomplished, i.e., learning decision boundaries for all projections.

Computational Complexity: The proposed method involves training (and testing) a NN on an augmented data set k times larger than the original data. Therefore, for each training (testing) batch, it takes theoretically k times as much processing time. For NNs trained on GPUs however, we have observed that: (i) the total wall-clock training time usually increases by less than k-fold, thanks to the increased efficiency of GPUs when processing larger training batches, and (ii) training with the random projection regularizer typically takes fewer epochs to converge compared to other regularizers such as Dropout. In addition, $O(knd^2)$ time is required for generating the projected data on-the-fly. This can be done effectively in parallel on the CPU, while the GPU is busy processing the previous batch.

ROBUSTNESS

In this section, we first explain the mechanisms used to generate adversarial and fooling samples, and give insights on how Random Projection affects these mechanisms.

Adversarial Noise: Let $\mathcal{L}(\mathbf{x}, C, \mathbf{W})$ denote the loss function of a NN w.r.t. the input $\mathbf{x} \in \mathbb{R}^d$, input label C and weights $\mathbf{W}$. At training time, $\mathbf{W}$ is varied to find the optimal network parameters. Given a fixed $\mathbf{W}$, at test time, our goal is to find, for each input $\mathbf{x}$, an adversarial noise vector $\boldsymbol{\epsilon}$, such that the loss function $\mathcal{L}(\mathbf{x}+\boldsymbol{\epsilon}, C, \mathbf{W})$ is maximized. Furthermore, we also would like the error to be small. Szegedy et al. [1] proposed to find adversarial samples via solving a box-constrained optimization problem using L-BFGS. Herein, we propose a simple method to generate adversarial samples, by taking a small step in the positive gradient direction $\nabla_{\mathbf{x}}\mathcal{L}$. Thus, the adversarial noise for each sample can be defined as:

$$\boldsymbol{\epsilon} = \lambda \frac{\|\mathbf{x}\|}{\|\nabla_{\mathbf{x}}\mathcal{L}\|}\nabla_{\mathbf{x}}\mathcal{L} \quad (2)$$

Note that we have rescaled the gradient to the same norm as the input $\mathbf{x}$, thus the parameter λ can be meaningfully interpreted as the strength of the adversarial perturbation relative to the norm of $\mathbf{x}$. Also, we define the random noise vector $\boldsymbol{\epsilon}_r$ as a random permutation of $\boldsymbol{\epsilon}$, i.e., keeping the noise strength, just changing the direction.

Fooling Samples: We first give a formal definition for the confidence of a prediction:

Definition 1. *Given a NN with softmax output, the* ***confidence*** *of a prediction is the probability of assigning the object to the predicted class label.*

The confidence thus ranges within $[0, 1]$, with a value of 1 for absolute confidence. Nguyen et al. [3] proposed using Genetic Algorithms (GA) to search for fooling samples, by evolving a population of samples with the fitness function being the confidence of prediction for a class of interest. Following this approach, we employ Genetic Algorithms with real number encoding and a Gaussian mutation operator to generate fooling samples similar to the ones in Fig. 2.

An intriguing characteristic of both adversarial and fooling samples is that they *generalize well across different NN models trained even on different datasets* [1]. This generalization property is particularly concerning, since it is possible for the attacker to craft attacks on a target machine learning system,

without full knowledge of its internal details. In this paper, we indeed adopt this assumption, that the attacker does not know the internal parameters of the target systems and its training data, but does have access to the same data distribution.

Under this assumption, random projection can be regarded as a strategy for obfuscating the original training data. Note that both the adversarial noise and fooling samples are generally designed via an optimization process, such as gradient ascent in (2). Random projection perturbs the direction of the adversarial noise vectors, hence making them less effective. In the next section, we carry out experiments to verify this hypothesis.

Experimental Evaluation

Using NNs, we first compare the Random Projection regularizer with Dropout and L2 regularizers in terms of classification accuracy. Next, we compare the regularizers w.r.t. their robustness to adversarial noise and fooling samples. Our NN implementation is based on the Matconvnet toolbox [16] running on a dual Xeon processor server with 64GB of memory and a Tesla K40 GPU. All neural networks comprise fully connected layers.

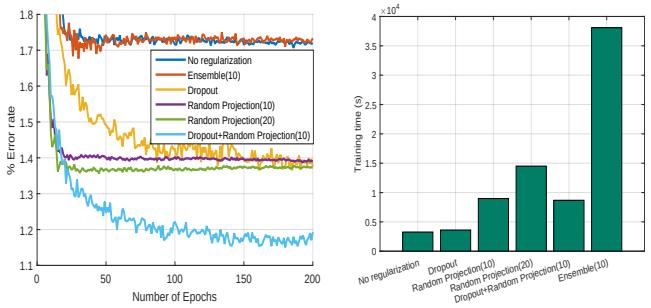
Regularization Effects

MNIST: We test the regularizers with different network architectures on the well known MNIST data set. In this experiment, we compare the Random Projection (RP) regularizer with 10 and 20 projections against a vanilla network (i.e., no regularization), Dropout, and also an ensemble of 10 vanilla NNs, Ensemble(10), on 10 random projections.

Network structure: We tested 6 different network structures with 2-3 hidden layers consisting of $\{800-800\}$, $\{1024-1024-1024\}$, $\{2048-2048\}$, $\{2048-2048-2048\}$, $\{4096-4096\}$ and $\{8192-8192\}$ nodes. For the Ensemble model, however, we only tested 3 network structures being $\{800-800\}$, $\{2048-2048\}$ and $\{4096-4096\}$ since training these ensembles is very time consuming. In all networks, rectified linear units (ReLU) were used. The input layer comprises 784 original features (+1 index feature if the RP regularizer was used), and 10 softmax output nodes.

Hyper parameters setting: For all networks, the learning rate was set to 10^{-2} and the momentum was set to 0.9. All networks were trained for 200 epochs on the training data set comprising 60K samples. The test error rate was reported for the 10K test samples. No other additional form of regularization (e.g., weight decay or early stopping) was employed. The MNIST data was rescaled to [0,1] and no further pre-processing was carried out.

In Figure 4(a), we report the average test error rate over all network structures. Due to lack of space, we do not provide individual plots for different network structures. The trend is however very similar. The difference in performance between different network structures was also observed to be minor. The vanilla NN models without any regularization achieve $\sim 1.75\%$ error rate, which is similar to previously reported results [5]. Dropout brings the test error rate down to $\sim 1.4\%$. The Random Projection regularizers with 10 and



(a) Average test error rate over all network structures (best viewed in color)

(b) Training time on the $\{800-800\}$ network for 200 epochs

Figure 4. Experiments on the MNIST data set

20 projections both bring the test error rate down to $\sim 1.4\%$, similar to dropout. We note that the Ensemble models with 10 vanilla networks—Ensemble(10)—trained on 10 random projections (including the original data) perform no better than a single vanilla NN without regularization. Each NN in the ensemble is still prone to overfitting and ensembling these overfitted-NNs did not bring about much regularization effect. The best performing model in our test was a combination of Dropout and the Random Projection regularizer, which achieves a test error rate of slightly less than 1.2% on average. It is noted that this result on the MNIST is competitive for non-convolutional network architectures.

A typical training time pattern is presented in Fig. 4(b). In general, the Random Projection regularizer required more training time, but is less than linear in the number of projections k. In particular, 10 and 20 projections require $\sim$3x and $\sim$5x training time, respectively. This is mainly due to the improved efficiency of GPUs when processing larger batches of data. An important remark is that while each epoch of the RP regularizer is more expensive, it generally takes fewer epochs to converge compared to the vanilla model, ensemble model and dropout. This is evident in Fig. 4(a). It takes the RP regularizer only 25 epochs to reach an error rate of 1.4% and then stabilize, while dropout required more than 150 epochs.

Other Data

We also tested different regularizers on a collection of 13 data sets from the UCI repository [17]. Due to space limitations, the details of these experiments are presented in the accompanying online supplementary material[1]. Overall, it is observed that the Random Projection regularizers perform competitively, either as a stand-alone regularizer or in conjunction with dropout. Furthermore, it is worth noting that these regularizers do not exclude the use of each other, and therefore can be used in tandem. From the experimental evidence, we promote the Random Projection regularizer as a new tool to add into existing NN toolkits.

Robustness to Adversarial Noise

In this section, we test the robustness of the regularizers against adversarial noise. We set up an experiment with the

[1]Also available at https://sites.google.com/site/icpr16supp/

Table I
% TEST ERROR RATE UNDER RANDOM NOISE AND ADVERSARIAL NOISE ON THE MNIST DATA

Noise level (λ)	Net 1 (Dropout)		Net 2											
			No Regularization		L2(5e-4)		Dropout		Ensemble(10)		RP(10)		RP(20)	
	RN	AN	RN	AN	RN	AN	RN	AN	RN	AN	RN	AN	RN	AN
0%	1.94	1.94	2.68	2.68	2.56	2.56	2.11	2.11	2.59	2.59	2.12	2.12	2.07	**2.07**
1%	1.96	2.71	2.69	3.11	2.57	3.05	2.10	2.47	2.59	2.95	2.14	2.51	2.08	**2.43**
5%	1.97	9.15	2.70	6.69	2.62	6.34	2.10	5.11	2.52	6.42	2.15	4.62	2.08	**4.40**
10%	2.03	29.62	2.71	17.58	2.62	15.40	2.09	15.59	2.45	12.71	2.14	9.81	2.11	**9.02**
20%	2.16	75.50	2.91	54.40	2.69	49.48	2.21	42.55	2.43	50.73	2.28	32.80	2.20	**30.38**

RN: Random noise; AN: Adversarial noise

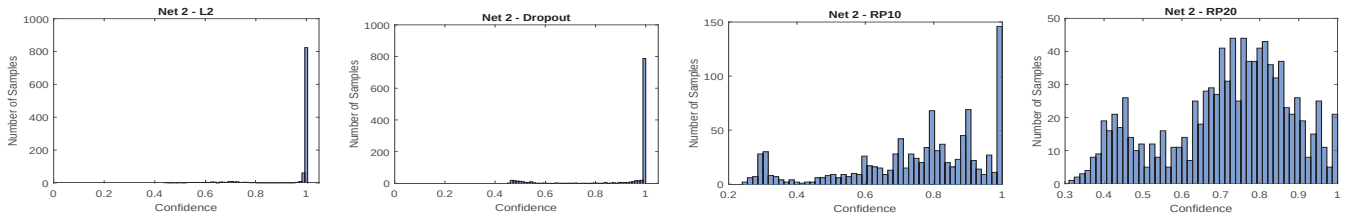


Figure 5. Confidence Distribution for Fooling Samples on MNIST dataset. L2 and Dropout produces many highly confident predictions, while with RPs, many fooling samples are predicted with low confidence.

MNIST dataset as follows:

- **Net-1**: 2 hidden layers of $\{800-800\}$ ReLU units, trained on 30K training samples with dropout.

- **Net-2**: 3 hidden layers of $\{1024-1024-1024\}$ ReLU units, trained on the remaining 30K training samples.

Our goal is to use Net-1 to design adversarial noise for the 10K test samples, and bring these adversarial samples to disrupt the prediction of Net-2 (of which no details were used for generating the adversarial samples). The two networks were trained for 50 epochs, with learning rate 1e-2 and momentum 0.9. We aim to test the robustness of different regularizers for Net-2 against adversarial noise at different noise levels.

From Table I, it can be observed that adversarial noise has a marked effect on Net-1, pushing the error rate to as high as $\sim 75\%$. On the other hand, *random noise of the same magnitude hardly has any effect at all* (herein, we report the average error rate for 10 random permutations of the adversarial noise). Net-2 is less affected compared to Net-1 (since these adversarial samples were specifically tailored for Net-1). Nevertheless, *adversarial samples generalize well to Net-2*, which was trained on a different dataset with a different architecture. The effect of different regularizers can be observed. In particular, weight decay, dropout, Random Projection 10 and Random Projection 20 all improve Net 2 robustness against adversarial samples, with increasing effectiveness in the respective order.

Robustness to Fooling Samples

In this section, we test the robustness of different regularizers w.r.t. fooling samples, which are unrecognizable objects specifically crafted to trick NN models to classify the samples to a certain class with very high confidence. Herein, we carry out an experiment on the MNIST dataset with the same Net-1 and Net-2 described in the previous section. Our goal is to use Net-1 to design a set of fooling samples, then bring these samples to fool Net-2, of which no details were known to the fooling-samples generator. Using genetic algorithms and Net-1, we generate 100 samples for each digit from 0-9, thus 1000 samples in total, similar to ones in Fig. 2. All these fooling 'digits' were classified with 100% confidence by Net-1 to the respective classes. We now bring these digits to Net-2 and test the robustness of different regularizers w.r.t. these fooling samples.

Table II
% OF MNIST FOOLING DIGITS CLASSIFIED WITH VERY HIGH CONFIDENCE

Confidence	Net-1	Net-2					
		No Reg.	L2(5e-4)	Dropout	Ensemble(10)	RP(10)	RP(20)
$\geq 99\%$	100%	88.1%	85.4%	78.5%	45.5%	13.5%	**1.9%**
$\geq 90\%$	100%	90.1%	90.0%	87.1%	73.5%	26.0%	**10.8%**

It can be observed from Table II that fooling samples generalize well from Net-1 to Net-2. Without regularization, more than 88% of the fooling samples are still classified by Net-2 with confidence $\geq 99\%$, i.e., almost absolute confidence. L2 regularization and Dropout both improve the situation only slightly. The Random Projection regularizers are by far the most robust against fooling samples, with only 13.5% and 1.9% of samples classified with $\geq 99\%$ confidence when 10 and 20 projections are employed, respectively. From the confidence distribution in Fig. 5, it can be observed that unlike L2 and Dropout regularizers for which the majority of predictions still have very high confidence, for the Random Projection regularizers RP(10) and RP(20), many fooling samples are now classified with confidence in the low range.

Other Classifiers

To demonstrate the universality of adversarial samples as well as the Random Projection Regularizer, in this section, we again carry out the adversarial experiment in the previous

section, but this time, replacing **Net-2** with two different learners: the Random Forest (RF) and Gradient Boosting Machines (GBM) [18]. That is, we still keep the NN **Net-1** trained on the first 30K samples to generate the 10K adversarial test samples, while the target systems are now RF and GBM, both trained on the other 30K samples. When Random Projection is employed, RF and GBM take the place of the NN in Fig. 3, where each decision tree in the ensemble is trained on the concatenated data set of all projected versions of the data. Note that both RF and GBM are ensemble methods that are remarkably popular and successful in practice. In [19], via benchmarking 179 classifiers on 121 UCI datasets, Delgado et al. concluded that the RFs are among the most successful ones. GBMs on the other hand, have been used to win many Kaggle competitions. In this experiment, we employed RFs with 500 trees using the Gini splitting criterion, achieving a test error rate of 3.49%, while GBMs were employed with learning rate 0.5 and 500 base learners, achieving a 3.13% test error rate. From the experimental results in Table III, two remarks are in order. First, we note the surprisingly high effectiveness of adversarial samples across classifiers which are of markedly different characteristics. Indeed, both RFs and GBMs employ decision trees as the base learners, which are not trained with gradient descent like NNs. Yet, adversarial noise (generated by NN **Net-1**) pushes the error rates of these ensembles up to $\sim$68% and $\sim$77% respectively. Second, it is observed that Random Projection significantly boosts the resistance of RFs and GBMs against adversarial manipulation, reducing their error rates from $\sim$68% to $\sim$17%, and from $\sim$77% to $\sim$23% respectively (on the 20%-noise adversarial test set).

Table III
% ERROR RATE OF RF AND GBM ON ADVERSARIAL SAMPLES

Method	**Adversarial noise level**				
	0%	1%	5%	10%	20%
GBM	3.13	39.6	59.8	70.4	76.8
GBM+1RP	2.85	4.52	6.55	10.6	25.7
GBM+5RP	2.95	3.61	5.45	8.84	22.91
GBM+10RP	2.94	3.42	5.30	8.99	22.68
RF	3.49	10.65	42.66	55.44	68.03
RF+1RP	4.02	5.63	8.77	12.60	21.63
RF+5RP	4.75	5.29	6.99	9.89	17.33
RF+10RP	4.87	5.39	6.98	9.54	17.04

DISCUSSION AND CONCLUSION

We have presented the proof-of-concept for a novel regularization scheme: the Random Projection regularizer that makes use of random projection as a domain-independent data augmentation scheme. On neural networks, our experiments with the RP regularizers on a variety of real data sets show promising results with respect to classification accuracy and robustness to adversarial and fooling samples, compared to other regularization schemes, such as Dropout or weight decay. For Random Forests and Gradient Boosting Machines, we demonstrate that Random Projection significantly boosts the model's resistance against adversarial manipulation.

We note that in the context of neural networks, while the application of the RP regularizer is straightforward for fully-connected networks, its application on convolutional architectures requires further consideration. This is because the traditional random projection operator will generally destroy the spatial correlation within the data, e.g., applying RP on a natural image will result in a non-image-like projected version, without any correlation between adjacent pixels. In this regard, our future research will investigate two possible approaches: (i) One can develop a RP operator specifically designed for data with high local spatial feature correlation, such as images, so that the projected data points possess the same characteristic, or (ii) one can insert a 'Random Projection layer' in the middle of the network, in-between convolutional layers and fully-connected layers. The results of these investigations will be presented on a separate occasion.

ACKNOWLEDGMENTS

This work is supported by the Australian Research Council via grant numbers FT110100112 and DP140101969. Vinh Nguyen supported by a University of Melbourne ECR grant.

REFERENCES

[1] C. Szegedy et al., "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.
[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2014.
[3] A. Nguyen et al., "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *CVPR*, 2015.
[4] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, Jan. 1995.
[5] N. Srivastava et al., "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, pp. 1929–1958, 2014.
[6] S. Wager, S. Wang, and P. Liang, "Dropout training as adaptive regularization," in *NIPS*, 2013, pp. 351–359.
[7] X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," *CoRR*, vol. abs/1506.08700, 2016.
[8] L. Wan, M. Zeiler, S. Zhang, Y. Lecun, and R. Fergus, "Regularization of neural networks using dropconnect," in *ICML*, 2013.
[9] D. Achlioptas, "Database-friendly Random Projections: Johnson-Lindenstrauss with Binary Coins," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 671–687, Jun. 2003.
[10] S. Rifai et al., "Adding noise to the input of a model trained with a regularized objective," *CoRR*, vol. abs/1104.3250, 2011.
[11] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural Comput.*, vol. 8, no. 3, pp. 643–674, Apr. 1996.
[12] A. Y. Hannun et al., "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014.
[13] A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1106–1114.
[14] A. Schclar and L. Rokach, "Random projection ensemble classifiers," in *Enterprise Information Systems*, 2009, vol. 24, pp. 309–316.
[15] R. J. Durrant and A. Kaban, "Random projections as regularizers: learning a linear discriminant from fewer observations than dimensions," *Machine Learning*, vol. 99, no. 2, pp. 257–286, 2015.
[16] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," *CoRR*, vol. abs/1412.4564, 2014.
[17] K. Bache and M. Lichman, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2013.
[18] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.
[19] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *JMLR*, vol. 15, pp. 3133–3181, 2014.