JumpReLU: A Retrofit Defense Strategy for Adversarial Attacks

N. Benjamin Erichson[1], Zhewei Yao[2] and Michael W. Mahoney[1]
[1]*ICSI and Department of Statistics, University of California at Berkeley, U.S.A.*
[2]*Department of Mathematics, University of California at Berkeley, U.S.A.*

Keywords: Adversarial Learning, Robust Learning, Deep Neural Networks.

Abstract: It has been demonstrated that very simple attacks can fool highly-sophisticated neural network architectures. In particular, so-called adversarial examples, constructed from perturbations of input data that are small or imperceptible to humans but lead to different predictions, may lead to an enormous risk in certain critical applications. In light of this, there has been a great deal of work on developing adversarial training strategies to improve model robustness. These training strategies are very expensive, in both human and computational time. To complement these approaches, we propose a very simple and inexpensive strategy which can be used to “retrofit” a previously-trained network to improve its resilience to adversarial attacks. More concretely, we propose a new activation function—the JumpReLU—which, when used in place of a ReLU in an already-trained model, leads to a trade-off between predictive accuracy and robustness. This trade-off is controlled by the jump size, a hyper-parameter which can be tuned during the validation stage. Our empirical results demonstrate that this increases model robustness, protecting against adversarial attacks with substantially increased levels of perturbations. This is accomplished simply by retrofitting existing networks with our JumpReLU activation function, without the need for retraining the model. Additionally, we demonstrate that adversarially trained (robust) models can greatly benefit from retrofitting.

1 INTRODUCTION

As machine learning methods become more integrated into a wide range of technologies, there is a greater demand for robustness, in addition to the usual efficiency and high-quality prediction, in machine learning algorithms. Deep neural networks (DNNs), in particular, are ubiquitous in many technologies that shape the modern world (LeCun et al., 2015; Goodfellow et al., 2016), but it has been shown that even the most sophisticated network architectures can easily be perturbed and fooled by simple and imperceptible attacks. For instance, single pixel changes which are undetectable to the human eye can fool DNNs into making erroneous predictions. These adversarial attacks can reveal important fragilities of modern neural networks (Szegedy et al., 2013; Goodfellow et al., 2014; Liu et al., 2016), and they can reveal flaws in network training and design which pose security risks (Kurakin et al., 2016). Partly due to this, evaluating and improving the robustness of DNNs is an active area of research. Due to the unpredictable and sometimes imperceptible nature of adversarial attacks, however, it can be difficult to test and evaluate network robustness comprehensively. See, *e.g.*, Figure 1, which provides an illustration of how a relatively small adversarial perturbation can lead to incorrect classification.

Most work in this area focuses on training, *e.g.*, developing adversarial training strategies to improve model robustness. These training strategies are expensive, in both human and computational time. For example, a single training run can be expensive, and typically many runs are needed, as the analyst “fiddles with” parameters and hyper-parameters.

Motivated by this observation, we propose a complementary approach to improve the robustness of the model to the risk of adversarial attacks. The rectified linear unit (ReLU) will be our focus here since it is the

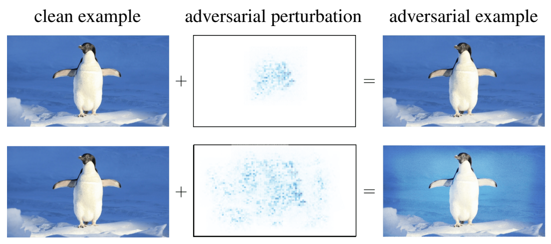


Figure 1: Adversarial examples are constructed by perturbing a clean example with a small amount of non-random noise in order to fool a classifier. Often, an imperceptible amount of noise is sufficient to fool a model (top row). The JumpReLU improves the robustness, *i.e.*, a higher level of noise is required to fool the retrofitted model (bottom row).

Erichson, N., Yao, Z. and Mahoney, M.
JumpReLU: A Retrofit Defense Strategy for Adversarial Attacks.
DOI: 10.5220/0009316401030114
In *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2020)*, pages 103-114
ISBN: 978-989-758-397-1; ISSN: 2184-4313

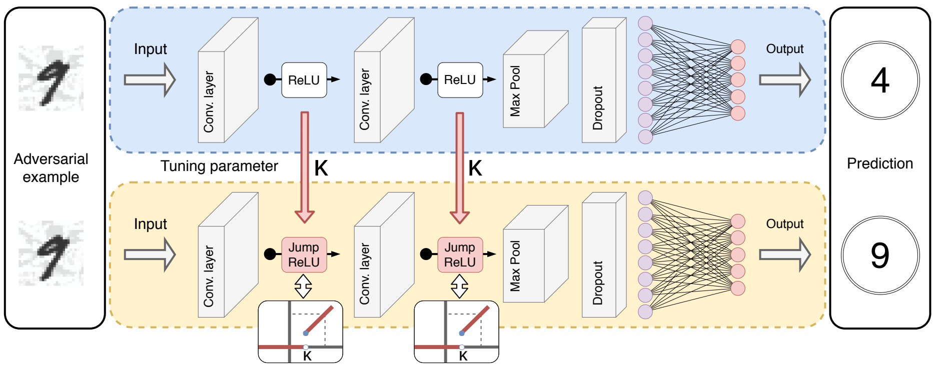


Figure 2: Simplified illustration of a neural network architecture using ReLU activation functions. JumpReLU can be activated by setting the jump size (threshold value) κ larger than zero to increase the resilience to adversarial attacks. (One could use different values of κ for different layers, but we did not observe that to help.)

most widely-used and studied activation function in the context of adversarial attacks (but we expect that the same idea can be applied more generally). For networks trained with ReLUs, our method will replace the ReLU with what we call a *JumpReLU* function, a variant of the standard ReLU that has a jump discontinuity. See Figure 2 for an illustration of the basic method. The jump discontinuity in the JumpReLU function has the potential to dampen the effect of adversarial perturbations, and we will "retrofit" existing ReLU-based networks by replacing the ReLU with the JumpReLU, as a defense strategy, to reduce the risk of adversarial attacks. The magnitude of the jump is a parameter which controls the trade-off between predictive accuracy and robustness, and it can be chosen in the validation stage, *i.e.*, without the need to retrain the network. In more detail, our contributions are the following:

- We introduce and propose the *JumpReLU activation function*, a novel rectified linear unit with a small jump discontinuity, in order to improve the robustness of trained neural networks.
- We show that the JumpReLU activation function can be used to "retrofit" already deployed, *i.e.*, pre-trained, neural networks—*without the need to perform an expensive retraining of the original network*. Our empirical results show that using the JumpReLU in this way leads to networks that are resilient to substantially increased levels of perturbations, when defending classic convolutional networks and modern residual networks. We also show that JumpReLU can be used to enhance adversarially trained (robust) models.
- We show that the popular Deep Fool method requires increased noise levels by a factor of about 3–7 to achieve nearly 100 percent fooling rates for the retrofitted model on CIFAR10. We show that these increased noise levels are indeed critical, *i.e.*, the detection rate of adversarial examples is substantially increased when using an additional add-on detector.
- The magnitude of the jump is an additional hyper-parameter in the JumpReLU activation function that provides a trade-off between predictive accuracy and robustness. This single parameter can be efficiently tuned during the validation stage, *i.e.*, without the need for network retraining.

In summary, the JumpReLU activation function improves the model robustness to adversarial perturbations, while attaining a "good" accuracy for clean examples. Further, the impact on the architecture is minimal and does not effect the inference time.

2 RELATED WORK

Adversarial examples are an emerging threat for many machine learning tasks. Szegedy *et al.* (Szegedy et al., 2013) discovered that neural networks are particularly susceptible to such adversarial examples. This can lead to problems in safety- and security-critical applications such as medical imaging, surveillance, autonomous driving, and voice command recognition. Due to its importance, adversarial learning has become an intense area of research, posing a cat-and-mouse game between attackers and defenders.

Indeed, there is currently a lack of theory to explain why deep learning is so sensitive to this form of attack. Early work hypothesized that the highly non-linear characteristics of neural networks and the tendency toward almost perfect interpolation of the training data are the reasons for this phenomena. Tanay and Griffin (Tanay and Griffin, 2016) argued that the adversarial strength is related to the level of regularization and that the effect of adversarial examples can be mitigated by using a proper level of regularization. In contrast, Goodfellow *et al.* (Goodfellow et al., 2014) impressively demonstrated that the linear structure in deep networks with respect to their inputs is sufficient to craft adversarial examples.

Let x be an input such as an image. The problem of crafting an adversarial example $\tilde{x}$

$$\tilde{x} = x + \Delta x \tag{1}$$

requires finding an additive perturbation Δx so that $\tilde{x}$ fools a specific model F under attack. The minimal perturbation with respect to a p-norm $\|\cdot\|_p$ can be obtained by using an optimization based strategy which aims to minimize

$$\Delta x := \underset{\Delta\hat{x}}{\arg\min} \|\Delta\hat{x}\|_p \quad \text{s.t.} \quad F(x+\Delta\hat{x}) \neq F(x), \tag{2}$$

so that the example x is misclassified.

Note, the perturbation used to construct adversarial examples needs to be small enough to be unnoticeable for humans, or add-on detection algorithms. Intuitively, the average minimum perturbation which is required to fool a given model yields a plausible metric to characterize the robustness of a model (Papernot et al., 2016). Hence, we can quantify the robustness for a trained model F as

$$\rho_F := \mathbb{E}_{(X,Y)\sim\mathcal{D}} \left[\frac{\|\Delta X\|_p}{\|X\|_p} \right], \tag{3}$$

where the input-target-pairs (X,Y) are drawn from distribution $\mathcal{D}$, and ΔX is the minimal perturbation that is needed to fool the model F.

2.1 Attack Strategies

There are broadly two types of attacks: targeted and non-targeted attacks. Targeted attacks aim to craft adversarial examples which fool a model to predict a specific class label. Non-targeted attacks have a weaker objective, *i.e.*, simply to classify an adversarial example incorrectly.

Independent of the type, attack strategies can be categorized broadly into two families of threat models. *Black-box attacks* aim to craft adversarial examples without any prior knowledge about the target model (Su et al., 2017; Sarkar et al., 2017; Cisse et al., 2017; Dong et al., 2017). *White-box attacks*, in contrast, require comprehensive prior knowledge about the target model. There are several popular white-box attacks for computer vision applications (Szegedy et al., 2013; Goodfellow et al., 2014; Liu et al., 2016; Moosavi-Dezfooli et al., 2016; Kurakin et al., 2016; Poursaeed et al., 2017). *Gray-box* attacks are a slightly weaker threat model in which the adversary has only partial knowledge about the target model.

The following (non-targeted) attack methods are particularly relevant for our results.

- First, the Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014), which crafts adversarial perturbations Δx by using the sign of the gradient of the loss function $\mathcal{L}$ with respect to the clean input image x. Let's assume that the true label of x is y. Then, the adversarial example $\tilde{x}$ is constructed as

$$\tilde{x} = x + \varepsilon \cdot \text{sign}(\nabla_x \mathcal{L}(F(x), y)), \tag{4}$$

  where ε controls the magnitude of the perturbation. Here, the operator sign is an element-wise function, extracting the sign of a real number. Relatedly, the iterative variant IFGSM (Kurakin et al., 2016) constructs adversarial examples using $1,\ldots,k$ steps

$$\tilde{x}_k = \text{clip}_x \left[\tilde{x}_{k-1} + \varepsilon \cdot \text{sign}(\nabla \mathcal{L}(F(\tilde{x}_{k-1}), y))\right], \tag{5}$$

  where clip_x is an element-wise clipping function. This approach is essentially a projected gradient descent (PGD) method (Madry et al., 2017).

- Second, the Deep Fool (DF) method, which is also an iterative method. (Moosavi-Dezfooli et al., 2016). The DF method first approximates the model under consideration as a linear decision boundary, and then seeks the smallest perturbation needed to push an input image over that boundary. DF can minimize the loss function using either the L_∞ or L_2 norm.

- Third, the recently introduced trust region (TR) based attack method (Yao et al., 2018). The TR method performs similarly to the Carlini and Wagner (CW) (Carlini and Wagner, 2017) attack method, but is more efficient in terms of the computational resources required to construct the adversarial examples.

2.2 Defense Strategies

Small perturbations are often imperceptible for both humans and the predictive models, making the design of counterattacks a non-trivial task. Commonly used techniques for preventing overfitting (*e.g.*, including weight decay and dropout layers and then retraining) do not robustify the model against adversarial examples. Akhtar and Mian (Akhtar and Mian, 2018) segment defense strategies into three categories.

The first category includes strategies which rely on specialized add-on (external) models which are used to defend the actual network (Akhtar et al., 2017; Lee et al., 2017; Shen et al., 2017; Xu et al., 2017). The second category includes defense strategies which modify the network architecture in order to increase the robustness (Gu and Rigazio, 2014; Ross and Doshi-Velez, 2017; Papernot et al., 2016; Nayebi and Ganguli, 2017; Guo et al., 2017). Closely related to our work, Zantedeschi *et al.* (Zantedeschi et al., 2017) recently proposed a bounded ReLU activation function as an efficient defense against adversarial attacks. Their motivation is to dampen large signals to prevent accumulation of the adversarial perturbation over layers as a signal propagates forward, using the function. The third category aims to modify the input data for the training and validation stage in order to improve the robustness of the model (Miyato et al., 2016; Guo et al., 2017; Bhagoji et al., 2018; Luo et al., 2015; Liao et al., 2017).

A drawback of most state-of-the-art defense strategies is that they involve modifying the network architecture. Such strategies require that the new network is re-trained or that new specialized models are trained from scratch. This retraining is expensive in both human and computation time. Further, specialized external models can require considerable effort to be deployed and often increase the need of computational resources and inference time.

3 JumpReLU

The rectified linear unit (ReLU) and its variants have arguably emerged as the most popular activation functions for applications in the realm of computer vision. The ReLU activation function has beneficial numerical properties, and also has sparsity promoting properties (Glorot et al., 2011). Indeed, sparsity is a widely used concept in statistics and signal processing (Hastie et al., 2015). For a given input x and an arbitrary function $f : \mathcal{R} \to \mathcal{R}$, the ReLU function can be defined as the positive part of the filter output $z = f(x)$ as

$$R(z) := \max(z, 0), \tag{6}$$

illustrated in Figure 3a. The ReLU function is also known as the ramp function which has several other interesting definitions. For instance, we can define the ReLU function as

$$R(z) := zH(z), \tag{7}$$

where H is the discrete Heaviside unit step function

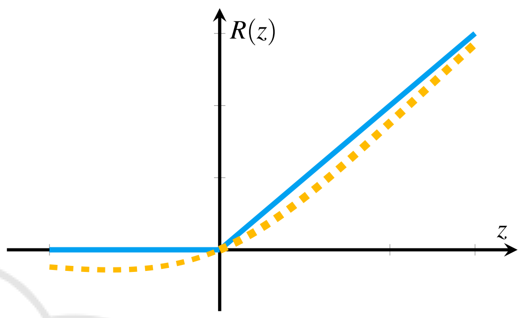
$$H(z) := \begin{cases} 0 & \text{if } z \leq 0, \\ 1 & \text{if } z > 0. \end{cases} \tag{8}$$

Alternatively, the logistic function can be used for smooth approximation of the Heaviside step function

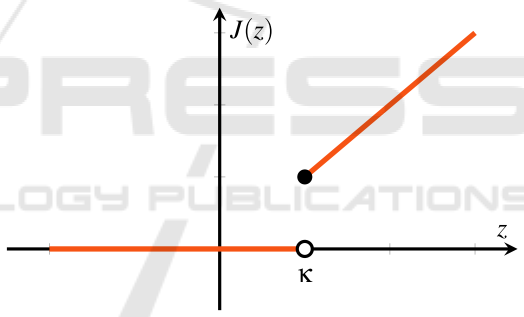
$$H(z) :\approx \frac{1}{1 + \exp(-2\beta z)}. \tag{9}$$

This smooth approximation resembles the Swish activation function (Ramachandran et al., 2017), which is defined as

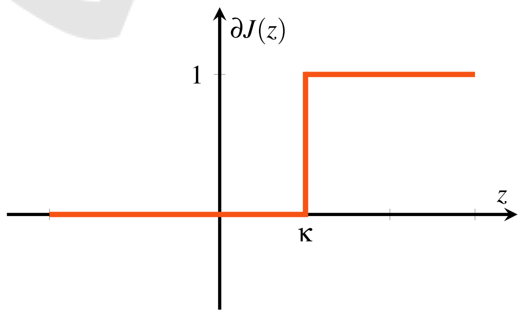
$$S(z) := z\frac{1}{1 + \exp(-2\beta z)}. \tag{10}$$



(a) ReLU and Swish (dashed).



(b) JumpReLU activation function.



(c) Subgradient of JumpReLU is shifted by κ.

Figure 3: The ReLU is the most widely studied activation function in context of adversarial attacks, shown in (a). In addition its smooth approximation (Swish) is shown, with $\beta = 0.5$. The JumpReLU activation (b) introduces robustness and an additional amount of sparsity, controlled via the jump size (threshold value) κ. In other words, JumpReLU suppresses small positive signals. The corresponding subdifferential $\partial J(z)$ as a function of z is shown in (c).

The ReLU activation function works extremely well in practice. However, a fixed threshold value 0 seems arbitrary. Looking to biophysical systems can inspire more biologically plausible as well as more robust neural networks and activation functions (Glorot et al., 2011; Nayebi and Ganguli, 2017). In particular, the brain utilizes nonlinear activation functions to filter out noise sources in the neural response. Biophysical properties inherent to each neuron determines a threshold for eliciting an action potential (Amit, 1989). This allows the neuron to collect and transmit relevant information. Indeed, thresholding filters out noise sources in the neural response. This allows the neuron to collect and transmit relevant information. Thus, it seems reasonable to crop activation functions so that they turn on only for inputs greater than or equal to the threshold value κ. In this case, sub-threshold signals are suppressed, while significant signals are allowed to pass.

We introduce the JumpReLU function which suppresses signals of small magnitude and negative sign

$$J(z) := zH(z-\kappa) = \begin{cases} 0 & \text{if } z \leq \kappa \\ z & \text{if } z > \kappa, \end{cases} \tag{11}$$

illustrated in Figure 3b. This activation function introduces a jump discontinuity, yielding piece-wise continuous functions. While this idea can likely be transferred to other activations, we restrict our focus to the family of discrete ReLU activation functions.

Glorot *et al*. (Glorot et al., 2011) note that too much sparsity can negatively affect the predictive accuracy. Indeed, this might be an issue during the training stage, however, a fine-tuned threshold value κ can improve the robustness of the model during the validation stage by introducing an extra amount of sparsity. The tuning parameter κ can be used to control the trade-off between predictive accuracy and robustness of the model. Importantly, JumpReLU can be used to retrofit previously trained networks in order to mitigate the risk to be fooled by adversarial examples.

Note that the threshold value κ can be tuned cheaply during the validation stage once the network is trained, *i.e*., without the need for expensive retraining of the model.

4 EXPERIMENTS

We first outline the setup which we use to evaluate the performance of the proposed JumpReLU activation function. We restrict our evaluation to MNIST and CIFAR10, since these are the two standard datasets which are most widely used in the literature to study adversarial attacks.

- The MNIST dataset (LeCun et al., 1998) provides 28×28 gray-scale image patches for 10 classes (digits), comprising $60,000$ instances for training, and $10,000$ examples for validation. For our experiments we use a LeNet5 like architecture with an additional dropout layer.
- The CIFAR10 dataset (Krizhevsky and Hinton, 2009) provides 32×32 RGB image patches for 10 classes, comprising $50,000$ instances for training, and $10,000$ examples for validation. For our CIFAR10 experiments we use a simple AlexLike architecture proposed by (Carlini and Wagner, 2017); a wide ResNet architecture (Zagoruyko and Komodakis, 2016) of depth 30 and with width factor 4; and a MobileNetV2 architecture which is using inverted residuals (Sandler et al., 2018).

We aim to match the experimental setup for creating adversarial examples as closely as possible to prior work. Thus, we follow the setup proposed by Madry *et al*. (Madry et al., 2017) and Buckman *et al*. (Buckman et al., 2018). For all MNIST experiments we use $\varepsilon = 0.01$ and 40 attack iterations; for experiments on CIFAR10 we use the same ε, and 7 steps for PGD and Deep Fool attacks. For the TR attack method we use 1000 steps. Note, these values are chosen by following the assumption that an attacker aims to construct adversarial examples which have imperceptible perturbations. Further, we assume that the attacker has only a limited budget of computational resources at disposal.

We also evaluate the effectiveness of the JumpReLU for adversarially trained networks. Training the model with adversarial examples drastically increases the robustness with respect to the specific attack method used to generate the adversarial training examples (Goodfellow et al., 2014). Here, we use the FGSM method to craft examples for adversarial training with $\varepsilon = 0.3$ for MNIST, and $\varepsilon = 0.03$ for CIFAR10. Unlike Madry *et al*. (Madry et al., 2017), we perform robust training with mixed batches composed of both clean and adversarial examples. This leads to an improved accuracy on clean examples, while being slightly less robust. The specific ratio of the numbers of clean to adversarial examples can be seen as a tuning parameter, which may depend on the application.

4.1 Results

In the following, we compare the performance of JumpReLU to the standard ReLU activation function for both gray-box and white-box attack scenarios. For each scenario, we consider three different iterative attack methods: the Projected Gradient Descent (PGD) method, the Deep Fool (DF) method using both the

L_2 (denoted as DF_2) and L_∞ norm (denoted as DF_∞) as well as the Trust Region (TR) attack method. Here, we use the TR method as a surrogate for the more popular Carlini and Wagner (CW) (Carlini and Wagner, 2017) attack method. This is because the CW method requires enormous amounts of computational resources to to construct adversarial examples, and recent work has shown that the TR method performs similarly with much less computational cost (Yao et al., 2018). For instance, it takes about one hour to construct 300 adversarial examples for CIFAR10 using the CW method, despite using a state-of-the-art GPU and the implementation provided by (Rauber et al., 2017).

4.1.1 Gray-box Attack Scenario

We start our evaluation by considering the gray-box attack scenario. In this "vanilla" flavored setting, we assume that the adversary has only partial knowledge about the model. Here, the adversary has full access to the ReLU network to construct adversarial examples, but it has no information about the JumpReLU setting during inference time.

Table 1 shows a summary of results for MNIST and CIFAR10 using different network architectures. The ReLU network is used as a source network to craft adversarial examples which are then used to attack the JumpReLU network. We present results for models trained on clean data only (base) and adversarially trained models (robust).

The results presented in Table 1 show that the positive benefits of JumpReLU are pronounced across the different datasets and architectures, while the loss of accuracy on clean examples is moderate. Clearly, more complex residual networks appear to be more vulnerable than the simpler AlexLike network. Still, JumpReLU is able to increase the robustness with respect to Deep Fool attacks.

Surprisingly, the JumpReLU is also able to substantially improve the resilience of robustly trained models. Indeed, this demonstrates the flexibility of our approach and shows that retrofitting is not limited to weak models only. Further, we see that the adversarially trained models are more robust with respect to the specific attack method used for training, while still being vulnerable to other attack methods. In contrast, our defense strategy based on the JumpReLU is agnostic to specific attack methods, *i.e.*, we improve the robustness with respect to all attacks considered here. Note we could further increase the jump value for the robust models, in order to increase the robustness to the Deep Fool and TR attack method. However, this comes with the price of sacrificing slightly more accuracy on clean data.

Table 1: Summary of results for gray-box attacks. The numbers indicate the accuracy, *i.e.*, the percentage of correctly classified instances (higher numbers indicate better robustness). The ReLU network (indicated by a '*') is used as the source model to generate adversarial examples. The best performance in each category is highlighted in bold letters. Robust adversarial trained models are highlighted in gray.

Model	Accuracy	PGD	DF_∞	DF_2	TR
ReLU (Base)*	**99.55%**	66.69%	0.0%	0.0%	0.0%
JumpReLU	99.53%	91.65%	**81.39%**	**58.93%**	**58.90**
ReLU (Robust)*	99.50%	91.39%	0.0%	0.0%	0.0%
JumpReLU	99.47%	**97.07%**	70.84%	45.17%	53.24%

Results for LeNet like network (MNIST); $\kappa = 1.0$.

Model	Accuracy	PGD	DF_∞	DF_2	TR
ReLU (Base)*	**89.46%**	6.38%	0.0%	0.0%	0.0%
JumpReLU	87.52%	45.75%	**61.82%**	**60.55%**	**53.08%**
ReLU (Robust)*	87.93%	51.88%	0.0%	0.0%	0.0%
JumpReLU	86.19%	**67.65%**	52.28%	46.9%	51.52%

Results for AlexLike network (CIFAR10); $\kappa = 0.4$.

Model	Accuracy	PGD	DF_∞	DF_2	TR
ReLU (Base)*	**94.31%**	0.0%	0.0%	0.0%	0.0%
JumpReLU	92.58%	0.39%	**37.33%**	**40.21%**	**45.90%**
ReLU (Robust)*	93.72%	60.43	0.0%	0.0%	0.0%
JumpReLU	93.01%	**70.25%**	28.62%	26.11%	35.33%

Results for Wide ResNet (CIFAR10); $\kappa = 0.07$.

Model	Accuracy	PGD	DF_∞	DF_2	
ReLU (Base)*	**92.07%**	0.0%	0.0%	0.0%	0.0%
JumpReLU	90.43%	0.54%	**40.69%**	**41.61%**	**43.18%**
ReLU (Robust)*	91.69%	53.98	0.0%	0.0%	0.0%
JumpReLU	90.12%	**66.37%**	37.31%	35.45%	40.4%

Results for MobileNetV2 (CIFAR10); $\kappa = 0.06$.

4.1.2 White-box Attack Scenario

We next consider the more challenging white-box attack scenario. Here, the adversary has full knowledge about the model, and it can access their gradients.

Table 2 summarizes the results for the different datasets and architectures under consideration. Again, we see some considerable improvements for the retrofitted models—especially, the retrofitted robustly trained wide ResNet and MobileNetV2 excel. The performance of JumpReLU is even competitive in comparison to more sophisticated techniques such as one-hot and thermometer encoding (the authors provide only scores for the FGSM and PGD attack method) (Buckman et al., 2018). This is despite the fact that JumpReLU does not require that the model is re-trained from scratch. We can simply select a suitable jump value κ during the validation stage. The choice of the jump value depends thereby on the desired trade-off between accuracy and robustness, *i.e.*, large jump values improve the robustness, while decreasing the accuracy on clean examples. We also considered comparing with the bounded ReLU method (Zantedeschi et al., 2017), but our preliminary results showed a poor performance.

The adversarially trained (robust) models provide a good defense against the PGD attack. However, Deep Fool is able to fool all instances in the test set using only 7 iterations, and TR using 1000 iterations. On first glance, this performance seems to be undesirable. We can see, however, that Deep Fool requires substantially increased average minimum perturbations in order to achieve such a high fool rate. The numbers in parentheses in Table 2 indicate the

Table 2: Summary of results for white-box attacks. The numbers indicate the accuracy, *i.e.*, the percentage of correctly classified instances (higher numbers indicate better robustness). The Deep Fool method is able to fool all instances using only 7 iterations, hence we show here the average minimum perturbations in parentheses. The best performance in each category is highlighted in bold letters. Robust adversarial trained models are highlighted in gray.

Model	Accuracy	PGD	DF$_\infty$	DF$_2$	TR
ReLU (Base)	**99.55%**	66.69%	(17.9%)	(21.8%)	(18.9%)
JumpReLU	99.53%	83.21%	(34.1%)	(44.9%)	(25.0%)
ReLU (Robust)	99.50%	91.39%	(28.4%)	(31.4%)	(24.7%)
JumpReLU	99.47%	**94.36%**	(**46.6%**)	(**53.3%**)	(**32.8%**)
Madry	98.80%	93.20%	-	-	-
Vanilla	99.03%	91.36%	-	-	-
One-hot	99.01%	93.77%	-	-	-
Thermo	99.23%	93.70%	-	-	-

Results for LeNet like network (MNIST); $\kappa = 1.0$.

Model	Accuracy	PGD	DF$_\infty$	DF$_2$	TR
ReLU (Base)	**89.46%**	6.38%	(1.2%)	(1.5%)	(1.3%)
JumpReLU	87.52%	18.56%	(9.80%)	(10.6%)	(1.7%)
ReLU (Robust)	87.93%	51.88%	(3.6%)	(4.2%)	(3.6%)
JumpReLU	86.19%	**56.70%**	(**13.2%**)	(**14.1%**)	(**4.3%**)

Results for AlexLike network (CIFAR10); $\kappa = 0.4$.

Model	Accuracy	PGD	DF$_\infty$	DF$_2$	TR
ReLU (Base)	**94.31%**	0.37%	(1.4%)	(1.8%)	(1.3%)
JumpReLU	92.58%	0.95%	(14.3%)	(18.5%)	(1.9%)
ReLU (Robust)	93.72%	60.43%	(6.4%)	(7.5%)	(4.8%)
JumpReLU	93.01%	**67.89%**	(**44.4%**)	(**43.8%**)	(**6.1%**)
Madry	87.3%	50.0%	-	-	-
Vanilla	87.16%	34.71%	-	-	-
One-hot	92.19%	58.96%	-	-	-
Thermo	92.32%	65.67%	-	-	-

Results for WideResNet (CIFAR10); $\kappa = 0.07$.

Model	Accuracy	PGD	DF$_\infty$	DF$_2$	TR
ReLU (Base)	**92.07%**	0.74%	(0.7%)	(0.9%)	(0.7%)
JumpReLU	91.10%	0.92%	(5.3%)	(6.8%)	(1.0%)
ReLU (Robust)	91.69%	53.98%	(4.7%)	(5.3%)	(4.1%)
JumpReLU	90.12%	**59.66%**	(**62.6%**)	(**51.4%**)	(**4.9%**)

Results for MobileNetV2 (CIFAR10); $\kappa = 0.06$.

average minimum perturbations which are needed to achieve a nearly 100 percent fooling rate. These numbers provide a measure for the empirical robustness of the model, which we compute by using the following plug-in estimator

$$\tilde{\rho}_F := \frac{1}{n}\sum_{i}^{n} \frac{\|x_i - \tilde{x}_i\|_p}{\|x_i\|_p}, \qquad (12)$$

with $\tilde{x}_i = x_i + \Delta x_i$. Here, we compute the relative rather than absolute perturbations. The relative measure provides a more intuitive interpretation, *i.e.*, the numbers reflecting the average percentage of changed information in the adversarial examples.

The numbers show that the retrofitted models feature an improved robustness, while maintaining a ‘good’ predictive accuracy for clean examples. For MNIST, the noise levels need to be increased by a factor of about 2 in order to achieve a 100 percent fooling rate. Here, we set the JumpReLU threshold value to $\kappa = 1.0$. For CIFAR10, we achieve a stellar performance of resilience to the Deep Fool attacks, *i.e.*, the noise levels are required to be increased by a factor of 3 to 7 to achieve a successful attack.

Clearly, we can see that the TR method is a stronger attack than Deep Fool. However, we are still able to achieve an improved resilience to this attack.

4.1.3 Performance Trade-offs

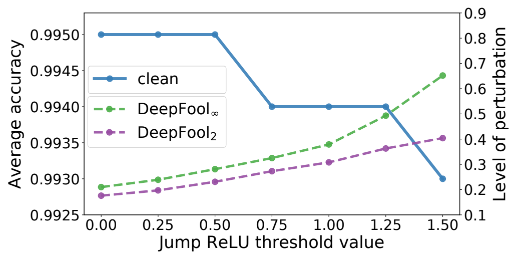
As mentioned, the JumpReLU activation function provides a trade-off between robustness and classification accuracy. The user can control this trade-off in a post-training stage by tuning the threshold value κ, where $\kappa = 0$ resembles the ReLU activation function.

Figure 4 contextualizes this trade-off for the LeNet Like network (MNIST) and the AlexLike network (CIFAR10). We see that the threshold value κ is positively correlated to the level of perturbation which is required in order to achieve a 100 percent fooling rate. Choosing larger threshold values increase the robustness of the model, while sacrificing only a slight amount of predictive accuracy.

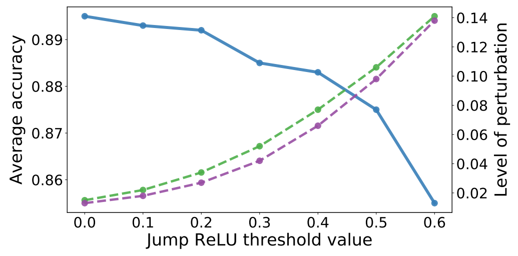
Larger thresholds only marginally affect the accuracy of the LeNet like network on clean examples, while the the AlexLike network (CIFAR10) is more sensitive. Thus, the decision of a ‘good’ jump value is application dependent.

4.1.4 Visual Results

The interested reader may ask whether the increased adversarial perturbations are of any practical significance. To address this question, we show some visual results which illustrate the magnitude of the effect.



(a) LeNet like network (MNIST)



(b) AlexLike network (CIFAR10)

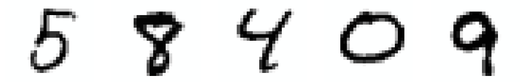
Figure 4: JumpReLU performance trade-offs for MNIST and CIFAR10. The left axis shows the average predictive accuracy for clean examples for varying values of the JumpReLU threshold. The right axis shows the minimum level of perturbation required to construct adversarial examples which achieve a nearly 100% fooling rate.

Recall the aim of the adversary is to construct unobtrusive adversarial examples.

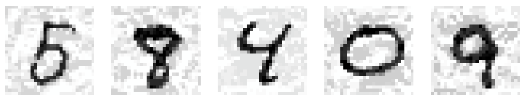
Figure 5 shows both clean and adversarial examples for the MNIST dataset, which are crafted by the Deep Fool algorithm. Clearly, the adversarial examples which are needed to fool the retrofitted LeNet like network are visually distinct from those examples which are sufficient to fool the unprotected model. We also show the corresponding perturbation patterns, *i.e.*, the absolute pixel-wise difference between the clean and adversarial examples, to better contextualize the difference. Note that we use a “reds” color scheme here: white indicates no perturbations, light red indicates very small perturbations, dark red indicates large perturbations.

Next, Figure 6 shows visual results for the CIFAR10 dataset. It is well known that models for this dataset are highly vulnerable, *i.e.*, very small perturbations Δx are already sufficient for a successful attack. Indeed, the minimal perturbations which are needed to fool the unprotected network (here we show results for the AlexLike network) are nearly imperceptible by visual inspection. In contrast, the crafted adversarial examples to attack the retrofitted model show distinct perturbation patterns, and one can recognize that the examples were altered.

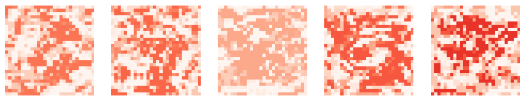
In summary, the visual results put the previously presented relative noise levels into perspective, and



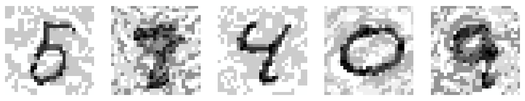
(a) Clean examples which are used for training.



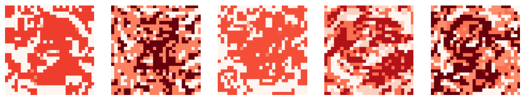
(b) Adversarial examples to fool a standard model.



(c) Perturbation patterns to fool a standard model.



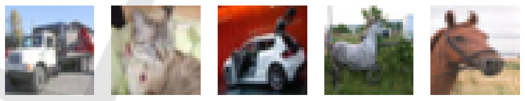
(d) Adversarial examples to fool the retrofitted model.



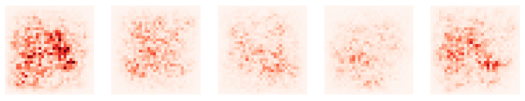
(e) Perturbation patterns to fool the retrofitted model.

Figure 5: Visual results for MNIST to verify the effect of the JumpReLU defense strategy against the DF$_\infty$ attack. Noticeable higher levels of perturbations are required to successfully attack the retrofitted network. Subfigures (c) and (e) show the corresponding perturbation patterns.

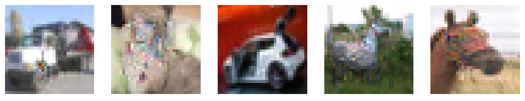
(a) Clean



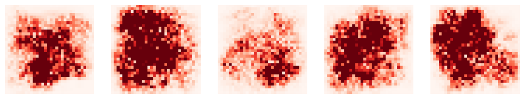
(b) Adversarial examples to fool a standard model.



(c) Perturbation patterns to fool a standard model.



(d) Adversarial examples to fool the retrofitted model.



(e) Perturbation patterns to fool the retrofitted model.

Figure 6: Visual results for CIFAR10. The DF$_2$ attack requires noticeable higher levels of perturbations in order to successfully attack the retrofitted network.

Table 3: AUC scores as measure of the discrimination power between clean and adversarial examples using LID characteristics. Here we compare ReLU and JumpReLU.

Model	PGD	DF_∞	DF_2	TR
LID + ReLU	72.54	73.41	72.93	72.47
LID + JumpReLU	**74.25**	**78.24**	**75.84**	**74.71**

they show that average minimum perturbations of about 5% to 10% are lucid. Thus, it can be concluded that the JumpReLU is an effective strategy for improving the model robustness.

4.1.5 Detection with LID

As a proof-of-concept, we demonstrate that the increased minimum perturbations, which are required to attack the retrofitted model can help to improve the discrimination power of add-on adversarial detectors.

We follow the work by Ma *et al*. (Ma et al., 2018), who use the idea of Local Intrinsic Dimensionality (LID) to characterize adversarial subspaces. The idea is that clean and adversarial examples show distinct patterns so that the LID characteristics allow to discriminate between such examples.

Intuitively, adversarial examples which show increased perturbation patterns should feature more extreme LID characteristics. Hence, a potential application of the JumpReLU is to combine it with an LID based detector. Table 3 shows the area under a receiver operating characteristic curve (AUC) as a measure for the discriminate power between clean and adversarial examples. Indeed, the results show that the combination with JumpReLU improves the detection performance for CIFAR10.

5 CONCLUSION

We have proposed a new activation function—the *JumpReLU* function—which, when used in place of a ReLU in an already pre-trained model, leads to a trade-off between predictive accuracy and robustness. This trade-off is controlled by a parameter, the jump size, which can be tuned during the validation stage. That is, no additional training of the pre-trained model is needed when the JumpReLU function is used. (Of course, if one wanted to perform additional expensive training, then one could do so.) Our experimental results show that this simple and inexpensive strategy improves the resilience to adversarial attacks of previously-trained networks.

Limitations of our approach are standard for current adversarial defense methods, in that stand-alone methods do not guarantee a holistic protection and that sufficiently high levels of perturbation will be able to break the defense. That being said, JumpReLU can easily be used as a stand-alone approach to "retrofit" previously-trained networks, improving their robustness, and it can also be used to support other more complex defense strategies.

ACKNOWLEDGEMENT

NBE and MWM would like to acknowledge support from DARPA and the Air Force Research Laboratory under agreement number FA8750-17-2-0122. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA and the Air Force Research Laboratory or the U.S. Government.

REFERENCES

Akhtar, N., Liu, J., and Mian, A. (2017). Defense against universal adversarial perturbations. *arXiv preprint arXiv:1711.05929*.

Akhtar, N. and Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *arXiv preprint arXiv:1801.00553*.

Amit, D. J. (1989). *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press.

Bhagoji, A. N., Cullina, D., Sitawarin, C., and Mittal, P. (2018). Enhancing robustness of machine learning systems via data transformations. In *Information Sciences and Systems (CISS), 2018 52nd Annual Conference on*, pages 1–5. IEEE.

Buckman, J., Roy, A., Raffel, C., and Goodfellow, I. (2018). Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*.

Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE.

Cisse, M., Adi, Y., Neverova, N., and Keshet, J. (2017). Houdini: Fooling deep structured prediction models. *arXiv preprint arXiv:1707.05373*.

Dong, Y., Liao, F., Pang, T., Su, H., Hu, X., Li, J., and Zhu, J. (2017). Boosting adversarial attacks with momentum. *arXiv preprint arXiv:1710.06081*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

Gu, S. and Rigazio, L. (2014). Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*.

Guo, C., Rana, M., Cisse, M., and van der Maaten, L. (2017). Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical learning with sparsity: the lasso and generalizations*. CRC press.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, H., Han, S., and Lee, J. (2017). Generative adversarial trainer: Defense to adversarial perturbations with gan. *arXiv preprint arXiv:1705.03387*.

Liao, F., Liang, M., Dong, Y., Pang, T., Zhu, J., and Hu, X. (2017). Defense against adversarial attacks using high-level representation guided denoiser. *arXiv preprint arXiv:1712.02976*.

Liu, Y., Chen, X., Liu, C., and Song, D. (2016). Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*.

Luo, Y., Boix, X., Roig, G., Poggio, T., and Zhao, Q. (2015). Foveation-based mechanisms alleviate adversarial examples. *arXiv preprint arXiv:1511.06292*.

Ma, X., Li, B., Wang, Y., Erfani, S. M., Wijewickrema, S., Schoenebeck, G., Song, D., Houle, M. E., and Bailey, J. (2018). Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Miyato, T., Dai, A. M., and Goodfellow, I. (2016). Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*.

Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582.

Nayebi, A. and Ganguli, S. (2017). Biologically inspired protection of deep networks from adversarial attacks. *arXiv preprint arXiv:1703.09202*.

Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE.

Poursaeed, O., Katsman, I., Gao, B., and Belongie, S. (2017). Generative adversarial perturbations. *arXiv preprint arXiv:1712.02328*.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Rauber, J., Brendel, W., and Bethge, M. (2017). Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*.

Ross, A. S. and Doshi-Velez, F. (2017). Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. *arXiv preprint arXiv:1711.09404*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

Sarkar, S., Bansal, A., Mahbub, U., and Chellappa, R. (2017). Upset and angri: Breaking high performance image classifiers. *arXiv preprint arXiv:1707.01159*.

Shen, S., Jin, G., Gao, K., and Zhang, Y. (2017). APW-GAN: Adversarial perturbation elimination with GAN. *arXiv preprint arXiv:1707.05474*.

Su, J., Vargas, D. V., and Kouichi, S. (2017). One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tanay, T. and Griffin, L. (2016). A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint arXiv:1608.07690*.

Xu, W., Evans, D., and Qi, Y. (2017). Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*.

Yao, Z., Gholami, A., Xu, P., Keutzer, K., and Mahoney, M. (2018). Trust region based adversarial attack on neural networks. *arXiv preprint arXiv:1812.06371*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

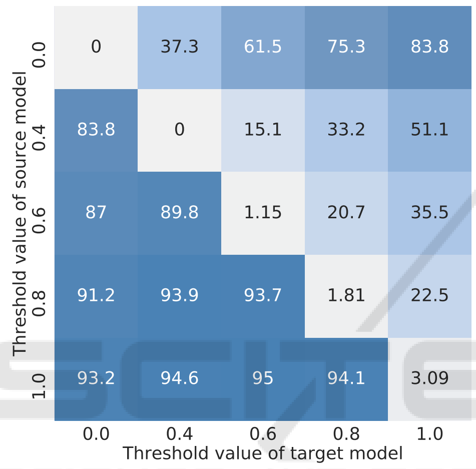
Zantedeschi, V., Nicolae, M.-I., and Rawat, A. (2017). Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 39–49. ACM.

APPENDIX

A Second Look to the Gray-box Attack

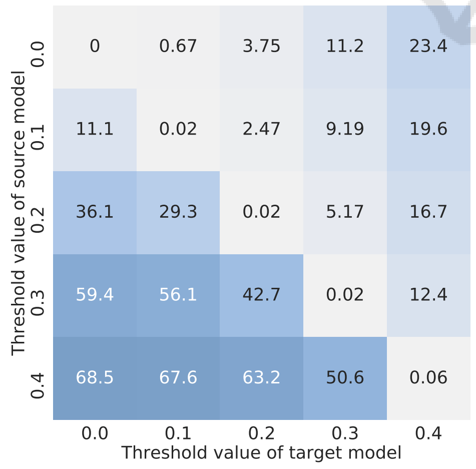
We present an extended set of results for the gray-box attack scenario. Specifically, we study the situation where the adversary has full access to a retrofitted model (which has a fixed jump value) in order to construct adversarial examples, but the adversary has no information about the jump value of the target network during inference time.

Here, the adversarial examples are crafted by using the projected gradient descent (PGD) attack method. Figure 7 shows the efficiency of a non-targeted attack on networks using different jump values. Note, we run the attack with a large number of iterations, enough so that the crafted adversarial examples achieve a nearly 100 percent fool rate for the source model.



Source \ Target	0.0	0.4	0.6	0.8	1.0
0.0	0	37.3	61.5	75.3	83.8
0.4	83.8	0	15.1	33.2	51.1
0.6	87	89.8	1.15	20.7	35.5
0.8	91.2	93.9	93.7	1.81	22.5
1.0	93.2	94.6	95	94.1	3.09

(a) Transferability for MNIST.



Source \ Target	0.0	0.1	0.2	0.3	0.4
0.0	0	0.67	3.75	11.2	23.4
0.1	11.1	0.02	2.47	9.19	19.6
0.2	36.1	29.3	0.02	5.17	16.7
0.3	59.4	56.1	42.7	0.02	12.4
0.4	68.5	67.6	63.2	50.6	0.06

(b) Transferability for CIFAR10.

Figure 7: Gray-box attack matrix for different jump values. Each cell (i, j) indicates the predictive accuracy of a model retrofitted with the jump value j (target), which is being attack by using adversarial examples generated by a model with jump value i (source). Higher cell values indicate better robustness.

We see that the attack is unidirectional, *i.e.*, adversarial examples crafted by using source models which have a low jump value can be used to attack models which have a higher jump value. However, retrofitted models which have a low jump value are resilient to adversarial examples generated by source models which have a large jump value. Thus, one could robustify the network by using a large jump size κ for evaluating the gradient, while using a smaller jump size for inference. Of course, this is a somewhat pathological setup, yet these results reveal some interesting behavioral properties of the JumpReLU.

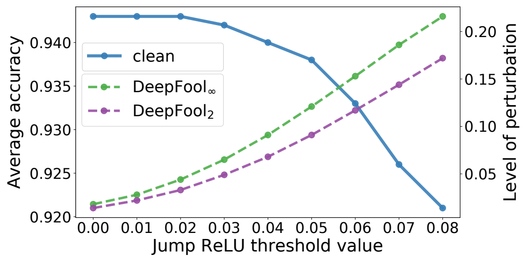
Additional Results to Characterize the Performance Trade-offs

The JumpReLU activation function poses a trade-off between robustness and classification accuracy. Here we provide additional results for both the wide ResNet and MobileNetV2 architectures. Recall, the user can control this trade-off in a post-training stage by tuning the threshold value κ, where $\kappa = 0$ resembles the ReLU activation function.

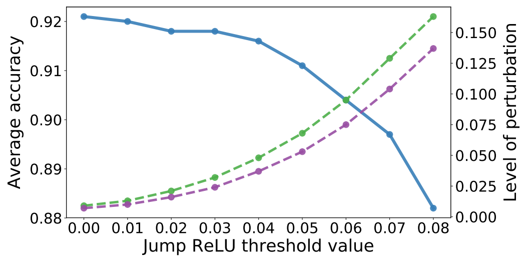
Figure 8 shows the performance trade-offs for the two different architectures. We can see, that the adversarial examples (crafted by using the Deep Fool method) require increased levels of perturbations in order to successfully attack the retrofitted models which have higher jump values. Of course, the user needs to decide how much accuracy on clean data he is willing to sacrifice in order to buy more robustness. However, this sacrifice is standard to most robustification strategies. For instance, for adversarial training one must choose the ratio between clean and adversarial examples used for training, where a higher ratio of adversarial examples improves the robustness while decreasing the predictive accuracy.

Accuracy vs Number of Iterations

Iterative attack methods can be computational demanding if a large number of iterations is required to craft strong adversarial examples. Of course, it is an easy task to break any defense with unlimited time and computational resources. However, it is the aim of an attacker to design efficient attack strategies (*i.e.*, fast generation of examples which have minimal perturbations), while the defender aims to make models

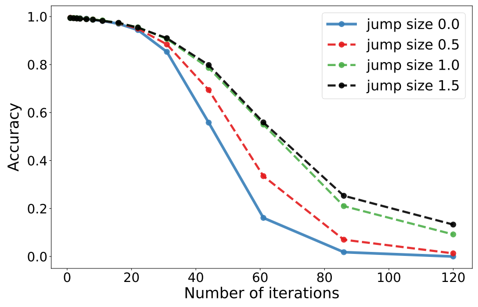


(a) Wide ResNet (CIFAR10)

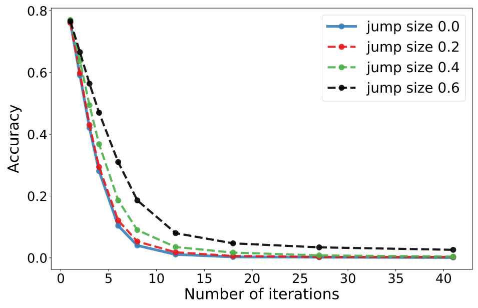


(b) MobileNetV2 (CIFAR10)

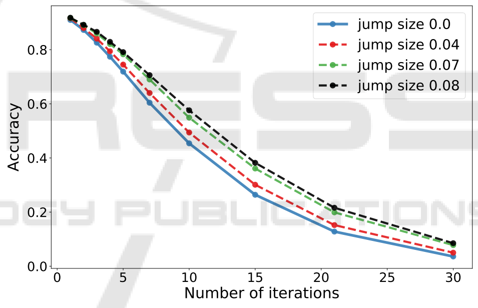
Figure 8: JumpReLU performance trade-offs for MNIST and CIFAR10. The left axis shows the average predictive accuracy for clean examples for varying values of the JumpReLU threshold. The right axis shows the minimum level of perturbation required to construct adversarial examples which achieve a nearly 100% fooling rate.



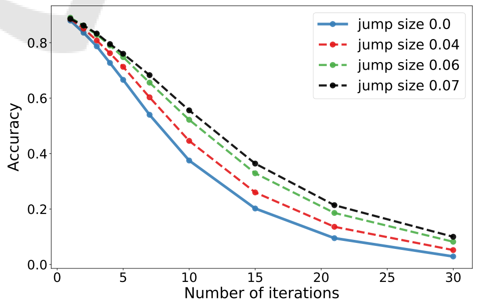
(a) LeNet like network (MNIST).



(b) AlexLike (CIFAR10).



(c) Robust Wide ResNet (CIFAR10).



(d) Robust MobileNetV2 (CIFAR10).

Figure 9: Strength of the PGD attack for increasing numbers of iterations. The PGD method requires a large number of iterations to craft strong adversarial examples. The JumpReLU increases the model robustness, *i.e.*, the fooling rate is reduced for a fixed number of iterations.

more robust to these attacks (*i.e.*, force the attacker to increase the average minimal perturbations which are needed to fool the model).

Figure 9 contextualizes the accuracy vs the number of iterations for the PGD attack. Attacking the retrofitted model requires a larger number of iterations in order to achieve the same fool rate as for the unprotected network. This is important, because a large number of iterations requires more computational resources as well as increases the computational time. To put the numbers into perspective, it takes about 4 minutes to run 7 iterations to attack the unprotected wide ResNet. In contrast, it takes about 5 minutes to run 7 steps to attack the retrofitted model.