LEARNING WITH ADVERSARY

Ruitong Huang, Bing Xu, Dale Schuurmans & Csaba Szepesvári
Department of Computer Science
University of Alberta
Edmonton, AB T6G2E8 Canada
{ruitong, bx3, daes, szepesva}@ualberta.ca

ABSTRACT

In this paper, we propose a method, learning with adversary, to learn a robust network. Our method takes finding adversarial examples as its mediate step. A new and simple way of finding adversarial examples are presented and experimentally shown to be more 'efficient'. Lastly, experimental results shows our learning method greatly improves the robustness of the learned network.

1 INTRODUCTION

Deep Neural Network (DNN) models have shown its powerful learning capacity on many visual and speech classification problems. (Krizhevsky et al., 2012; Hinton et al., 2012) Part of reason is believed to be its high expressiveness from the deep architecture. Even though the misclassification rate is the main performance metric used to evaluate classifiers, robustness is also a highly desirable property. In particular, a classifier is expected to be 'smooth' so that a small perturbation of a datapoint does not change the prediction of the model. However, a recent intriguing discovery suggests that DNN models do not have such property of robustness. Szegedy et al. (2013) A well performed DNN model may misclassify most of the datapoints because of a human-indistinguishable perturbation on the original dataset. We call the perturbed dataset 'adversarial examples'. It is even more curious that a same set of such adversarial examples are consistently misclassified by a large group of DNN models which are learned with different architectures and hyperparameters.

Following the paper of (Szegedy et al., 2013), more and more attentions have been attracted toward such curious 'adversary phenomenon' in the deep learning community. (Szegedy et al., 2013; Goodfellow et al., 2014; Fawzi et al., 2015; Miyato et al., 2015; Nøkland, 2015; Tabacof and Valle, 2015). Goodfellow et al. (2014) suggests that the reason that cause the existence of adversarial examples may be its linearity of the model in high dimension. Further exploration is conduct by Tabacof and Valle (2015), showing that in an image domain, adversarial images inhabit large "adversarial pockets" in the pixel space. Based on these observations, different ways of finding adversarial examples are proposed, among which the most relevant one is proposed in the paper of Goodfellow et al. (2014) where a linear approximation is used and thus it does not require to solve an optimization problem. In this paper, we further investigate this problem, and proposed another simple way of finding adversarial examples. Experimental results suggest that our method is more efficient in the sense that DNN has worse performance under same magnitude of perturbation.

The main contribution of this paper is learn a robust classifier that can still maintain a high classification performance. Goodfellow et al. (2014) suggest using a new objective function that is a combination of the original one and the one after the datapoints are perturbed to improve the robustness of the network. While in (Nøkland, 2015), as a specific case of the method in (Goodfellow et al., 2014), it is suggested to only use the objective function that is defined on the perturbed data. However, there is no theoretical analysis to justify that the learned classifier is indeed robust. In particular, both methods are proposed heuristically. Also, the perturbed datapoints that are used to evaluate the robustness of the learned classifier in the experiments are generated from the original network that is trained with clean data. It would have been more convincing if such perturbed datapoints were generated from the current classifier, if one is to argue that the current classifier is robust. Recently a theoretical exploration about the robustness of classifiers in (Fawzi et al., 2015) suggest that, as expected, there is a trade-off between the expressive power and the robustness. This paper can be consider as a further exploration about this trade-off in the engineering side. We formulate the learning procedure as a min-max problem so that it forces the DNN model to prepare for the worst situation. In particular, we allow an adversary to play different perturbation on each datapoint, then the learning procedure is to minimize the misclassification error on the intendedly perturbed dataset. We call such learning procedure as 'learning with adversary'. It turns out that an efficient way of finding such adversarial examples is required, as an intermediate step, to solve such min-max problem, which goes back to the first part of our paper. We observe that our learning procedure turns out to be very similar to the one proposed in (Nøkland, 2015), while both works are conduct totally independently from different understandings of this problem.

The organization of the rest of this paper is as follows: we propose our method of find adversarial examples in Section 2. Section 3 is devoted to our main contribution: Learning with adversary. Finally we present our experimental results on MNIST and CIFAR-10 in Section 4.

1.1 NOTATIONS

We denote the samples by $\underline{Z} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. Let K be the number of classes in our classification problem. The loss function that is used for training is denoted by ℓ. Given a norm $\|\cdot\|$, let $\|\cdot\|_*$ denote its duel norm that $\|u\|_* = \max_{\|v\| \le 1} < u,\, v >$. Denote the network by $\mathcal{N}$ whose last layer is a softmax layer $g(x) \triangleq \alpha = (\alpha_1, \ldots, \alpha_K)$.

2 FINDING ADVERSARIAL EXAMPLES

Consider a network that uses softmax as its last layer for classification, denoted by $\mathcal{N}$. Given an sample $(x, y) \in \mathcal{X} \times \{1, 2, \ldots, T\}$ such that $\mathcal{N}(x) = y$, where y is the true label for x. Our goal is to find a small perturbation $r \in \mathcal{X}$ so that $\mathcal{N}(X + r) \neq y$. This problem is first investigated in (Szegedy et al., 2013) which proposes the following learning procedure: given x,

$$\begin{aligned} &\min_r \|r\| \\ s.t. \quad &\mathcal{N}(X + r) \neq \mathcal{N}(X) \end{aligned}$$

Our simple method to find such perturbation r is based on the linear approximation of $g(x)$, $\hat{g}(x + r) = g(x) + Tr$, where $T = \frac{\partial g}{\partial w}|_x$ is the derivative matrix.

Consider the following question: for a fixed index $j \neq J$, what is the minimal $r_{(j)}$ satisfying $\mathcal{N}(x + r_{(j)}) = j$? Replacing g by its linear approximation $\hat{g}$, one of the necessary condition for such perturbation r is:

$$T_j r_{(j)} - T_J r_{(j)} \le \alpha_J - \alpha_j,$$

where T_j is the j-th row of T. Therefore, the norm of the optimal $r^*_{(j)}$ is greater than the following objective value:

$$\begin{aligned} &\min_{r_{(j)}} \|r_{(j)}\| \\ s.t.\, &T_j r_{(j)} - T_J r_{(j)} \le \alpha_J - \alpha_j. \end{aligned} \tag{1}$$

Optimal solution to this problem is provided in Proposition 2.1.

Proposition 2.1. *It is straight forward that the optimal objective value is* $\|r_{(j)}\| = \frac{\alpha_J - \alpha_j}{\|T_j - T_J\|_*}$. *The optimal* $r^*_{(j)}$ *for common norms are :*

1. *If* $\|\cdot\|$ *is* L_2 *norm, then* $r^*_{(j)} = \frac{\alpha_J - \alpha_j}{\|T_j - T_J\|_2^2}(T_j - T_J)$;
2. *If* $\|\cdot\|$ *is* L_∞ *norm, then* $r^*_{(j)} = \frac{\alpha_J - \alpha_j}{\|T_j - T_J\|_1} sign(T_j - T_J)$;
3. *If* $\|\cdot\|$ *is* L_1 *norm, then* $r^*_{(j)} = \frac{c}{\|T_j - T_J\|_\infty} e_I$ *where* I *satisfies* $|(T_j - T_J)_I| = \|T_j - T_J\|_\infty$. *Here* V_I *is the* I*-th element of* V.

However, such $r^*_{(j)}$ is necessary but NOT sufficient to guarantee that $\operatorname{argmax}_i \hat{g}(x + r_{(j)})_i = j$. The following proposition shows that in order to have $\hat{g}$ make wrong prediction, it is enough to use the minimal one among all the $r^*_{(j)}$'s.

Proposition 2.2. *Let $I = \operatorname{argmin}_i \|r^*_{(i)}\|$. Then r^*_I is the solution of the following problem:*

$$\min_r \|r\|$$
$$s.t. \operatorname*{argmax}_i \left(\hat{g}(X+r)\right)_i \neq J.$$

Putting all things together, we have an algorithm to find adversarial examples, as shown in Algorithm 1.

Algorithm 1 Finding Adversarial Examples

input (x, y); Network $\mathcal{N}$;
output r

Compute T by performing forward-backward propagation from the input layer to the softmax layer $g(x)$
for $j = 1, 2, \ldots, d$ **do**
    Compute $r^*_{(j)}$ for Equation (1)
end for
Return $r = r^*_{(I)}$ where $I = \operatorname{argmin}_i \|r^*_{(i)}\|$.

3 TOWARD THE ROBUSTNESS OF NEURAL NETWORK

We enhance the robustness of the neural network by preparing the network for the worst cases, as follows.

$$\min_f \sum_i \max_{\|r^{(i)}\| \leq c} \ell(g(x_i + r^{(i)}), y_i). \tag{2}$$

The hyperparameter c that control the magnitude of the perturbation needs to be tuned. Note that when $\ell(g(x_i + r^{(i)}), y_i) = \mathbb{I}_{(\max_j (g(x_i + r^{(i)})_j) \neq y_i)}$, the objective function is the misclassification error under perturbations. Oftentimes, ℓ is a surrogate loss which is differentiable and smooth. Let $L_i(g) = \max_{\|r^{(i)}\|_2 \leq c} \ell(g(x_i + r^{(i)}), y_i)$. Thus the problem is to find $f^* = \arg\min_f \sum_i L_i(f)$.

To solve the problem (2) using SGD, one need to compute the derivative of L_i with respect to f. The following proposition suggest a way of computing this derivative.

Proposition 3.1. *Given $f : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$ differentiable almost everywhere, define $L(v) = \max_{u \in \mathcal{U}} f(u, v)$. Assume that L is uniformly Lipschitz-continuous as a function of v, then the following results holds almost everywhere:*

$$\frac{\partial L}{\partial v}(v_0) = \frac{\partial f}{\partial v}(u^*, v_0),$$

where $u^* = \arg\max_u f(u, v_0)$.

Proof. Note L is uniformly Lipschitz-continuous, therefore by Rademacher's theorem, L is differentiable almost everywhere. For v_0 where L is differentiable, the Fréchet subderivative of L is actually a singleton set of its derivative.

Consider the function $\hat{L}(v) = f(u^*, v)$. Since f is differentiable, $\frac{\partial f}{\partial v}(u^*, v_0)$ is the derivative of $\hat{L}$ at point v_0. Also $\hat{L}(v_0) = L(v_0)$. Thus, by Proposition 2 of (Neu and Szepesvári, 2012), $\frac{\partial f}{\partial v}(u^*, v_0)$ also belongs to the subderivative of L. Therefore,

$$\frac{\partial L}{\partial v}(v_0) = \frac{\partial f}{\partial v}(u^*, v_0).$$

□

The differentiability of f in Proposition 3.1 usually holds. The uniformly Lipschitz-continuous of neural networks was also discussed in the paper of Szegedy et al. (2013). It still remains to compute u^* in Proposition 3.1. In particular given (x_i, y_i),

$$\max_{\|r^{(i)}\| \leq c} \ell(g(x_i + r^{(i)}), y_i). \tag{3}$$

We postpone the solution for the above problem to the end of this section. Given that we can have an approximate solution for Equation (3), a simple SGD method to compute a local solution for Equation (2) is shown in Algorithm 2.

Algorithm 2 Learning with Adversary

input (x_i, y_i) for $1 \le i \le N$; Initial f_0;
output $\hat{f}$
for $t = 1, 2, \ldots, T$ **do**
  for (x_i, y_i) in the current batch **do**
    Use forward-backward propagation to compute $\frac{\partial \alpha}{\partial x}$
    Compute r^* as the optimal perturbation to x, using the proposed methods in Section 3.1
    Create a pseudo-sample to be $(\hat{x}_i = x_i + c\frac{r^*}{\|r^*\|_2}, y_i)$
  end for
  Update the network $\hat{f}$ using forward-backward propagation on the pseudo-sample $(\hat{x}_i, y_i)$ for $1 \le i \le N$
end for
Return $\hat{f}$.

For complex data, deeper neural networks are usually proposed, which can be interpreted as consist of two parts: the lower layers of the networks learns a representation for the datapoints, while the upper layers learns a classifier. The number of layers that should be categorized as in the representation network is not clear and varies a lot for different datasets. Given such a general network, denote the representation network as $\mathcal{N}_{\text{rep}}$ and the classification network as $\mathcal{N}_{\text{cal}}$. We propose to perform the perturbation over the output of $\mathcal{N}_{\text{rep}}$ rather than the raw data. Thus the problem of learning with adversary can be fomulated as follows:

$$\min_{\mathcal{N}_{\text{rep}}, \mathcal{N}_{\text{cal}}} \sum_i \max_{\|r^{(i)}\| \le c} \ell\left(\mathcal{N}_{\text{cal}}\left(\mathcal{N}_{\text{rep}}(x_i) + r^{(i)}\right), y_i\right). \tag{4}$$

Similarly, Equation (4) can be solved by the following SGD method, as shown in Algorithm 3.

Algorithm 3 Learning with Adversary

input (x_i, y_i) for $1 \le i \le N$; Initial $\mathcal{N}_{\text{cal}}$ and $\mathcal{N}_{\text{rep}}$;
output $\hat{f}$
for $t = 1, 2, \ldots, T$ **do**
  for (x_i, y_i) in the current batch **do**
    Use forward propagation to compute the output of $\mathcal{N}_{\text{rep}}$, $\tilde{x}_i$
    Take $\tilde{x}_i$ as the input for $\mathcal{N}_{\text{cal}}$
    Use forward-backward propagation to compute $\frac{\partial \alpha}{\partial \tilde{x}_i}$
    Compute r^* as the optimal perturbation to $\tilde{x}_i$, using the proposed methods in Section 3.1
    Create a pseudo-sample to be $(\hat{\tilde{x}}_i = \tilde{x}_i + c\frac{r^*}{\|r^*\|}, y_i)$
  end for
  Use forward propagation to compute the output of $\mathcal{N}_{\text{cal}}$ on $(\hat{\tilde{x}}_i, y_i)$ for $1 \le i \le N$
  Use backward propagation to update both $\mathcal{N}_{\text{cal}}$ and $\mathcal{N}_{\text{rep}}$ for $1 \le i \le N$
end for
Return $\mathcal{N}_{\text{cal}}$ and $\mathcal{N}_{\text{rep}}$

3.1 Computing the perturbation

We propose two method based on two different principles. Our method, similar to that of (Goodfellow et al., 2014), does not require to solve an optimization problem. Experimental results show that our method, compared to the method proposed in (Goodfellow et al., 2014), is more efficient in that under the same magnitude of perturbation, the performance of the network is worse on our adversarial examples.

3.1.1 LIKELIHOOD BASED LOSS

Assume the loss function $\ell(x, y) = h(\alpha_y)$ where h is a non-negative decreasing function. One of the typical examples would be the logistic regression model. In fact, most of the network models use a softmax layer as the last layer and a cross-entropy objective function. All these networks can fit into this type of loss function. Recall that we would like to find

$$r^* = \arg \max_{\|r^{(i)}\| \leq c} h\left(g(x_i + r^{(i)})_{y_i}\right),$$

where x_i could be the raw data or the output of $\mathcal{N}_{\text{rep}}$. Since h is decreasing, $r^* = \arg\min_{\|r^{(i)}\| \leq c} g(x_i + r^{(i)})_{y_i}$.

This problem can be still difficult in general. We propose to compute a approximate solution based on the linear approximation of the function g. Replacing $g(x_i + r^{(i)})_{y_i}$ by its linear approximation $\tilde{g}(x_i + r^{(i)})_{y_i}$, i.e. $g(x_i + r^{(i)})_{y_i} \approxeq \tilde{g}(x_i + r^{(i)})_{y_i} = g(x_i)_{y_i} + < T_{y_i}, r^{(i)} >$, r^* can be solved for $\tilde{g}(x_i + r^{(i)})_{y_i}$ as $r^* = \{r : \|r\| \leq c; < T_{y_i}, r^{(i)} >= c\|T_{y_i}\|_*\}$.

The optimal r^* for common norms are :

1. If $\|\cdot\|$ is L_2 norm, then $r^*_{(j)} = c\,\frac{T_{y_i}}{\|T_{y_i}\|_2}$;
2. If $\|\cdot\|$ is L_∞ norm, then $r^*_{(j)} = c\,\text{sign}(T_{y_i})$;
3. If $\|\cdot\|$ is L_1 norm, then $r^*_{(j)} = c\,e_I$ where I satisfies $|T_{y_i}| = \|T_{y_i}\|_\infty$. Here V_I is the I-th element of V.

Note that the second item here is exactly the method suggested in (Goodfellow et al., 2014).

3.1.2 MISCLASSIFICATION BASED LOSS

In the case that the loss function ℓ is a surrogate loss for the misclassification rate, in Equation (3) it is reasonable to still use the misclassification rate as the loss function ℓ. Thus Equation (3) is to find a perturbation $r : \|r\| \leq c$ that make $\mathcal{N}$ misclassify x_i. In practice, in order for $\mathcal{N}$ to achieve good approximation, c is pick to be a small value, thus may not be large enough to force the misclassification of $\mathcal{N}$. One intuitive way is to have r the same direction as the one that is found in Section 2, since such direction is arguable to be an 'efficient' direction for the perturbation. Therefore,

$$r^* = c\,r_I^*/\|r_I^*\|,$$

where r_I^* is the output of Algorithm 1.

4 EXPERIMENTAL RESULTS

We use MNIST (LeCun et al., 1998b) and CIFAR-10 to test our methods of finding adversarial examples and training robust netowrks.

MNIST dataset contains grey scale handwrite images in size of 28x28. We random choose 50,000 images for training and 10,000 for testing. We normalize each pixel into range [0, 1] by dividing 256.

"Introduction to CIFAR-10": Data distribution

4.1 FINDING ADVERSARIAL EXAMPLES

We test different perturbation methods on MNIST including: 1. Perturbation based on α using ℓ_2 norm as shown in Section 2 (Adv_Alpha); 2. Perturbation based on Loss function using loss function using ℓ_2 norm as shown in Section 3.1 (Adv_Loss); 3. Perturbation based on Loss function using loss function using ℓ_∞ norm as shown in Section 3.1 (Adv_Loss_Sign);. In particular, a standard Lenet is trained on MNIST, with the training and validation accuracy being 100% and 99.1%. Based on the learned network, different validation sets are then generated by perturbing the original data with different perturbation methods. The magnitudes of the perturbations range from 0.0 to 4.0 in ℓ_2 norm. The classification accuracies on differently perturbed datasets are reported in Figure 1. As the

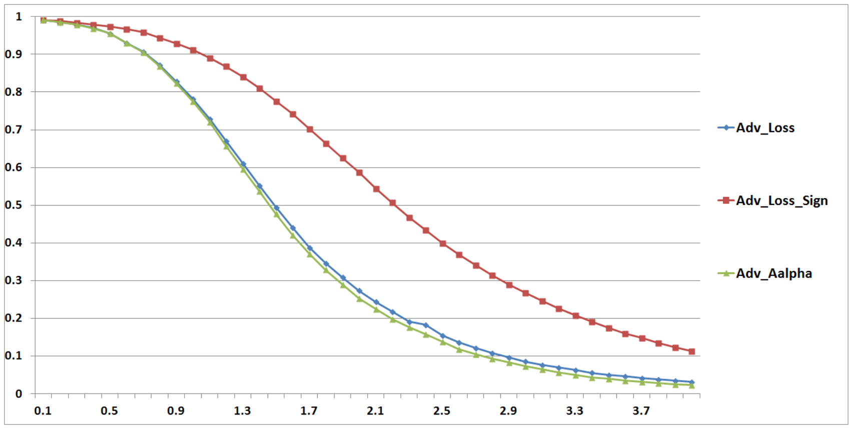

Figure 1: Validation accuracies of different perturbation methods. x axis denotes the ℓ_2 norm of the perturbation.

growing of the magnitude of the perturbation, the network's performance decreases. Experimental results suggest that Adv_Alpha is consistently, but slightly, more efficient than Adv_Loss, and these two method are significantly more efficient than Adv_Loss_Sign.

Is it reasonable to use ℓ_2 norm to measure the magnitude?
One may concern that if the following case may happen: a small perturbation in ℓ_2 norm has most of its weight on some specific position, and thus change the picture distinguishably. One of this example is shown as in Figure ******. However, the above example is artificially created, and we don't observe such phenomenon in our experiments.

Drawback of using α to find perturbations
Note that the difference in perturbation efficiency between using α and using the loss function is small. On the other hand, to compute the perturbation using α, one need to compute $\frac{\partial \alpha}{\partial x}$, which is actually d times computation complexity as that of the method using the loss function and thus only need to compute $\frac{\partial \ell}{\partial x}$. Here d is the number of classes.

4.2 Learning with adversary

We test our method on both MNIST and CIFAR-10.

Experiments on MNIST:
We first test different learning methods on a 2-hidden-layers neural network that has 1000 hidden nodes for each hidden layer: 1. Normal back-forward propagation training (Normal); 2. Normal back-forward propagation training with Dropout (Dropout); 3. The method in (Goodfellow et al., 2014) (Goodfellow's method); 4. Learning with adversary at raw data (LWA); 5. Learning with adversary at representation layer (LWA_Rep). The robustness of each classifier is measured on various adversarial sets. A same type of adversarial set for different learned classifiers are generated based on its targeted classifier. We generate 3 types of adversarial datasets for the above 5 classifiers corresponding to Adv_Alpha, Adv_Loss, and Adv_Loss_Sign. Moreover, we also evaluate the performances of these 5 classifiers on a fixed adversarial set which is generated based on the 'Normal' network using Adv_Loss. Lastly, we also report the original validation accuracies of different networks. All the results are tested under perturbations of ℓ_2 norm being 1.5.

The normal method can not afford any perturbation on the validation set, showing that it is not robust at all. By training with the dropout technique, both performance and robustness of the neural network are improved, but its robustness is still weak. Especially, for the adversarial set that is generated by Adv_Loss, its classification accuracy is only 13.5%. Goodfellow's method improves

Table 1: Classification accuracies for 2-hidden-layers neural network: the best performance on each adversarial sets are shown in bold. Note that for the LWA_Rep, we use 1000 hidden units for each hidden layer. The magnitude of perturbations are 1.5 in ℓ_2 norm.

METHODS	Validation Sets				
	Validation	Fixed	Adv_Loss_Sign	Adv_Loss	Adv_Alpha
Normal	0.980	0.314	0.193	0.091	0.078
Dropout	0.982	0.366	0.256	0.135	0.238
Goodfellow's Method	**0.987**	0.938	0.923	0.713	0.701
LWA	**0.987**	**0.962**	**0.935**	**0.860**	**0.857**
LWA_Rep*	0.986	0.899	0.900	0.802	0.767

the network's robust greatly, compared to the previous methods. The best performance and the most robustness are both achieved by LWA. In particular, on the adversarial set that is generated by our methods (Adv_Loss and Adv_Alpha), the performance is improved from 71.3% to 86.0%, and from 70.1% to 85.7%. The result of LWA_Rep is also reported for comparison. Overall, it achieves fair comparable performance to Goodfellow's method (Goodfellow et al., 2014).

We also evaluate these learning methods on LeNet (LeCun et al., 1998a), which is more complicated including convolution layers. Its learning curve is reported in Figure 2. It is interesting that we do not observe the trade-off between robustness and its performance. This phenomenon also happens to the 2-hidden-layers neural network. The final result is summarized in Table 2, which shows its

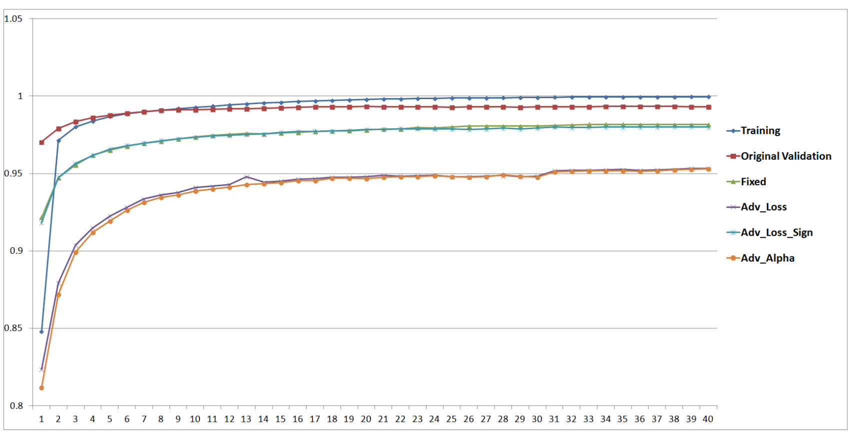

Figure 2: Validation accuracies of different perturbation methods. x axis denotes the ℓ_2 norm of the perturbation.

great robustness. Recall that from the results in

Table 2: Classification accuracies for LeNet trained using LWA. The magnitude of perturbations are 1.5 in ℓ_2 norm.

METHODS	Validation Sets				
	Validation	Fixed	Adv_Loss_Sign	Adv_Loss	Adv_Alpha
Normal	0.9916	0.6342	0.6940	0.3998	0.3792
LWA	0.9932	0.9817	0.9800	0.9533	0.9529

Experiments on CIFAR-10:

STAY TUNED!!!

ACKNOWLEDGMENTS

We thank Naiyan Wang and Ian Goodfellow for meaningful discussions. This work was supported by the Alberta Innovates Technology Futures and NSERC.

REFERENCES

Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *arXiv preprint arXiv:1502.02590*, 2015.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998b.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing by virtual adversarial examples. *arXiv preprint arXiv:1507.00677*, 2015.

Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. *arXiv preprint arXiv:1206.5264*, 2012.

Arild Nøkland. Improving back-propagation by adding an adversarial gradient. *arXiv preprint arXiv:1510.04189*, 2015.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Pedro Tabacof and Eduardo Valle. Exploring the space of adversarial images. *arXiv preprint arXiv:1510.05328*, 2015.