STOCHASTIC ACTIVATION PRUNING FOR ROBUST ADVERSARIAL DEFENSE

Anonymous authors
Paper under double-blind review

ABSTRACT

Following recent work, neural networks are widely-known to be vulnerable to adversarial examples. Carefully chosen perturbations to real images, while imperceptible to humans, induce misclassification, threatening the reliability of deep learning in the wild. To guard against adversarial examples, we take inspiration from game theory and cast the problem as a minimax zero-sum game between the adversary and the model. In general, in such settings, optimal policies are *stochastic*. We propose *stochastic activation pruning* (SAP), an algorithm that prunes a random subset of activations, scaling up the survivors to compensate. The algorithm preferentially keeps activations with larger magnitudes. SAP can be applied to pre-trained neural networks, even adversarially trained models, without fine-tuning, providing robustness against adversarial examples. Experiments demonstrate that in the adversarial setting, SAP confers robustness, increasing accuracy and preserving calibration.

1 INTRODUCTION

While deep neural networks have gained nearly uncontested primacy in supervised learning, they remain vulnerable to adversarial examples (Szegedy et al., 2013). Small, carefully chosen perturbations to input data can induce misclassification with high probability. In the image domain, even perturbations so small as to be imperceptible to humans can fool powerful convolutional neural networks (Szegedy et al., 2013; Goodfellow et al., 2014). This fragility presents an obstacle to using machine learning in the wild. For example, a vision system vulnerable to adversarial examples might be fundamentally unsuitable for a computer security application. Even if a vision system isn't explicitly used for security, these weaknesses might be critical. Moreover these problems seem unnecessary. If these perturbations are not perceptible to people, why should they fool a machine?

Since this problem was first identified, several papers have investigated techniques both for generating and guarding against adversarial attacks. One way of generating adversarial examples was introduced in Goodfellow et al. (2014), called the "fast gradient sign method". To produce an adversarial example, we take one step in the direction of the sign of the gradient of the loss with respect to the inputs.

Many papers attempt to defend against adversarial examples by training the neural network on adversarial examples, either using the same model as in Goodfellow et al. (2014), Madry et al. (2017), or using an ensemble of models as in Tramèr et al. (2017). Taking a different approach, Nayebi & Ganguli (2017) draw inspiration from biological systems. They propose that to harden neural networks against adversarial examples one should learn flat, compressed representations that are sensitive to a minimal number of input dimensions.

In this paper, we introduce a method for guarding pre-trained networks, even adversarially trained models, against adversarial examples. During the forward pass, we stochastically prune a subset of the activations in each layer, preferentially retaining activations with larger magnitudes. Following the pruning, we scale up the surviving activations to normalize the dynamic range of the inputs to the subsequent layer. Unlike other adversarial defense methods, our method can be applied post-hoc to already-trained networks and requires no additional fine-tuning. The method could be applied to any existing neural network, rendering it more robust to adversarial examples.

2 PRELIMINARIES

We denote an n-layered deep neural network $h : \mathcal{X} \to Y$ as a chain of functions $h = h_n \circ h^{n-1} \circ \ldots \circ h^1$ where $h^i \in \mathcal{F}, \forall i \in \{1, \ldots, n\}$. In this work, we consider $\mathcal{F}$ to be a set of nonlinear functions ϕ, e.g. *Sigmoid, ReLU, MaxPool, etc.*, applied to a linear transformation of the input, $i.e.$ $\mathcal{F}(z) = \{\phi(Wz) \mid \forall \phi, \forall W\}$ and denote $h^i(h^{i-1}) := \phi^i(W^i h^{i-1})$. Given a set of nonlinearities and weight matrices, a deep neural network provides a nonlinear mapping from inputs x in an input space $\mathcal{X}$ to outputs $\hat{y}$ in an output space Y, i.e.

$$\hat{y} := h(x) = \phi^n(W^n \phi^{n-1}(W^{n-1} \phi^{n-2}(\ldots \phi^1(W^1 x))))$$

In supervised classification and regression problems, we are given a data set $\mathcal{D}$ of pairs (x, y), where each pair is drawn from an unknown joint distribution. For classification, y is a categorical variable, and for regression, y is a real-valued vector. Given a dataset, network architecture, and a loss function $J(y, \hat{y})$ (e.g. cross entropy, $L2$-loss), a learning training algorithm produces a set of learned parameters $\theta := \{W^i\}_{i=1}^n$. We denote $J(\theta, x, y)$ as the loss of a learned network parameterized by θ on a data point (x, y). While we apply our methods more broadly, for simplicity in notation, we focus on classification now for introducing the method.

Consider an input x that is correctly classified by neural network h. An adversary seeks to apply a small perturbation to x, $x^{adv} = x + \Delta x$, such that $h(x) \neq h(x^{adv})$. By *small* we mean that the perturbation should be minimally perceptible to a human. For perturbations applied to images, the l_∞-norm is considered a better measure of human perceptibility than more common norms like l_2, and has been studied as a natural notion of such perturbations, as put forward in Goodfellow et al. (2014). We assume that the adversary can access our neural network h while the manipulative power of adversary, the perturbation Δx, is of bounded norm $\|\Delta x\|_\infty \leq \lambda$.

Given a classifier, one common way to generate an adversarial example is to perturb the input in the direction that increases the cross-entropy loss. This is equivalent to minimizing the probability assigned to the true label. Given the neural network h, network parameters θ, input data x, and corresponding true output y, the adversary determines the perturbation Δx as follows

$$\Delta x = \arg\max_{r \sim \rho} J(\theta, x + r, y) \tag{1}$$

where ρ is the adversary's policy, a deterministic mapping from input x to the space of bounded (allowed) perturbations R ($\leq \lambda$). Due to nonlinearity of underlying neural network, and therefore that of the objective function J, solving the optimization Eq. 1 is difficult. Following Madry et al. (2017); Goodfellow et al. (2014), we use the Taylor expansion of the loss function up to the first order term and we get

$$\Delta x = \arg\max_{r \sim \rho} [J(\theta, x, y) + r^\top \mathcal{J}(\theta, x, y)] \qquad \text{where } \mathcal{J} = \frac{\partial J}{\partial x}$$

The first term in the optimization does not depend on the adversary's policy, therefore

$$\Delta x = \arg\max_{r \sim \rho} r^\top \mathcal{J}(\theta, x, y)$$

Given that input, label and a fixed model, the adversary finds a deterministic policy and chooses r to be in the direction of $\mathcal{J}(\theta, x, y)$, i.e. $\Delta x = \lambda \text{sign}(\mathcal{J}(\theta, x, y))$. This is the "fast gradient sign method" introduced by Goodfellow et al. (2014). Note that these methods of forming adversarial examples assume that the adversary has access to the model, i.e., the adversary can compute $\mathcal{J}(\theta, x, y)$, which is the gradient of the loss function with respect to the input data x.

3 DEFENSE AGAINST ADVERSARIAL ATTACK

Considering the defense problem from a game-theoretic perspective, the defender aims to change the model in order to minimize its maximum possible loss against the best policy of the adversary. Therefore, we can rewrite the Eq. 1 as follows:

$$\pi^* \ , \ \rho^* := \arg\min_{p \sim \pi} \max_{r \sim \rho} \mathbb{E}_{\pi, \rho} \left[J(M_p(\theta), x + r, y) \right] \tag{2}$$

where the adversary tries to maximize the loss of the defender by perturbing the input under a stochastic strategy ρ and the defender tries to minimize the loss by changing model parameters θ to a new $M_p(\theta)$ under stochastic strategy π. The optimization problem in Eq. 2 is a *minimax* zero-sum game between the adversary and defender where the optimal strategies $(\pi^*\ ,\ \rho^*)$, in general, are mixed Nash equilibrium, i.e. stochastic policies. Recall that the set of policies for the adversary is limited to perturbations with constrained l_∞ norm. For the defender, the policy set is constraint to those policies which preserve the overall accuracy of the model.

If the adversary perturbs examples by using the fast gradient sign method, then a defender might deceive the adversary by using a stochastic policy. In this case, the adversary could only estimate the gradient. Once the model decides on a policy π, the adversary computes the $\mathbb{E}_{p\sim\pi}\left[\mathcal{J}(M_p(\theta), x, y)\right]$, the best response strategy, and comes up with a perturbed image.

In this paper, we propose Stochastic Activation Pruning (SAP). Intuitively, the idea is to stochastically drop out nodes in each layer during forward propagation. We retain nodes with probabilities proportional to the magnitude of their activation and scale up the surviving nodes to preserve the dynamic range of the activations in each layer. Empirically, the approach preserves the accuracy of the original model. Notably, the method can be applied post-hoc to already-trained models.

Formally, assume a given pre-trained model, with activation layers (*ReLU, Sigmoid, etc.*) and input pair of (x, y). For each of those layers, SAP converts the activation map to a multinomial distribution, choosing each activation with a probability proportional to its absolute value. In other words, we obtain the multinomial distribution of each activation layer by L_1 projection of activation values on a simplex. Given the i'th layer activation map, $h^i \in \mathbb{R}^{m^i}$, the probability of sampling the j'th activation with value $(h^i)_j$ is given by

$$p^i_j = \frac{|(h^i)_j|}{\sum_{k=1}^{m^i} |(h^i)_k|}$$

Random samples are drawn from the activation map given the probability distribution described above. Samples are drawn with replacement. This makes it convenient to determine whether a sample would be sampled at all. If an activation is sampled, we scale it up by the inverse of the probability of sampling it over all the draws. If not, we set the activation to 0. In this way, SAP preserves inverse propensity scoring of each activation.

Under an instance p of policy π, we draw r^i_p samples with replacement from this multinomial distribution. The new activation map, $M_p(h^i)$ is given by

$$M_p(h^i) = h^i \odot m^i_p \qquad\qquad (m^i_p)_j = \frac{\mathbb{I}((h^i)_j)}{1-(1-p^i_j)^{r^i_p}}$$

where $\mathbb{I}((h^i)_j)$ is the indicator function that returns 1 if $(h^i)_j$ was sampled at least once, and 0 otherwise. The algorithm is described in Algorithm 1. In this way, the model parameters are changed from θ to $M_p(\theta)$, for instance of p under policy π, while the reweighing $1-(1-p^i_j)^{r^i_p}$ preserves $\mathbb{E}_{p\sim\pi}[M_p(h^i)_j] = (h^i)_j$. If the model was linear, the proposed pruning method would behave the same way as the original model in expectation. But even with the non-linearity, SAP would do similar to the un-pruned model in expectation. This prompts us to test out the model on pre-trained models, without any fine-tuning.

3.1 Advantage against adversarial attack

We attempt to explain the advantages of SAP under the assumption that we are applying it to a pre-trained model that achieves high generalization accuracy. For instance p under policy π, if the number of samples drawn for each layer i, r^i_p, is large, then fewer parameters of the neural network are pruned, and the scaling factor gets closer to 1. Under this scenario, the stochastically pruned model performs almost identically to the original model. The stochasticity is not advantageous in this case, but there is no loss in accuracy in the pruned model as compared to the original model.

On the other hand, with fewer samples in each layer, r^i_p, a large number of parameters of the neural network are pruned. Under this scenario, the SAP model's accuracy will drop compared to the original model's accuracy. But this model is stochastic and has more freedom to deceive the adversary while

Algorithm 1 Stochastic Activation Pruning (SAP)

Input: Model h, input data x
Set $h^0 = x$
for i in $1{:}n$ **do**
  Output for layer i, $h^i \leftarrow \phi^i(W^i h^{i-1})$, where $h^i \in \mathbb{R}^{m^i}$
  Form probability distribution, p^i, over h^i, where $p^i_j \leftarrow \frac{|(h^i)_j|}{\sum_{k=1}^{m^i} |(h^i)_k|}$
  Number of samples to be drawn, $r^i \in \mathbb{N}$
  Set $S = \{\}$
  for j in $1{:}r^i$ **do**
    Draw a sample s from h^i with probability distribution p^i
    $S = S \cup \{s\}$
  end for
  for j in $1{:}m^i$ **do**
    if $(h^i)_j \in S$ **then**
      $(h^i)_j \leftarrow \frac{(h^i)_j}{1-(1-p^i_j)^{r^i}}$
    else
      $(h^i)_j \leftarrow 0$
    end if
  end for
end for
$\hat{y} \leftarrow h^n$

return $\hat{y}$

it perturbs the input data. So the advantage of SAP comes if we can balance the number of samples drawn in a way that negligibly affects the original model's accuracy while conferring robustness against adversarial examples.

SAP is similar to the *dropout* technique due to Srivastava et al. (2014). However, there is a crucial difference. SAP is more likely to sample activations that are high in absolute value, whereas dropout samples each activation with the same probability. Because of this difference, SAP, unlike dropout, can be applied post-hoc *to pre-trained models* without significantly decreasing the accuracy of the model. Experiments comparing SAP and Dropout are included in section 5. Interestingly, dropout confers little advantage over the baseline. We suspect that the reason for this is that the dropout training procedure encourages all possible dropout masks to result in similar mappings.

4 ADVERSARIAL ATTACK ON SAP MODEL

If the adversary knows that our defense policy is to apply SAP, they might try to calculate the best strategy against the SAP model. Given the neural network h, input data x, corresponding true output y, a policy ρ over the allowed perturbations, and a policy π over the model parameters that come from SAP (this result holds true for any stochastic policy chosen over the model parameters), the adversary determines the perturbation Δx as follows

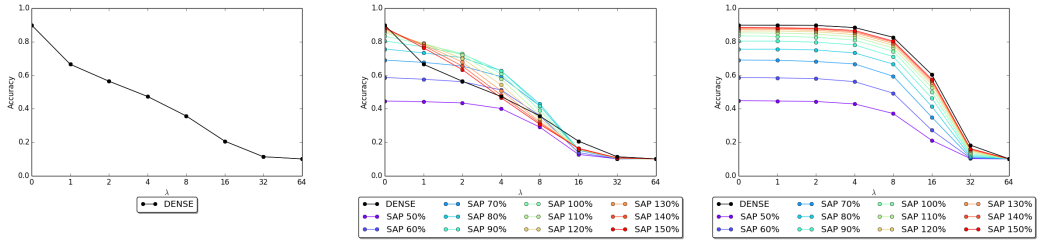
$$\Delta x = \arg\max_{r \sim \rho} \mathbb{E}_{p \sim \pi}[J(M_p(\theta), x + r, y)]$$

Using the result from section 2

$$\Delta x = \arg\max_{r \sim \rho} r^\top \mathbb{E}_{p \sim \pi}[\mathcal{J}(M_p(\theta), x, y)]$$

To maximize the term, the adversary will set r to be in the direction of $\mathbb{E}_{p \sim \pi}[\mathcal{J}(M_p(\theta), x, y)]$.

Analytically computing $\mathbb{E}_{p \sim \pi}[\mathcal{J}(M_p(\theta), x, y)]$ is not feasible. However, the adversary can use Monte Carlo sampling to estimate the expectation as $\widetilde{\mathcal{J}}(M_p(\theta), x, y)$. Then, using the "fast gradient sign method", $\Delta x = \lambda \text{sign}(\widetilde{\mathcal{J}}(M_p(\theta), x, y))$.



(a) (b) (c)

Figure 1: Accuracy-λ plots. (a) Dense model with adversarial examples generated on the dense model. (b) SAP models, tested against adversarial examples generated via MC. (c) SAP models, tested against randomly perturbed images. The legend indicates number of samples drawn.

5 EXPERIMENTS

Our experiments to evaluate SAP address two tasks: image classification and reinforcement learning. The method was applied to the *ReLU* activation maps of pre-trained neural networks. The perturbations were generated using the "fast gradient sign method", $\Delta x = \lambda \text{sign}(\mathcal{J}(M_p(\theta), x, y))$. For stochastic models, $\mathcal{J}(M_p(\theta), x, y)$ was estimated using Monte Carlo sampling, unless otherwise mentioned. In our experiments, only images are being perturbed. So the relative perturbation with respect to the data would be $\frac{\lambda}{256}$. And since the images need to be valid images even after perturbation, the image after perturbation had integer values and were clipped to be in the range $[0, 255]$. For simplicity, $\lambda = \{0, 1, 2, 4, 8, 16, 32, 64\}$ were considered.

For the evaluation of models in the image classification domain, we look at two aspects: the model accuracy for varying values of λ, and the calibration of the models. In the case of accuracy, the higher the accuracy is the better the model is considered to be. For stochastic models, the accuracy is computed over an average of forward passes. In the case of the model calibration, linear calibration is ideal, as it suggests that the model gets an accuracy that is proportional to the confidence level of its output.

For the evaluation of models in the reinforcement learning domain, we look at the average score the model achieves on the games played, for varying values of λ. The higher the score, the better is the model's performance. Since the reward values hold no meaning, we will report the percentage relative changes in rewards.

5.1 ADVERSARIAL ATTACKS IN IMAGE CLASSIFICATION

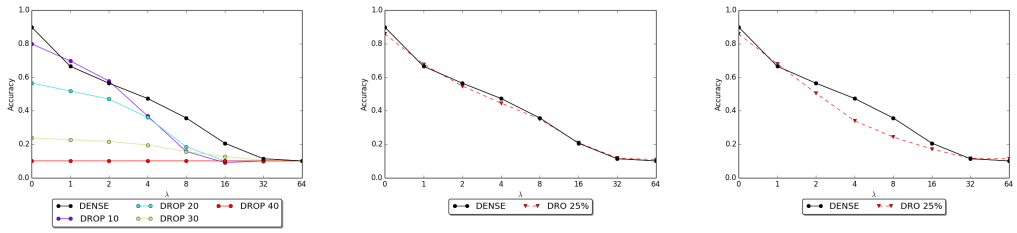
The Cifar-10 dataset was used for the image classification domain. A ResNet-20 model (He et al., 2016) was trained using SGD, with minibatch size of 512, momentum of 0.9, weight decay of 0.0001, and a learning rate of 0.5 for the first 100 epochs, then 0.05 for the next 30 epochs, and then 0.005 for the next 20 epochs. This achieved an accuracy of 89.8% with cross-entropy loss and *ReLU* non-linearity. For all the figures in this section, this model is referenced to as the "dense" model.

The accuracy of the dense model degrades quickly with λ. For $\lambda = 1$, the accuracy drops down to 66.3%, and for $\lambda = 2$ it is 56.4%. These are small (hardly perceptible) perturbations in the input images, but the dense model's accuracy decreases significantly. This is illustrated in Fig. 1(a).

5.1.1 STOCHASTIC ACTIVATION PRUNING (SAP)

We apply SAP to the dense model. For each activation map $h^i \in \mathbb{R}^{m^i}$, we pick $k\%$ of m^i activations to keep. Since activations are sampled with replacement, k can be more than 100. We will refer to k as the percentage of samples drawn.

Fig. 1(b) plots performance of these models against adversarial examples. With many samples drawn, SAP converges to the dense model. With few samples drawn, accuracy diminishes, even without perturbation. The plot explains this balance well. We achieve the best performance with $\sim 100\%$ samples picked. Fig. 1(c) plots performance of SAP models against examples perturbed with random



(a) (b) (c)

Figure 2: Accuracy-λ plots: (a) Dropout applied to pretrained model. (b) Model trained with 25% dropout, applied during training and testing. (c) Dropout 25% model, with dropout applied only in training. Dropout rates denoted in legend.

noise. Perturbations of size $\lambda = 64$ are readily perceptible and push all models under consideration to near-random outputs, so we focus our attention on smaller values of λ.

We will now only look at SAP 100% (SAP-100). Against adversarial examples, with $\lambda = 1, 2$ and 4, we observe a $12.2\%, 16.3\%$ and 12.8% absolute increase in accuracy respectively. However, for $\lambda = 0$, we observe a 6.5% absolute decrease in accuracy. For $\lambda = 16$ again, there is a 5.2% absolute decrease in accuracy.

5.1.2 DROPOUT (DRO)

Dropout, a technique due to Srivastava et al. (2014), was also tested to compare with SAP. Similar to the SAP setting, this method was added to the *ReLU* activation maps of the Dense model. We see that low dropout rate perform similar to the Dense model for small λ values, but its accuracy starts decreasing very quickly for higher λ values. This is illustrated in Fig. 2(a).

We also trained a ResNet-20 model, similar to the Dense model, but with a dropout rate of 25%. This achieved an accuracy of 85.9% under the no perturbation case. The model does similar to the Dense model against adversarial examples generated on itself. For validation with dropout, the accuracy was 67.8% and 54.8% for $\lambda = 1$ and 2 respectively. For validation without dropout, the accuracy was 68.0% and 50.5% for $\lambda = 1$ and 2 respectively. Fig. 2 (b) and (c) illustrates these results.

5.1.3 ADVERSARIAL TRAINING (ADV)

Adversarial training (Goodfellow et al., 2014) has emerged a standard method for defending against adversarial examples. It has been adopted by Madry et al. (2017); Tramèr et al. (2017) to maintain high accuracy levels even for large λ values.

We trained a ResNet-20 using SGD, with minibatch size of 512, momentum of 0.9, weight decay of 0.0001, and a learning rate of 0.5 to begin with, which was halved every 10 epochs, for a total of 100 epochs. It was trained on a dataset consisting of 80% un-perturbed data and 20% adversarially perturbed data, generated on the model from the previous epoch, with $\lambda = 2$. This was done on the Cifar-10 dataset. This achieved an accuracy of 75.0% on the un-perturbed validation set, with cross-entropy loss and *ReLU* non-linearity. Note that the model capacity was not changed.

When tested against adversarial examples, the accuracy dropped to $72.9\%, 70.9\%$ and 67.5% for $\lambda = 1, 2$ and 4 respectively. We ran SAP-100 on the ADV model (referred to as ADV+SAP-100). The accuracy in the no perturbation case was 74.1%. For adversarial examples, both models act similar to each other for small values of λ. But for $\lambda = 16$ and 32, ADV+SAP-100 gets a higher accuracy than ADV by an absolute increase of 7.8% and 7.9% respectively.

We compare the accuracy-λ plot for dense, SAP-100, ADV and ADV+SAP-100 models. This is illustrated in Fig. 3 For smaller values of λ, SAP-100 achieves high accuracy. As λ gets larger, ADV+SAP-100 performs better than all the other models. We also compare the calibration plots for these models, in Fig. 4. The dense model is not linear for any $\lambda \neq 0$. The other models are well calibrated (close to linear), and behave similar to each other for $\lambda \leq 4$. For higher values of λ, we see that ADV+SAP-100 is the closest to a linearly calibrated model.

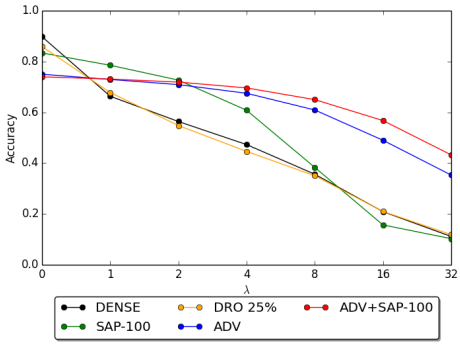


Figure 3: Accuracy-λ plots for dense, SAP-100, ADV and ADV+SAP-100 models.

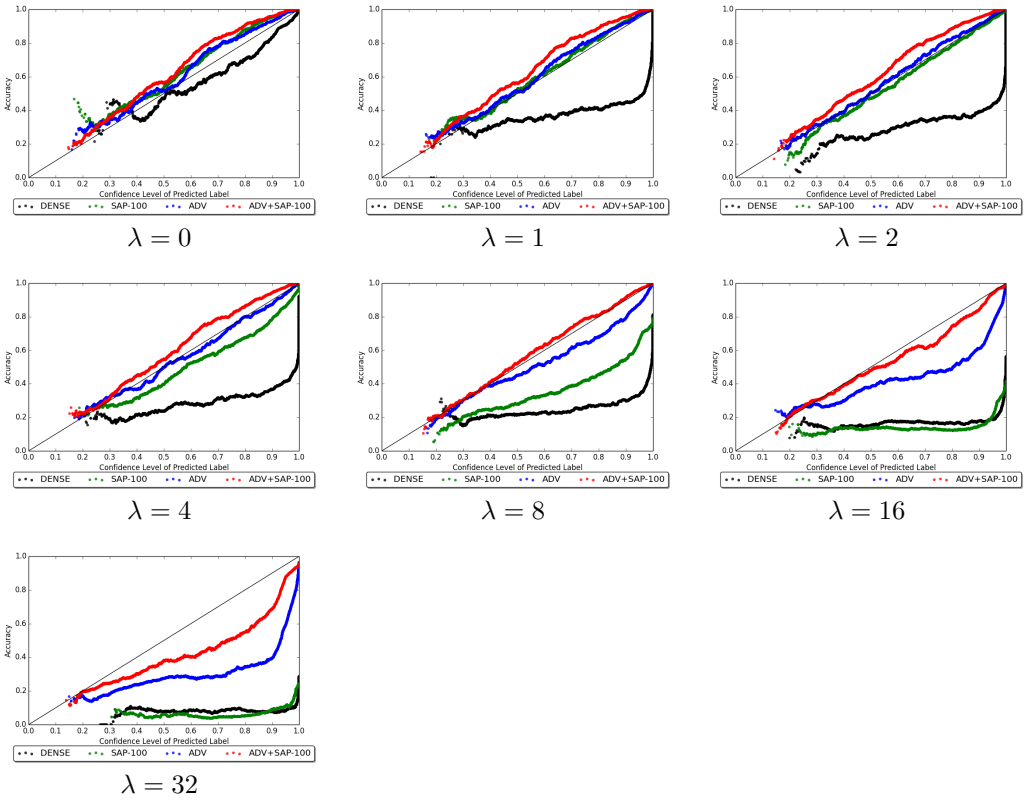


Figure 4: Calibration plots for dense, SAP-100, ADV and ADV+SAP-100 models.

5.2 ADVERSARIAL ATTACKS IN DEEP REINFORCEMENT LEARNING (RL)

Previously, (Behzadan & Munir, 2017; Huang et al., 2017; Kos & Song, 2017) have shown that the Reinforcement learning agents can also be easily manipulated by adversarial examples. The RL agent learns the long term value $Q(a, s)$ of each state-action pair (s, a) through interaction with an environment where given a state s, the optimal action is $\arg\max_a Q(a, s)$. A regression based algorithm, Deep Q-Network (DQN)(Mnih et al., 2015) and its improved version, Double DQN (DDQN) have been proposed for the popular Atari games (Bellemare et al., 2013) as benchmarks. We deploy DDQN algorithm and train an RL agent in variety of different Atari game settings.

Table 1: Percentage relative increase in rewards gained for SAP-100 compared to original model while playing different Atari games.

λ	**Assault**	**Asterix**	**BankHeist**	**BattleZone**	**BeamRider**	**Bowling**
0	-12.2	-33.4	-59.2	-65.8	-15.8	-4.5
1	10.4	13.3	131.7	-22.0	164.5	3425.9
2	9.8	20.8	204.8	110.1	92.3	
4	12.4	14.0	1760.0	202.6		
8	16.6	7.4	60.9	134.8		

Similar to the image classification experiments, we tested SAP on a pre-trained model (the model is described in the Appendix section A), by applying the method on the *ReLU* activation maps. SAP-100 was used for these experiments. Table 1 specifies the percentage relative increase in rewards of SAP-100 as compared to the original model. For all the games, we observe a drop in performance for the no perturbation case. But for $\lambda \neq 0$, the relative increase in rewards is positive (except for in the BattleZone game), which is very high in some cases (3425.9% for $\lambda = 1$ for the Bowling game).

6 Related work

Recently, the paradigm of adversarial attacks and defenses to machine learning models have been addressed as a serious problem in the field of machine learning where creating such attacks and finding ways to defend against them are the essential core this paradigm. Goodfellow et al. (2014) introduced the fast gradient sign method, which involves taking a step in the direction of the gradient of the loss function with respect to the input data. Kurakin et al. (2016) proposed an iterative method where fast gradient sign method is used for smaller step sizes, which leads to a better approximation of the gradient. Papernot et al. (2017) observed that adversarial examples could be transferred to other models as well. Madry et al. (2017) introduce adding random noise to the image and then using the fast gradient sign method to come up with adversarial examples.

Being robust against adversarial examples has primarily focused on training on the adversarial examples. Goodfellow et al. (2014) use the fast gradient sign method and inject data perturbed in that way into their training dataset. Madry et al. (2017) use an iterative fast gradient sign method approach of creating adversarial examples to train on. Madry et al. (2017) introduced an ensemble adversarial training method of training on the adversarial examples created on the model itself and an ensemble of other pre-trained models. These works have been successful, achieving only a small drop in accuracy form the clean and adversarially generated data.

Other work on making models robust against adversarial examples involve training a new model, with a new objective function. One such example is the technique introduced due to Nayebi & Ganguli (2017), where the activation are encouraged to be saturated. In doing so, small perturbations in the input data does not effect the output much.

7 Discussion

We introduced SAP, demonstrating that it confers significant improvements in the robustness of pre-trained models against adversarial examples. The approach guards networks without requiring any additional training. In the image classification domain, we showed that in the adversarial setting, applying SAP improves both the accuracy of the model and its calibration, achieving close to linear calibration for small perturbations and outperforming other models. Applying SAP is enough to be robust against small perturbation values. For larger perturbations, even SAP models suffer, but they compare favorably to their dense counterparts. Notably, SAP can be combined with adversarial training, compounding the respective benefits of each approach. Even in the domain of reinforcement learning, SAP is able to defend against adversarial examples better than the original model.

REFERENCES

Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. *arXiv preprint arXiv:1701.04143*, 2017.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

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

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

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

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Aran Nayebi and Surya Ganguli. Biologically inspired protection of deep networks from adversarial attacks. *arXiv preprint arXiv:1703.09202*, 2017.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 506–519. ACM, 2017.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

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

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

A REINFORCEMENT LEARNING MODEL ARCHITECTURE

For the experiments run in section 5.2, we trained the network with RMSProp, minibatches of size 32, a learning rate of 0.00025, and a momentum of 0.95 and as in (Mnih et al., 2015) where the discount factor is $\gamma = 0.99$, the number of steps between target updates to 10000 steps. We updated the network every 4 steps by randomly sampling a minibatch of size 32 samples from the replay buffer and trained the agents for a total of $100M$ steps per game. The experience replay contains the $1M$ most recent transitions. For training we used an ε-greedy policy with ε annealed linearly from 1 to 0.1 over the first $1M$ time steps and fixed at 0.1 thereafter.

The input to the network is $4 \times 84 \times 84$ tensor with a rescaled, gray-scale version of the last four observations. The first convolution layer has 32 filters of size 8 with a stride of 4. The second convolution layer has 64 filters of size 4 with stride 2. The last convolution layer has 64 filters of size 3 followed by two fully connected layers with size 512 and the final fully connected layer Q-value of each action where *ReLU* rectifier is deployed for the nonlinearity at each layer.

B OTHER METHODS ON CIFAR-10

We tried a variety of different methods that could be added to pre-trained models and check their performance against adversarial examples. The following is a continuation of section 5.1, where we use the dense model again on the Cifar-10 dataset.

B.1 DETERMINISTIC WEIGHT PRUNING (DWP)

Following from the motivation of preventing perturbations to propagate forward in the network, we tested deterministic weight pruning, where the top $k\%$ entries of a weight matrix were kept, while the rest were pruned to 0, according to their absolute values. This method was prompted by the success achieved by this pruning method, introduced by Han et al. (2015), where they also fine-tuned the model.

For low levels of pruning, these models do very similar to the dense model, even against adversarial examples. The adversary can compute the gradient of the sparse model, and the perturbations propagate forward through the un-pruned weights. For larger levels of sparsity, the accuracy in the no perturbation case drops down quickly. Fig. 5 illustrates this.

B.2 STOCHASTIC WEIGHT PRUNING (SWP)

Observing the failure of deterministic weight pruning, we tested a mix of stochasticity and pruning, the stochastic weight pruning method. Very similar to the idea of SAP, we consider all the entries of a weight matrix to be a multinomial distribution, and we sample from it with replacement. For a weight matrix $W^i \in \mathbb{R}^{m^i}$, we sample from it r^i times with replacement. The probability of sampling $(W^i)_j$ is given by

$$p^i_j = \frac{|(W^i)_j|}{\sum_{k=1}^{m^i} |(W^i))_k|}$$

The new weight entry, $M(W^i)_j$, is given by

$$M(W^i)_j = W^i_j \frac{\mathbb{I}(W^i_j)}{1-(1-p_i)^{r^i}}$$

where $\mathbb{I}(W^i_j)$ is the indicator function that returns 1 if W^i_j was sampled at least once, and 0 otherwise.

For these experiments, for each weight matrix $W^i \in \mathbb{R}^{m^i}$, the number of samples picked were $k\%$ of m^i. Since samples were picked with replacement, k could be more than 100. We will refer to k as the percentage of samples drawn.

These models behave very similar to the Dense model. A range of percentage of samples drawn were tried, but no evident robustness could be seen against adversarial examples. Fig. 6 illustrates this.

B.3 RANDOM NOISY ACTIVATIONS (RNA)

Next we change our attention to the activation maps in the Dense model. One simple way of introducing stochasticity to the activations is by adding random gaussian noise to each activation entry, with mean 0 and constant standard deviation, s. So each activation map h^i now changes to $M(h^i)$, where the j'th entry is given by

$$M(h^i)_j = (h^i)_j + \eta \qquad\qquad \eta \sim \mathcal{N}(0, s^2)$$

These models too do not offer any robustness against adversarial examples. Their accuracy drops quickly with λ and s. Fig. 7 illustrates this.

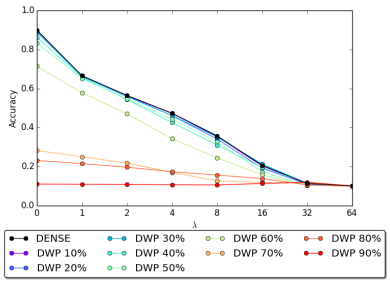


Figure 5: Accuracy-λ plot for deterministic weight pruning models.

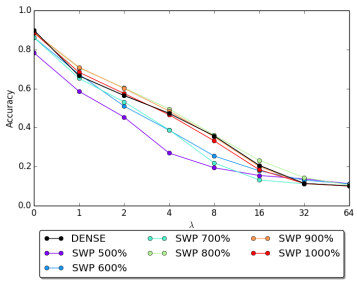


Figure 6: Accuracy-λ plot for stochastic weight pruning models.

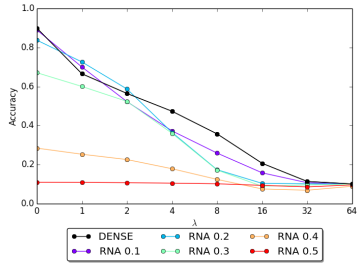


Figure 7: Accuracy-λ plot for random noisy activations models.

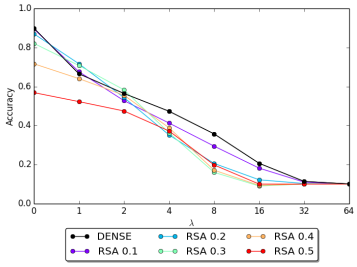


Figure 8: Accuracy-λ plot for randomly scaled activations models.

B.4 Randomly scaled activations (RSA)

Instead of having additive noise, we can also make the model stochastic by scaling the activations. The scale factor can be picked from a gaussian distribution, with mean 1 and constant standard deviation s. So each activation map h^i now changes to $M(h^i)$, where the j'th entry is given by

$$M(h^i)_j = \eta(h^i)_j \qquad \eta \sim \mathcal{N}(0, s^2)$$

These models perform similar to the Dense model, but again, no robustness is offered against adversarial examples. Fig. 8 illustrates this.