

# k-WTA Activation Function

Marco Costa, Andrea Luca Antonini

25 giugno 2020

## 1 Introduction

The purpose of this project is to implement a Neural Network in which we use k-WTA as activation function.

The use of k-WTA activation is motivated for defending against gradient-based adversarial attacks. In fact, provided a labeled data item  $(\mathbf{x}, y)$ , the attacker finds a perturbation  $\mathbf{x}'$  imperceptibly similar to  $\mathbf{x}$  but misleading enough to cause the network to output a label different from  $y$ . The most effective way to find such a perturbation, i.e. adversarial example, is by exploiting the gradient information of the network w.r.t. its input  $\mathbf{x}$ .

k-WTA activation function takes as input the entire output of a layer, retains its  $k$  largest values and deactivates all others to zero. If we use  $f(\mathbf{x}; \mathbf{w})$  to denote a k-WTA network taking an input  $\mathbf{x}$  and parameterized by weights  $\mathbf{w}$ , then the gradient  $\nabla_{\mathbf{x}} f(\mathbf{x}; \mathbf{w})$  at certain  $\mathbf{x}$  is undefined. Therefore,  $f(\mathbf{x}; \mathbf{w})$  is  $C^0$  discontinuous. The discontinuities in the activation function also renders  $f(\mathbf{x}; \mathbf{w})$  discontinuous w.r.t.  $\mathbf{w}$ , but these discontinuities are rather sparse in the space of  $\mathbf{w}$ , thus the network can be trained successfully.

Formally, k-WTA retains the  $k$  largest values of a  $N \times 1$  input vector and sets all others to be zero before feeding the vector to the next network layer:

$$\phi_k(\mathbf{y})_j = \begin{cases} y_j, & \text{if } y_j \in \{k \text{ largest element of } \mathbf{y}\} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where  $\phi_k : \mathbb{R}^N \rightarrow \mathbb{R}^N$  is the k-WTA function (parameterized by an integer  $k$ ),  $\mathbf{y} \in \mathbb{R}^N$  is the input to the activation, and  $\phi_k(\mathbf{y})_j$  denotes the  $j$ -th element of the output  $\phi_k(\mathbf{y})$ .

Instead of specifying  $k$ , we introduce a parameter  $\gamma \in \{0, 1\}$  called *sparsity ratio*. If a layer has an output dimension  $N$ , then its k-WTA activation has  $k = \lfloor \gamma \cdot N \rfloor$ . Even though the sparsity ratio can be set differently for different layers, in practice there is no gain from introducing such a variation, so we use a fixed  $\gamma$ .

In a Convolutional Neural Network (CNN), the output of a layer is a  $C \times H \times W$  tensor: in this case we will treat the tensor as a  $C \cdot H \cdot W \times 1$  vector input to the k-WTA activation function.

The runtime of computing a k-WTA activation is asymptotically  $O(N)$ , because finding  $k$  largest value in a list corresponds to finding the  $k$ -th largest value, which has  $O(N)$  complexity.

Concerning the training of k-WTA networks, we know that when the sparsity ratio  $\gamma$  is relatively small ( $\leq 0.2$ ), the network training converges slowly. In fact,

a smaller  $\gamma$  activates fewer neurons, effectively reducing more of the layer width and, therefore, the network size. Nevertheless, we prefer a smaller  $\gamma$  because it usually leads to better robustness against finding adversarial examples.

Theoretically speaking, consider one layer outputting values  $\mathbf{x}$ , passed through a k-WTA activation, and followed by the next layer whose linear weight matrix is  $W$ . We define the k-WTA *activation pattern* under the input  $\mathbf{x}$  as:

$$\mathcal{A}(\mathbf{x}) := \{i \in [l] \mid x_i \text{ is one of the } k \text{ largest values in } \mathbf{x}\} \subseteq [l]$$

where we use  $[l]$  to indicate integer set  $\{1, \dots, l\}$ .

Even an infinitesimal perturbation of  $\mathbf{x}$  can change  $\mathcal{A}(\mathbf{x})$ : some element  $i$  is removed from  $\mathcal{A}(\mathbf{x})$ , while another element  $j$  is added in. Corresponding to this change, in the evaluation of  $W\phi_k(\mathbf{x})$ , the contribution of  $W$ 's column vector  $W_i$  is useless, while another column vector  $W_j$  suddenly takes effect. It's this change that makes the result of  $W\phi_k(\mathbf{x})$   $C^0$  discontinuous.

Once  $W$  is determined, the discontinuity jump depends on the value of  $x_i$  and  $x_j$ , that are respectively the input value that does no more affect the result and the new value that actually has effect. We note from previous experiments that the smaller  $\gamma$  is, the larger the discontinuity jump will be: for this reason, we prefer a smaller  $\gamma$ , that will make the search of adversarial examples harder.

If the activation patterns of all layers are fixed, then  $f(\mathbf{x})$  is a linear function, but when the activation pattern changes,  $f(\mathbf{x})$  switches from one linear function to another linear function. Over the entire space of  $\mathbf{x}$ ,  $f(\mathbf{x})$  is *piecewise linear*. The specific activation patterns of all layers define a specific linear piece of the function, i.e. a *linear region*: in k-WTA, these linear regions are disconnected. If the linear regions are densely distributed, a small perturbation  $\Delta\mathbf{x}$  from any data point will likely cross the boundary of the linear region where  $\mathbf{x}$  is. Whenever boundary crossing occurs, the gradient becomes undefined.

The rationale behind k-WTA activation is to destroy network gradients, that are information needed in *white box attacks*.

k-WTA is able to universally improve the white-box robustness, regardless of the training method. The k-WTA robustness under *black box attacks* is not always significantly better than other activation function networks (e.g. ReLU).

## 2 Approach

The core of our approach is to substitute the most commonly used activation functions (e.g. ReLU) with k-WTA and measure their performance and robustness to adversarial attacks. The idea is to compare the performance of the models with k-WTA with those who have a most common ReLU. In order to do this, we used two different networks, i.e. ResNet and AlexNet, trained on different dataset, i.e. CIFAR-10 and SVHN.

By examining the resulting accuracies on both the adversarial and normal testing sets, we wanted to confirm the effectiveness of the method.

## 3 Experiments

### 3.1 Frameworks and Data

All the experiments have been done with Pytorch deep learning framework, using GPU acceleration for the execution of the experiments.

As said before, we considered ResNet and AlexNet models using the standard pre-trained implementations of ResNet18 and AlexNet architecture from the Torchvision package. We used these networks in two ways. First, since these networks are pre-trained on ImageNet dataset, we modified the final layer to output 10 features and fine-tuned the models on CIFAR-10 and SVHN dataset. Second, we trained the networks from scratch, directly on CIFAR-10 and SVHN datasets.

### 3.2 Training models

We replaced the ReLU activation layer with k-WTA using a sparsity ratio  $\gamma = 0.2$  and trained the models for several epochs: 80 in the case of models trained from scratch, 10 in the case of pre-trained models. Furthermore, we decreased the sparsity ratio to  $\gamma = 0.1$  and trained the models again.

We noticed that, obviously, training the models from scratch is much more expensive in terms of time than training pre-trained models.

Our training dataset was preprocessed with random cropping and random horizontal flipping. Finally, we additionally scaled up images to 224x224 since the Torchvision pre-trained model architectures expect this input dimensionality.

We used a batch size of 32 in the data loaders.

All of our models used the Cross Entropy loss function and were trained with Stochastic Gradient Descent using a learning rate of 0.01 for pre-trained models, while for the others learning rate is set to 0.1 for the first 50 epochs, while for the remaining 30 is manually changed to 0.01.

Moreover, we inserted a momentum of 0.9.

### 3.3 Robustness to Adversarial attacks

k-WTA activation function, as explained in the introduction, is wanted to be a defence against attacks that exploit the gradient of the network. In order to neutralize these attacks, k-WTA, thanks to its densely piecewise continuous nature, destroys network gradients and, therefore, represents a counterpoison to the venom of attacks, specially white box attacks. Our goal is to test the effective robustness to adversarial attacks of the models using k-WTA activation function, comparing the results with models using ReLU activation function. To achieve this, we tested the various models ((Resnet | AlexNet) + (k-WTA | ReLU)) under two kind of attacks, namely Projected Gradient Descent (PGD) and Deepfool. Both the attacks are run using the Foolbox third-party toolkit. They use a perturbation  $\epsilon = 0.031$  in the case of CIFAR-10 dataset, while in the case of SVHN dataset  $\epsilon = 0.047$ . PGD attack is carried on for 40 steps, with random starting and relative step size of 0.003. Instead, Deepfool attack is run for 20 steps with a candidate sub-sampling parameter of 10. Due to computational constraints, each of these attacks was run only on the first 10 mini-batches, each of size 32, of images from the test set.

### 3.4 Results

In *Table 1* and *Table 2* we show the results for training an AlexNet network on SVHN and CIFAR-10 respectively, using ReLU, k-WTA with  $\gamma = 0.2$  and k-WTA with  $\gamma = 0.1$ . The value  $A_{std}$  indicates the standard test accuracy, while the other two columns show the robustness accuracy under PGD and Deepfool attacks. *Table 3* and *Table 4* illustrates the results with the networks with same characteristics, but with the difference that they are pre-trained.

Activation	$A_{std}$	PGD	Deepfool
ReLU	19.59%	20.31%	16.25%
k-WTA-0.2	94.65%	26.88%	1.25%
k-WTA-0.1	19.59%	20.94%	16.88%

Table 1: AlexNet experiment results with SVHN

Activation	$A_{std}$	PGD	Deepfool
ReLU	86.66%	48.44%	0.00%
k-WTA-0.2	87.22%	59.38%	0.00%
k-WTA-0.1	88.98%	52.19%	0.00%

Table 2: AlexNet experiment results with CIFAR-10

Activation	$A_{std}$	PGD	Deepfool
ReLU	92.09%	20.62%	0.00%
k-WTA-0.2	91.76%	20.62%	0.00%
k-WTA-0.1	91.71%	30.94%	0.00%

Table 3: Pre-trained AlexNet experiment results with SVHN

Activation	$A_{std}$	PGD	Deepfool
ReLU	87.95%	5.00%	0.00%
k-WTA-0.2	87.23%	15.31%	0.00%
k-WTA-0.1	83.67%	36.56%	0.00%

Table 4: Pre-trained AlexNet experiment results with CIFAR-10

Then, we show the experiment results for training normally and under adversarial attack using network ResNet18, with SVHN dataset (*Table 5*) and CIFAR-10 dataset (*Table 6*) on ReLU, k-WTA-0.2 and k-WTA-0.1.

Activation	$A_{std}$	PGD	Deepfool
ReLU	94.91%	28.44%	0.00%
k-WTA-0.2	95.82%	70.31%	13.12%
k-WTA-0.1	95.02%	77.81%	33.12%

Table 5: ResNet18 experiment results with SVHN

Activation	$A_{std}$	PGD	Deepfool
ReLU	92.29%	28.44%	0.00%
k-WTA-0.2	85.20%	68.75%	14.38%
k-WTA-0.1	79.90%	58.44%	26.56%

Table 6: ResNet18 experiment results with CIFAR-10

Again, we show the results for networks with the same characteristics as before, but in this case they are pre-trained.

Activation	$A_{std}$	PGD	Deepfool
ReLU	94.71%	44.06%	0.00%
k-WTA-0.2	94.64%	10.00%	36.88%
k-WTA-0.1	93.12%	6.56%	41.56%

Table 7: Pre-trained ResNet18 experiment results with SVHN

Activation	$A_{std}$	PGD	Deepfool
ReLU	93.88%	44.06%	0.00%
k-WTA-0.2	93.06%	10.00%	36.88%
k-WTA-0.1	87.43%	6.56%	41.56%

Table 8: Pre-trained ResNet18 experiment results with CIFAR-10