A New Family of Neural Networks Provably Resistant to Adversarial Attacks

Rakshit Agrawal   Luca de Alfaro   David Helmbold*

Computer Science and Engineering Department
University of California, Santa Cruz
ragrawa1@ucsc.edu, luca@ucsc.edu, dph@ucsc.edu

January 23, 2019
Technical Report UCSC-SOE-19-01
School of Engineering, University of California, Santa Cruz

Abstract

Adversarial attacks add perturbations to the input features with the intent of changing the classification produced by a machine learning system. Small perturbations can yield adversarial examples which are misclassified despite being virtually indistinguishable from the unperturbed input. Classifiers trained with standard neural network techniques are highly susceptible to adversarial examples, allowing an adversary to create misclassifications of their choice.

We provide two main contributions advancing the state of the art in protecting against adversarial attacks. First, we introduce a new type of network unit, called MWD (max of weighed distance) units. These units have a specially designed non-linear structure that gives them built-in resistant to adversarial attacks, and we develop the techniques needed to effectively training them. Second, we introduce a method for propagating perturbation effects in a network that enables the efficient computation of robustness (i.e., accuracy guarantees) under any perturbations, including adversarial attacks.

MWD networks are significantly more robust to input perturbations than ReLU networks. On permutation invariant MNIST, when test examples can be perturbed by 20% of the input range, MWD networks provably retain accuracy above 83%, while the accuracy of ReLU networks drops below 5%. The provable accuracy of MWD networks is superior even to the observed accuracy of ReLU networks trained with the help of adversarial examples. In the absence of adversarial attacks, MWD networks match the performance of sigmoid networks, and have accuracy only slightly below that of ReLU networks.

*The authors are listed in alphabetical order.

1 Introduction

Machine learning via deep neural networks has been remarkably successful in a wide range of applications, including speech recognition, image classification, and language processing. While very successful, deep neural networks are susceptible to adversarial examples: small, carefully crafted, perturbations of inputs can change their predicted classification [SZS$^+$13]. These perturbations can often be so small as to make human detection difficult or impossible; this has been shown both in the case of images [SZS$^+$13, NYC15] and sounds [KGB16a, CW18]. Further, the adversarial examples are in some sense transferable from one neural network to another ([GSS14, NYC15, PMJ$^+$16, TPG$^+$17]), so they can be crafted even without precise knowledge of the targeted network's parameters. At a fundamental level, it is hard to be confident about the behavior of a deep neural network when most correctly classified inputs are in close proximity to very similar, yet differently classified inputs.

The dominant approach for increasing a neural network's resistance to adversarial attacks is to augment the training data with adversarial examples [GSS14, MMS$^+$17]. If the added adversarial examples are generated by efficient heuristics such as the *fast gradient sign method,* the networks learn to associate the specific adversarial examples with the corresponding unperturbed input in a phenomenon known as *label leaking* [KGB16b, MMS$^+$17, TKP$^+$17]. This does not result in increased resistance to more general adversarial attacks [MMS$^+$17, CW17b]. Networks with greater resistance to adversarial attacks can be obtained if the adversarial examples used in training are generated via more general optimization techniques, as in [MMS$^+$17]. This comes at the cost of a more complex and computationally expensive training regime, as well as an increase in network's capacity.

Here we propose a different approach, the use of neural network units that are inherently resistant to adversarial attacks, even when trained using only unperturbed input. [GSS14] connect the presence of adversarial examples to the (local) linearity of neural networks. Consider the linear form $\sum_{i=1}^n x_i w_i$ and perturbing each x_i by ϵ, taking $x_i + \epsilon$ if $w_i > 0$, and $x_i - \epsilon$ if $w_i < 0$. The output is then perturbed by $\epsilon \sum_{i=1}^n |w_i|$, or $\epsilon n \bar{w}$ where $\bar{w}$ the average magnitude of the w_i's. When the number of inputs n is large, as is typical of deep neural networks, a small input perturbation can cause a large output change, and this change can snowball through the layers. Of course, deep neural networks are not globally linear, but the insight of [GSS14] is that they may be sufficiently locally linear to promote the success of adversarial attacks. Following this insight, we develop networks composed of units that are highly non-linear.

After much experimentation, we found a promising node type that we call *Max-of-Weighted-Distance* (MWD) units. Like Gaussian radial basis functions [BL88a, CCG91, Orr96], MWD units activate based on the distance from their input vectors to a learned center. However, MWD units also learn a non-negative weight for each input component and measure distance from the center with the infinity-norm rather than the Euclidean norm. The component weighing can give a high sensitivity to some components while ignoring others, allowing a single MWD unit to cover a more flexible region of the input space. The use of the infinity norm reduces sensitivity to adversarial perturbations because any change in the output is due to the perturbation of one input component, rather than the sum of the perturbations to all of the input components. The output of a MWD unit $\mathcal{U}$ with parameters $\boldsymbol{u}$ and $\boldsymbol{w}$ on input $\boldsymbol{x}$ is

$$\mathcal{U}(\boldsymbol{u}, \boldsymbol{w})(\boldsymbol{x}) = \exp\Big(-\max_{1 \le i \le n} \big(u_i(x_i - w_i)\big)^2\Big) . \tag{1}$$

Using highly nonlinear models is hardly a new idea, but the challenge has been that such models are typically difficult to train. Indeed, we found that networks with MWD units cannot be satisfactorily trained using gradient descent. To get around this, we show that the networks can be trained efficiently, and to high accuracy, using *pseudogradient descent* where the *pseudogradient* is a proxy for the gradient that facilitates training. The maximum operator in (1) has non-zero derivative only for the maximizing input; our pseudogradient propagates a derivative signal back to all of the inputs. Also, the exponential function in (1) is very flat far away from the center, so our pseudogradient artificially widens the region of meaningful gradients. Tampering with the gradient may seem unorthodox, but methods such as AdaDelta ([Zei12]), and even gradient descent with momentum, cause training to take a trajectory that does not follow pure gradient descent. We simply go one step further, devising a scheme that operates at the granularity of the individual unit.

Our second contribution consists in a technique that yields *lower-bound guarantees* on the accuracy of networks subjected to adversarial attacks. Thus far, the effect of adversarial attacks on networks has been studied chiefly by exposing the networks to attack, and measuring the resulting performance. This yields accuracy *upper bounds:* one can measure the accuracy degradation under specific attacks, but one cannot obtain guarantees for *all* attacks, as the existence of more powerful attacks cannot be ruled out. Indeed, frequently networks were thought to be resistant, only for more powerful adversarial attacks to be discovered later [CW16, CW17a].

Specifically, our technique enables to verify whether all perturbations of a given input of size less than a specified $\epsilon > 0$ cause no misclassification. Thus, the technique can be used to prove assertions such as "For any input perturbations of size at most x in infinity norm, the network yields accuracy of at least y% on the testing set". The guarantees *are* dependent on the testing set, and in general, on the probability distribution over the inputs, but this is intrinsic to any guarantee or performance measure in machine learning. If general input distributions were allowed, obviously one could feed to the network exclusively inputs that are known to be misclassified and obtain accuracy 0. The technique is based on the forward propagation of *perturbation intervals* from the input to the output. We show that the computation of tight perturbation intervals is in general NP-complete, and we provide an approximate technique, of efficiency similar to simple forward network propagation, for propagating perturbation intervals and computing accuracy guarantees under attacks.

To conduct our experiments, we have implemented MWD networks on top of the PyTorch framework [PGC+17]. The code will be made available with the final version of the paper. We consider *permutation invariant MNIST,* which is a version of MNIST in which the 28×28 pixel images are flattened into a one-dimensional vector of 784 values and fed as a feature vector to neural networks [GSS14]. On this test set, we show that for nets of 512,512,512,10 units, MWD networks match the classification accuracy of sigmoid networks ($(96.96 \pm 0.14)\%$ for MWD vs. $(96.88 \pm 0.15)\%$ for sigmoid), and are close to the performance of network with ReLU units ($(98.62 \pm 0.08)\%$). When trained over standard training sets, for input perturbations of 20% of the input range, MWD networks guarantee an accuracy above 80% for any adversarial attack, while there are simple attacks that reduce the accuracy of ReLU and Sigmoid networks to below 10% (random guessing). Even when ReLU networks are trained with the benefit of adversarial attacks, for the most relevant range of input perturbations (from 5% to 25% of the input range), the accuracy lower bound guarantees offered by (normally trained) MWD networks exceed the upper bounds of ReLU networks due to known attacks.

Our results can be summarized as follows:

- We define a class of networks, MWD networks, inherently resistant to adversarial attacks, and we develop a pseudogradient-based way for effectively training them.
- We introduce an efficient technique for obtaining accuracy lower bounds (guarantees) in presence of *any* adversarial attack.
- On MNIST, we show that in absence of adversarial attacks, MWD networks match the accuracy of sigmoid networks, and have only slightly lower accuracy than ReLU networks.
- Again on MNIST, in presence of adversarial attacks, we show that the accuracy lower-bound guarantees of MWD networks far exceed the accuracy upper bounds of ReLU and Sigmoid networks. We show that the accuracy lower bounds of MWD networks are above the ReLU upper bounds even when the latter are trained with the help of adversarial examples.

Much work remains to be done, including experimenting with MWD units in convolutional networks. However, these initial results offer a practical method for training networks that are provably — and significantly – resistant to adversarial attacks.

2 Related Work

The vulnerability of neural networks to adversarial examples was first reported by [SZS+13], and they showed how to generate them via a simple optimization. [GSS14] established a connection between linearity and adversarial attacks. A fully linear form $\sum_{i=1}^{n} x_i w_i$ can be perturbed to $x_i + \epsilon \, \mathrm{sign}(w_i)$, creating an output change of magnitude $\epsilon \cdot \sum_{i=1}^{n} |w_i|$. In analogy, [GSS14] introduced the *fast gradient sign method* (FGSM) method of creating adversarial perturbations, by taking $x_i + \epsilon \cdot \mathrm{sign}(\nabla_i \mathcal{L})$, where $\nabla_i \mathcal{L}$ is the loss gradient with respect to input i. They also showed how adversarial examples are often transferable across networks, and asked if non-linear structures, perhaps like those of RBFs, would be more robust to adversarial attacks. This paper pursues this approach and provides positive answers to the conjectures and suggestions by [GSS14].

It was recently discovered that training on adversarial examples generated via FGSM does not confer strong resistance to attacks, as the network learns to associate the specific examples generated by FGSM with the original training examples in a phenomenon known as *label leaking* [KGB16b, MMS$^+$17, TKP$^+$17]. The FGSM method for generating adversarial examples was extended to an iterative method, I-FGSM, in [KGB16a]. In [TKP$^+$17], it is shown that using small random perturbations before applying FSGM enhances the robustness of the resulting network. The network trained in [TKP$^+$17] using I-FSGM and ensemble method won the first round of the NIPS 2017 competition on defenses with respect to adversarial attacks.

Carlini and Wagner show that training regimes based on generating adversarial examples via simple heuristics, or combinations of these, in general fail to convey true resistance to attacks [CW17a, CW17b]. They further advocate measuring the resistance to attacks with respect to adversarial examples created by more general optimization processes. In particular, FGSM and I-FGSM rely on the local gradient, and training techniques that break the association between the local gradient and the location of adversarial examples makes networks harder to attack via FGSM and I-FGSM, without making the networks harder to attack via general optimization techniques. We follow this suggestion by using a general optimization method, projected gradient descent (PGD), to generate adversarial attacks and evaluate network robustness. [CW16, CW17b] also show that the technique of *defensive distillation,* which consists in appropriately training a neural network on the output of another [PMW$^+$16], protects the networks from FGSM and I-FGSM attacks, but does not improve network resistance in the face of general adversarial attacks.

[MMS$^+$17] show that it is possible to obtain networks that are genuinely more resistant to adversarial examples by training on adversarial examples generated via PGD. The price to pay is a more computationally intensive training, and an increase in the network capacity required. We provide an alternative way of achieving such resistance that does not rely on a new training regime.

Finally, [PRGS17] develop a technique based on differential analysis for deriving lower bounds to resistance to adversarial attacks. The tecnhnique does not provide bound guarantees for networks including non-differentiable units, such as ReLU and MWD networks. Our technique based on the propagation of perturbation intervals offers such guarantees.

3 MWD Networks

[GSS14] link adversarial attacks to the linearity of the models. In a linear form $g(\boldsymbol{x}) = \sum_i x_i w_i$, if we perturb x_i by adding ϵ when $w_i > 0$, and subtracting it when $w_i < 0$, the perturbations on the various inputs add up, so that small input perturbations can yield large output changes. To achieve resistance to adversarial attacks, we seek units where the contributions of the inputs are not added up. The linear form represents the norm-2 distance of the input vector x to a hyperplane perpendicular to vector $\boldsymbol{w}$, scaled by $|\boldsymbol{w}|$ and its orientation. Our units will be based instead on infinity-norm distances.

We define our units as variants of the classical Gaussian *radial basis functions* [BL88b, Orr96]. We call our variant *Max-of-Weighted-Distance* (MWD), to emphasize the fact that they are built using infinity norm. An MWD unit $\mathcal{U}(\boldsymbol{u}, \boldsymbol{w})$ for an input in $\mathbb{R}^n$ is parameterized by two vectors of weights $\boldsymbol{u} = \langle u_1, \ldots, u_n \rangle$ and $\boldsymbol{w} = \langle w_1, \ldots, w_n \rangle$ Given an input $\boldsymbol{x} \in \mathbb{R}^n$, the unit produces output

$$\mathcal{U}(\boldsymbol{u}, \boldsymbol{w})(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{u} \odot (\boldsymbol{x} - \boldsymbol{w})\|_\infty^2\right) \ , \tag{2}$$

where $\odot$ is the Hadamard, or element-wise, product. In (2), the vector $\boldsymbol{w}$ is a point from which the distance to $\boldsymbol{x}$ is measured in infinity norm, and the vector $\boldsymbol{u}$ provides independent scaling factors for each coordinate. Without loss of expressiveness, we require the scaling factors to be non-negative, that is, $u_i \geq 0$ for all $1 \leq i \leq n$. The scaling factors provide the flexibility of disregarding some inputs x_i, by having $u_i \approx 0$, while emphasizing the influence of other inputs. Expanding (2) gives: $\mathcal{U}(\boldsymbol{u}, \boldsymbol{w})(\boldsymbol{x}) = \exp\Big(-\max_{1 \leq i \leq n}\big(u_i(x_i - w_i)\big)^2\Big)$ as in equation (1).

The output of MWD units are in $(0, 1]$ and are close to 1 only when $\boldsymbol{x}$ is close to $\boldsymbol{w}$ in the coordinates that have significant scaling factors. Thus, the unit functions somewhat like an AND gate, outputting 1 only when the relevant inputs take on a particular set of values. We also consider the negated MWD unit which functions somewhat like a NAND gate: $\mathcal{U}^{\text{NEG}}(\boldsymbol{u}, \boldsymbol{w}) = 1 - \mathcal{U}(\boldsymbol{u}, \boldsymbol{w})$. We construct neural networks out of MWD units using layers consisting of $\mathcal{U}$ units, layers consisting of $\mathcal{U}^{\text{NEG}}$ units, and mixed layers where the unit type is chosen at random at network initialization.

3.1 Training MWD Networks via Pseudogradients

The non-linearities in (1) make neural networks containing MWD units difficult to train using standard gradient descent. The problems are associated with the max-operator making the gradients sparse and the fast decay of Gaussian functions. Far from its peak for $\boldsymbol{x} = \boldsymbol{w}$, a function of the form (1) is rather flat, and its derivative may not be large enough top cause the vector of weights $\boldsymbol{w}$ to move towards useful places in the input space during training. To obtain networks that are easy to train, we replace the derivatives for exp and max with alternate functions, which we call *pseudoderivatives.* These *pseudoderivatives* are then used in the chain-rule computation of the loss gradient in lieu of the true derivatives, yielding a *pseudogradient.*

Exponential function. In computing the partial derivatives of (2) via the chain rule, the first step consists in computing $\frac{d}{dz}e^{-z} = -e^{-z}$. The problem is that $-e^{-z}$ is very close to 0 when z is large, and z in (1) is $\|\boldsymbol{u} \odot (\boldsymbol{x} - \boldsymbol{w})\|_\infty^2$, which can be large. Hence, in the chain-rule computation of the gradient, we replace $-e^{-z}$ with the "pseudoderivative" $-1/\sqrt{1+z}$, which decays much more slowly.

Max. The gradient of $y = \max_{1 \le i \le n} z_i$, of course, is given by $\frac{\partial y}{\partial z_i} = 1$ if $z_i = y$, and $\frac{\partial y}{\partial z_i} = 0$ if $z_i < y$. The problem is that this transmits feedback only to the largest input(s). This slows down training and can create instabilities. We use as pseudoderivative $e^{z_i - y}$, so that some gradient feedback is transmitted to the other inputs z_i based on their closeness to y.

One may be concerned that by using the loss pseudogradient as the basis of optimization, rather than the true loss gradient, we may converge to solutions where the pseudogradient is null, and yet, we are not at a minimum of the loss function. This can indeed happen. We experimented with switching to training with true gradients after the accuracy reached via pseudogradients plateaued; this increased the accuracy on the training set, but improved only slightly the accuracy on the testing set. We also experimented with pseudogradients parameterized by a parameter $\rho \in [0, 1]$, such that when $\rho = 0$ the pseudogradients coincide with the true gradients. During training the parameter ρ starts at 1 and then, as training proceeds, gradually decays to 0, thus allowing a smooth transition from a pseudogradient training regime to a standard gradient one. At least for MNIST, these more sophisticated schemes failed to significantly improve on the simple use of the pseudogradients above.

4 Accuracy Lower Bounds For Adversarial Attacks

For an arbitrary network $\mathcal{N}$, and input $\boldsymbol{x}$, we provide a technique for estimating the maximum output perturbation that can be obtained by perturbing the input by at most ϵ. Our technique calculates an input-dependent sensitivity by computing the range of values that can be produced at each node under a given amount of input perturbation, and is efficient for a wide variety of node types, including sigmoid, ReLU, and MWD.

Definitions. The *ϵ-perturbation* of an input $\boldsymbol{x}$ is any $\boldsymbol{x}'$ where $||\boldsymbol{x} - \boldsymbol{x}'||_\infty \le \epsilon$ and the input belongs to the input domain. Given a classifier, an *ϵ-adversarial example* for input $\boldsymbol{x}$ is an ϵ-perturbation of $\boldsymbol{x}$ that results in a different prediction. The *true ϵ-attack accuracy* of a trained network is the fraction of test examples for which the network predicts correctly and no ϵ-adversarial examples exist.

Since adversarial examples are often hard to find, this true attack accuracy is difficult to measure. Given a particular heuristic for producing ϵ-attacks, we can measure the fraction of examples in the testing set that are predicted correctly, and for which the heuristics is unable to find an ϵ-adversarial example. Each heuristic (or attack strategy) yields an upper bound on the true ϵ-attack accuracy of the network. Unfortunately, determining whether there exists an ϵ-perturbation that causes a misclassification (and thus, determining the true ϵ-attack accuracy) is NP-complete, as the following theorem shows. The theorem is a somewhat unsurprising consequence of the relationship between neural networks and boolean circuits.

Theorem 1 *Given a MWD network, input $\boldsymbol{x}$ and output $\boldsymbol{y}$, deciding whether there is an ϵ-perturbation of $\boldsymbol{x}$ that yields output $\boldsymbol{y}$ is NP-complete.*

Proof (sketch). The existence of polynomial-time witnesses shows that the problem is in NP. The proof consists of a reduction from 3-SAT [GJ79]. Consider a set of predicate letters $P = \{p_1, \ldots, p_n\}$, their complements $\bar{P} = \{\bar{p}_1, \ldots, \bar{p}_n\}$, and a set $C = \{c_1, \ldots, c_m\}$ of clauses, with $c_j \subseteq P \cup \bar{P}$ for $1 \leq j \leq m$.

Given an instance C of 3-SAT, we construct a network with two layers. We can show that the problem is NP-hard already for $\boldsymbol{x} = \langle 0.5, \ldots, 0.5\rangle$ and $\epsilon = 0.5$ (the theorem can be strenthened to hold for specific ϵ-values). In the first layer, there is one unit $N_j^{(1)}$ for each clause c_j. If $p_i \in c_j$, then we set $w_{ji}^{(1)} = 0$ and $u_{ji}^{(1)} = 1$; if $\bar{p}_i \in c_j$, we set $w_{ji}^{(1)} = 1$ and $u_{ji}^{(1)} = 1$; we set $w_{ji}^{(1)} = u_{ji}^{(1)} = 0$ otherwise. If the network inputs can be only either 0 or 1, the output z_j of $N_j^{(1)}$ can reach $1/e$ only if c_j is satisfied. In the second layer, there is only one unit, with one input for each clause. For this unit, we set $w_j^{(2)} = 1/e$ and $u_j^{(2)} = 1$ for each input j (from $N^{(j)}$); the output y of this layer can reach 1 only if every z_i is $1/e$.

It is not difficult to see that the network yields output 1 for an 0.5-perturbation of the input $\boldsymbol{x} = \langle 0.5, 0.5, \ldots, 0.5\rangle$ iff the 3-SAT instance is satisfiable. ∎

In view of this hardness result, we present an approximate technique which propagates *perturbation intervals* forward through the network. These over-approximate the actual range of values that the outputs of the nodes can attain when a particular input example is subject to perturbations. Given a network, an input $\boldsymbol{x}$, and an $\epsilon \geq 0$, the *ϵ-perturbation interval* $I_j^\epsilon(\boldsymbol{x})$ for each node j in the network is defined recursively as follows:

- If j is an input node with input value x, then $I_j^\epsilon(\boldsymbol{x}) = [x - \epsilon, x + \epsilon]$ (intersected with any input constraints).
- Otherwise, consider a node q computing a function $f(\cdot)$ of its n inputs, and let $I_1^\epsilon(\boldsymbol{x}), \ldots, I_n^\epsilon(\boldsymbol{x})$ be the perturbation intervals of its inputs. Let $\mathcal{I}^\epsilon(\boldsymbol{x}) = \prod_{1 \leq j \leq n} I_i^\epsilon(\boldsymbol{x})$ be the Cartesian product of the input intervals to the node, the perturbation interval of the node is then simply
  $I_q^\epsilon(\boldsymbol{x}) = \left[\min_{\boldsymbol{z} \in \mathcal{I}^\epsilon(\boldsymbol{x})} f(\boldsymbol{z}), \max_{\boldsymbol{z} \in \mathcal{I}^\epsilon(\boldsymbol{x})} f(\boldsymbol{z})\right]$.

The following theorem, provable by a simple induction on the network layers, states that the output of a network subject to input perturbations is always contained in the perturbation interval of its output.

Theorem 2 *If the network input $\boldsymbol{x}'$ is an ϵ-perturbation of $\boldsymbol{x}$, then every node j outputs values in its perturbation interval $I_j^\epsilon(\boldsymbol{x})$.*

The theorem, via the following corollary, yields a lower bound on a network's true ϵ-attack accuracy.

Corollary 1 *If the network predicts correctly on some test example $\boldsymbol{x}$ and every combination of values within the output nodes' ϵ-perturbation intervals lead to the same prediction, then $\boldsymbol{x}$ has no ϵ-adversary examples so $\boldsymbol{x}$ can be counted towards the network's true ϵ-attack accuracy.*

In particular, consider a classification problem with n classes, and a classifier network f with n outputs, $y_1, \dots, y_n$. Input $\boldsymbol{x}$ (belonging to, say, class 1) is classified correctly by $\boldsymbol{y} = f(\boldsymbol{x})$ if $y_1 > y_k$ for $k = 2, \dots, n$. To rule out the existence of ϵ-attacks for $\boldsymbol{x}$, it suffices to check that $l_1 > r_k$ for $k = 2, \dots, n$, where $[l_k, r_k] = \mathcal{I}_k^\epsilon(\boldsymbol{x})$ is the ϵ-perturbation interval for output $k = 1, \dots, n$ and input vector $\boldsymbol{x}$.

For many node types the perturbation intervals are easy to compute. In particular, for an MWD node, let $[l_1, r_1], \dots, [l_n, r_n]$ be the perturbation intervals of its n inputs for a given $\boldsymbol{x}$, and denote by $f(\cdot)$ the node function, defined as by (2). Its output perturbation interval $[l, r]$ for $\boldsymbol{x}$ can be computed via $l = \max_{1 \le i \le n} \min\big\{f(l_i), f(r_i)\big\}$, and $r = \max_{1 \le i \le n} m_i$, where

$$m_i = \begin{cases} 1 & \text{if } l_i \le w_i \le r_i \\ \max\big\{f(l_i), f(r_i)\big\} & \text{otherwise.} \end{cases}$$

Computing these input-dependent perturbation intervals takes effort comparable to (perhaps $2\times$ or $3\times$) that of calculating the network's output. Utilizing symbolic calculations, one could also calculate perturbation intervals as a function of the perturbation size ϵ.

5 Generating Adversarial Examples

We describe the known methods for generating candidate adversarial examples that we will use in the experiments. Consider a network trained with cost function $J(\theta, \boldsymbol{x}, \boldsymbol{y})$, where θ are the network parameters, $\boldsymbol{x}$ is the input, and $\boldsymbol{y}$ is the output. Let $\nabla_{\boldsymbol{x}} J(\theta, \boldsymbol{x}', \boldsymbol{y})$ be the gradient of J wrt its second argument (the input) computed at $\theta, \boldsymbol{x}', \boldsymbol{y}$. For a perturbation amount $\epsilon > 0$ and an input $\boldsymbol{x}$ belonging to the test set, we produce candidate adversarial examples $\tilde{\boldsymbol{x}}$ with $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_\infty \le \epsilon$ using the following techniques.

Fast Gradient Sign Method (FGSM) [GSS14]. In the FGSM technique, the candidate adversarial example is generated via:

$$\tilde{\boldsymbol{x}} = [\![\boldsymbol{x} + \epsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} J(\theta, \boldsymbol{x}, \boldsymbol{y}))]\!]_0^1 \ , \tag{3}$$

where $[\![\boldsymbol{x}]\!]_a^b$ is the result of clamping each component of $\boldsymbol{x}$ to the range $[a, b]$; the clamping is necessary to generate a valid MNIST image.

Iterated Fast Gradient Sign Method (I-FGSM) [KGB16a]. The I-FGSM attack computes a sequence $\tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_M$, where $\tilde{\boldsymbol{x}}_0 = \boldsymbol{x}$, and, for $0 \leq i < M$:

$$\tilde{\boldsymbol{x}}_{i+1} = \left[\!\!\left[\tilde{\boldsymbol{x}}_i + \frac{\epsilon}{M} \operatorname{sign}(\nabla_{\boldsymbol{x}} J(\theta, \tilde{\boldsymbol{x}}_i, \boldsymbol{y}))\right]\!\!\right]_0^1 . \tag{4}$$

We take $\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}_M$ as the candidate adversarial example.

Projected Gradient Descent (PGD) [MMS$^+$17]. For an input $\boldsymbol{x} \in \mathbb{R}^n$ and a given maximum perturbation size $\epsilon > 0$, we consider the set $B_\epsilon(\boldsymbol{x}) \cap [0, 1]^n$ of valid perturbations of $\boldsymbol{x}$, and we perform projected gradient descent (PGD) in $B_\epsilon(\boldsymbol{x}) \cap [0, 1]^n$ of the negative loss with which the network has been trained (or, equivalently, projected gradient ascent wrt. the loss). We perform this search with multiple restarts, each chosen uniformly at random from $B_\epsilon(\boldsymbol{x}) \cap [0, 1]^n$.

6 Experimental Setup

We implemented MWD networks in the PyTorch framework [PGC$^+$17]. To implement pseudogradients, we extend PyTorch with two new functions: a *LargeAttractorExp* function, with forward behavior e^{-x} and backward gradient propagation according to $-1/\sqrt{1+x}$, and *SharedFeedbackMax,* with forward behavior $y = \max_{i=1}^n x_i$ and backward gradient propagation according to $e^{x_i - y}$. These two functions are used in the definition of MWD units, as per (1), with the AutoGrad mechanism of PyTorch providing backward (pseudo)gradient propagation for the complete networks.

Dataset. We use the MNIST dataset [LBBH98] for our experiments, following the standard setup of 60,000 training examples and 10,000 testing examples. Each digit image was flattened to a one-dimensional feature vector of length $28 \times 28 = 784$, and fed to a fully-connected neural network; this is the so-called *permutation-invariant* MNIST.

Neural networks. We compared the accuracy of ReLU, sigmoid, and MWD networks. The output of ReLU networks [NH10] is fed into a softmax, and the network is trained via cross-entropy loss. Sigmoid and MWD networks are trained via square-error loss, which worked better than other losses in our experiments. We trained all networks with the AdaDelta optimizer [Zei12], which gave good results for all networks considered.

Network	Accuracy	FGSM, ϵ=0.3	I-FGSM, ϵ=0.3	PGD, ϵ=0.3
ReLU	$\mathbf{98.62 \pm 0.08}$	1.98 ± 0.42	0.06 ± 0.06	81.20
Sigmoid	96.88 ± 0.15	0.71 ± 0.43	0.11 ± 0.11	40.75
MWD	96.96 ± 0.14	$\mathbf{94.90 \pm 0.35}$	$\mathbf{93.27 \pm 0.48}$	**92.30**
MWD[psd]	96.96 ± 0.14	$\mathbf{85.88 \pm 2.02}$	$\mathbf{78.92 \pm 1.91}$	**90.70**

Table 1: Performance of 512-512-512-10 networks for MNIST testing input, and for input corrupted by adversarial attacks computed with perturbation size $\epsilon = 0.3$.

Attacks. We applied FGSM and I-FGSM attacks to the whole test set. In I-FGSM attacks, we performed 10 iterations of (4). As PGD attacks are computationally intensive, we apply them to one run only, and we compute the performance under PGD attacks for the first 2,000 inputs in the test set for ReLU and Sigmoid networks, and for 1,000 inputs for MWD networks. For each input $\boldsymbol{x}$ in the test set, we perform 100 PGD searches, or restarts. Each search starts from a random point in $B_\epsilon(\boldsymbol{x})$ and then does 100 steps of projected gradient descent using the AdaDelta algorithm to tune step size; if at any step a misclassified example is generated, the attack is considered successful.

For MWD networks, we can perform FGSM, I-FGSM, and PGD attacks based either on true gradient, or on the pseudogradients. We denote the pseudogradient-based attacks with [psd] in the figures and tables. Such attacks in general are more powerful than attacks based on the regular gradient, for the same reasons why the pseudogradient is more effective in training.

7 Results

Unless otherwise noted, we report results on networks with layers of 512, 512, 512, and 10 units. For MWD networks, the layers consist of $\mathcal{U}$, $\mathcal{U}^{\text{NEG}}$, $\mathcal{U}$, $\mathcal{U}^{\text{NEG}}$ units respectively, and we use a bound of $[0.01, 3]$ for the components of the u-vectors, and of $[0, 1]$ for the w-vectors, the latter corresponding to the value range of MNIST pixels.

Accuracy. In Table 1 we summarize the accuracies of networks trained on the MNIST training set, as measured both on the (un-perturbed) test set, and upper bounds on the true ϵ=0.3-attack accuracy provided by the various adversarial attacks. The results are computed as the average of 10 training runs for ReLU and Sigmoid networks, and of 5 runs for MWD and MWD[psd]. In each run we used different random seeds for weight initialization; each run consisted of 30 training epochs. In a result of the form $a \pm e$, a is the percentage accuracy, and e is the standard deviation in the accuracy of the individual runs.

In absence of perturbations, MWD networks lose $(1.66 \pm 0.21)\%$ performance compared to ReLU networks (from $(98.62 \pm 0.07)\%$ to $(96.96 \pm 0.14)\%$), and perform

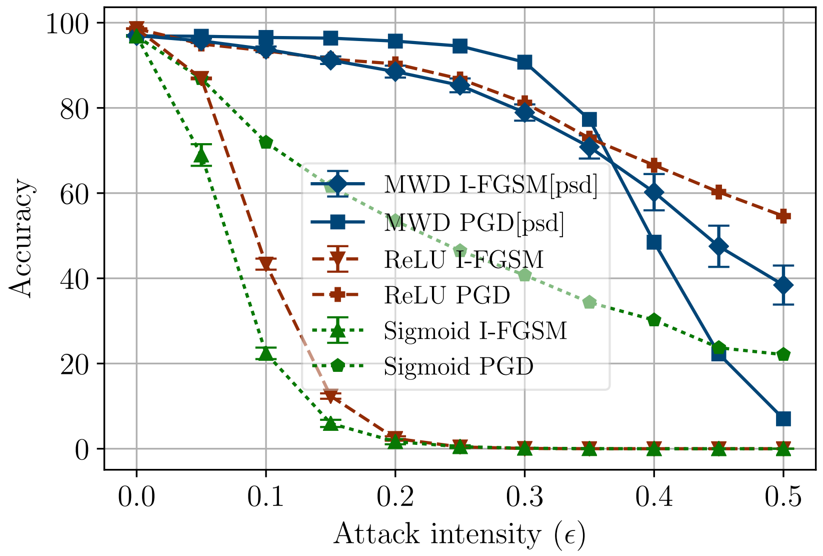

Figure 1: Performance of ReLU, Sigmoid, and MWD networks in presence of PGD and I-FGSM attacks.

comparably to sigmoid networks (the difference is below the standard deviation of the results). When heuristic perturbations are present, the performance of MWD networks is superior. I-FGSM attacks are usually the most effective.

In Figure 1 we compare the performance of the networks subjected to I-FGSM and PGD attacks for attack amplitudes up to $\epsilon = 0.5$. The error bars in this and subsequent graphs indicate the standard deviation across the runs, when available. For MWD networks, we plot only attacks based on pseudogradient, as they are more effective. We also omitted the results for FGSM, as it is a weaker attack than its iterated version I-FGSM. We see that for ReLU and Sigmoid networks, I-FGSM is the strongest attack, giving a rapidly-decaying upper bound on the networks' true ϵ-attack accuracy. For MWD networks, the relative strength of I-FGSM and PGD attacks depends on the intensity ϵ.

Comparing lower bounds for MWD with upper bounds for ReLU, Sigmoid. Figure 2 compares the lower bound on the true ϵ-attack accuracy of MWD networks with the best upper bounds on the true ϵ-attack accuracy from the various attacks on the MWD, Sigmoid, and ReLU networks. The lower bound for MWD networks is computed with the methods of Section 4, and holds for all possible adversarial attacks (or perturbations). The upper bound for MWD is derived as the minimum of the curves for PGD[psd] and I-FGSM[psd] in Figure 1. The upper bounds for ReLU and Sigmoid networks are derived from I-FGSM attacks, as they are (in our setting) strictly more powerful than PGD attacks for these net-

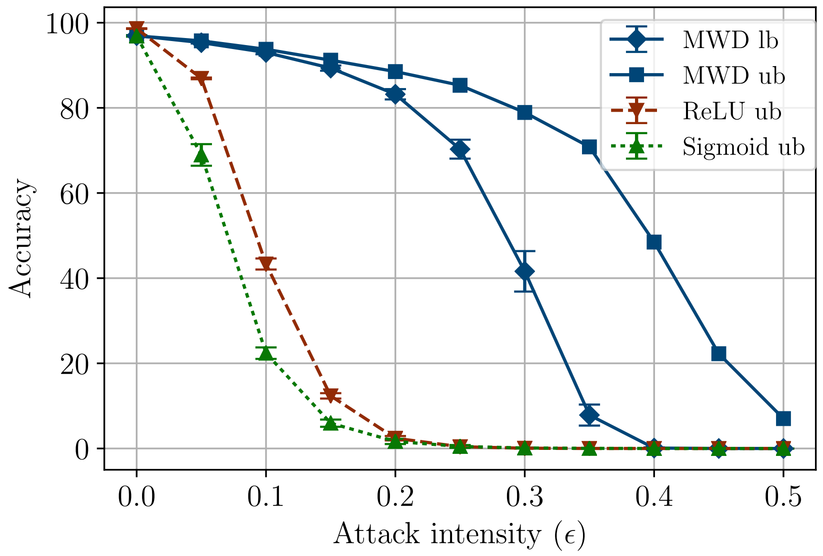

Figure 2: Upper and lower accuracy bounds for MWD vs. upper accuracy bounds for ReLU and Sigmoid networks.

works. The upper bounds are not tight: they could be strengthened by performing stronger attacks. The lower bound for MWD is also unlikely to be tight since it is based on an approximate analysis (see Section 4).

Even with these approximations, there is a large accuracy gap between the lower bound guarantee for MWD and the upper bounds for ReLU and Sigmoid networks. The gap is a visual indication of the stronger resistance to attacks of MWD networks. For $\epsilon \leq 0.2$, the figure shows that the gap between the upper and lower bounds for MWD is relatively small, indicating that the lower bounds of Section 4 are fairly accurate in this regime. As expected, the bounds go to 0 with increasing ϵ.

We stress that MWD lower bound in Figure 2 gives the *provable* margin of resistance of MWD networks with respect to adversarial attacks *as measured on the testing set.* If the test set is a representative sample of the inputs that will be seen while the net is in use, then Figure 2 can be interpreted as saying that, even with optimally perturbed ϵ=0.2-adversarial examples, our MWD network still provides an accuracy of over 80%. Of course, this guarantee does not hold if the examples to be predicted on come from a wildly different distribution. In other words, our accuracy guarantee is conditional to a given input distribution — as is typical in machine learning.

MWD vs. adversarially-trained ReLU networks. Including adversarial examples in the training set is the most common method used to make neural networks more resistant to adversarial attacks [GSS14, MMS$^+$17]. Therefore, it is interesting

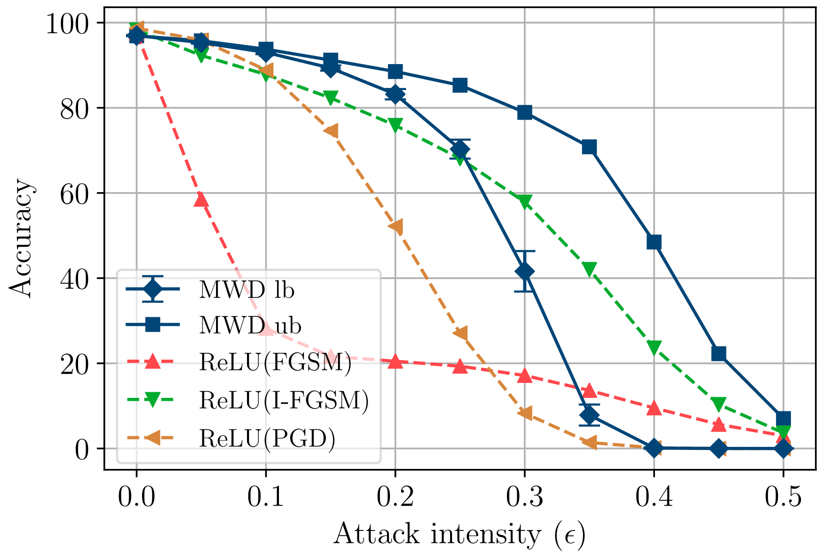

Figure 3: Upper and lower accuracy bounds for MWD vs. upper accuracy bounds for ReLU networks trained with adversarial examples.

to compare the lower bound guarantees for MWD networks with the upper bounds on the true ϵ-attack accuracies ReLU networks trained via a mix of normal and heuristically generated adversarial examples. For brevity, we omit the results for Sigmoid networks, as they were consistently inferior to those for ReLU networks. Before training on each batch of 100 labeled examples, we add candidate adversarial examples to the batch in one of three different ways.

- **ReLU(FGSM)** and **ReLU(I-FSGM):** for each $(\boldsymbol{x}, t)$ in the batch, we construct a potential adversarial example $\tilde{\boldsymbol{x}}$ via (3) or (4), and we feed both $(\boldsymbol{x}, t)$ and $(\tilde{\boldsymbol{x}}, t)$ to the network for training.
- **ReLU(PGD):** for each $(\boldsymbol{x}, t)$ in the batch, we perform 100 steps of projected gradient descent from a point chosen at random in $B_\epsilon(\boldsymbol{x}) \cap [0, 1]^n$; denoting by $\boldsymbol{x}'$ the ending point of the projected gradient descent, we feed both $(\boldsymbol{x}, t)$ and $(\boldsymbol{x}', t)$ to the network for training.

The candidate adversarial examples were generated with $\epsilon = 0.3$, which is consistent with [MMS+17]. Due to the high computational cost of adversarial training (and in particular, PGD adversarial training), we performed one run, and we trained the networks for 10 epochs, which was sufficient for their accuracy to plateau.

In Figure 3 we compare the previously reported accuracy upper and lower bounds for MWD, with the accuracy upper bounds for adversarially-trained ReLU networks. The accuracy upper bounds for adversarially-trained ReLU networks are obtained via I-FGSM attacks, which are more effective than PGD or FGSM attacks against such networks. We see that the upper bounds for MWD networks are above the upper bounds for adversarially-trained ReLU networks for all but the smallest attacks (for $\epsilon \geq 0.05$). Furthermore, we see that the lower bounds for MWD networks are above the upper bounds for adversarially-trained ReLU networks for a good range of attack sizes ($0.05 \leq \epsilon \leq 0.25$), in spite of both upper and lower bounds being conservative. Together, this indicates that at least in the training regimes we explored, MWD trained without the benefit of adversarial examples offer more resistance to adversarial attacks than ReLU networks trained with the benefit of adversarial examples.

Training MWD networks with pseudogradients vs. standard gradients. We compared the performance achieved by training MWD networks with standard gradients, and with pseudogradients. We considered networks with 512, 512, 512, and 10 units, where the first three layers consisted of a random mix of And and Nand units, while the last layer was composed of Nand units (the results do not depend strongly on such unit choices). After 30 epochs of training, pseudogradients yielded $(96.79 \pm 0.17)\%$ accuracy, while regular gradients only $(86.35 \pm 0.75)\%$. On smaller networks, that should be easier to train, the gap even widened: for networks with 128, 128, and 10 units, pseudogradients yielded $(95.00 \pm 0.29)\%$ accuracy and regular gradients only $(82.40 \pm 3.72)\%$. This indicates the need of resorting to pseudogradients for training MWDnetworks.

8 Conclusions and Future Work

In this paper we advanced the state of the art in producing neural networks resistant to adversarial attacks via two contributions. We introduced MWD network units, whose non-linear structure makes them intrinsically resistant to attacks, along with techniques for training networks including MWD units. We also provided an efficient technique for computing accuracy guarantees for networks under adversarial attacks, showing that MWD networks provide guarantees that are superior to common network types such as ReLU and sigmoid networks.

Much work remains to be done, including extending the results to convolutional networks, and exploring the design space of trainable nonlinear structures is clearly an interesting endeavor.

References

[BL88a] David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[BL88b] David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[CCG91] Sheng Chen, Colin FN Cowan, and Peter M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on neural networks*, 2(2):302–309, 1991.

[CW16] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[CW17a] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[CW17b] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[CW18] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *arXiv preprint arXiv:1801.01944*, 2018.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, California, 1979.

[GSS14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[KGB16a] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[KGB16b] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[MMS$^+$17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. June 2017.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[Orr96] Mark JL Orr. *Introduction to Radial Basis Function Networks.* Technical Report, Center for Cognitive Science, University of Edinburgh, 1996.

[PGC$^+$17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[PMJ$^+$16] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium On*, pages 372–387. IEEE, 2016.

[PMW$^+$16] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[PRGS17] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 804–813. Curran Associates, Inc., 2017.

[SZS$^+$13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*, December 2013.

[TKP$^+$17] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[TPG$^+$17] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[Zei12] Matthew D. Zeiler. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.