LEARNING ACTIVATION FUNCTIONS TO IMPROVE DEEP NEURAL NETWORKS

Forest Agostinelli
Department of Computer Science
University of California - Irvine
Irvine, CA 92697, USA
{fagostin}@uci.edu

Matthew Hoffman
Adobe Research
San Francisco, CA 94103, USA
{mathoffm}@adobe.com

Peter Sadowski, Pierre Baldi
Department of Computer Science
University of California - Irvine
Irvine, CA 92697, USA
{peter.j.sadowski,pfbaldi}@uci.edu

ABSTRACT

Artificial neural networks typically have a fixed, non-linear activation function at each neuron. We have designed a novel form of piecewise linear activation function that is learned independently for each neuron using gradient descent. With this adaptive activation function, we are able to improve upon deep neural network architectures composed of static rectified linear units, achieving state-of-the-art performance on CIFAR-10 (7.51%), CIFAR-100 (30.83%), and a benchmark from high-energy physics involving Higgs boson decay modes.

1 INTRODUCTION

Deep learning with artificial neural networks has enabled rapid progress on applications in engineering (e.g., Krizhevsky et al., 2012; Hannun et al., 2014) and basic science (e.g., Di Lena et al., 2012; Lusci et al., 2013; Baldi et al., 2014). Usually, the parameters in the linear components are learned to fit the data, while the nonlinearities are pre-specified to be a logistic, tanh, rectified linear, or max-pooling function. A sufficiently large neural network using any of these common nonlinear functions can approximate arbitrarily complex functions (Hornik et al., 1989; Cho & Saul, 2010), but in finite networks the choice of nonlinearity affects both the learning dynamics (especially in deep networks) and the network's expressive power.

Designing activation functions that enable fast training of accurate deep neural networks is an active area of research. The rectified linear activation function (Jarrett et al., 2009; Glorot et al., 2011), which does not saturate like sigmoidal functions, has made it easier to quickly train deep neural networks by alleviating the difficulties of weight-initialization and vanishing gradients. Another recent innovation is the "maxout" activation function, which has achieved state-of-the-art performance on multiple machine learning benchmarks (Goodfellow et al., 2013). The maxout activation function computes the maximum of a set of linear functions, and has the property that it can approximate any convex function of the input. Springenberg & Riedmiller (2013) replaced the $\max$ function with a probabilistic $\max$ function and Gulcehre et al. (2014) explored an activation function that replaces the $\max$ function with an L_P norm. However, while the type of activation function can have a significant impact on learning, the space of possible functions has hardly been explored.

One way to explore this space is to *learn* the activation function during training. Previous efforts to do this have largely focused on genetic and evolutionary algorithms (Yao, 1999), which attempt to select an activation function for each neuron from a pre-defined set. Recently, Turner & Miller (2014) combined this strategy with a single scaling parameter that is learned during training.

In this paper, we propose a more powerful adaptive activation function. This parametrized, piecewise linear activation function is learned independently for each neuron using gradient descent, and can represent both convex and non-convex functions of the input. Experiments demonstrate that like other piecewise linear activation functions, this works well for training deep neural networks, and we obtain state-of-the-art performance on multiple benchmark deep learning tasks.

公式：

2 Adaptive Piecewise Linear Units

Here we define the adaptive piecewise linear (APL) activation unit. Our method formulates the activation function $h_i(x)$ of an APL unit i as a sum of hinge-shaped functions,

$$h_i(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s) \tag{1}$$

The result is a piecewise linear activation function. The number of hinges, S, is a hyperparameter set in advance, while the variables a_i^s, b_i^s for $i \in 1, ..., S$ are learned using standard gradient descent during training. The a_i^s variables control the slopes of the linear segments, while the b_i^s variables determine the locations of the hinges.

The number of additional parameters that must be learned when using these APL units is $2SM$, where M is the total number of hidden units in the network. This number is small compared to the total number of weights in typical networks.

Figure 1 shows example APL functions for $S = 1$. Note that unlike maxout, the class of functions that can be learned by a single unit includes non-convex functions. In fact, for large enough S, $h_i(x)$ can approximate arbitrarily complex continuous functions, subject to two conditions:

Theorem 1 *Any continuous piecewise-linear function $g(x)$ can be expressed by Equation 1 for some S, and a_i, b_i, $i \in 1, ..., S$, assuming that:*

1. *There is a scalar u such that $g(x) = x$ for all $x \geq u$.*
2. *There are two scalars v and α such that $\nabla_x g(x) = \alpha$ for all $x < v$.*

This theorem implies that we can reconstruct any piecewise-linear function $g(x)$ over any subset of the real line, and the two conditions on $g(x)$ constrain the behavior of $g(x)$ to be linear as x gets very large or small. The first condition is less restrictive than it may seem. In neural networks, $g(x)$ is generally only of interest as an input to a linear function $wg(x) + z$; this linear function effectively restores the two degrees of freedom that are eliminated by constraining the rightmost segment of $g(x)$ to have unit slope and bias 0.

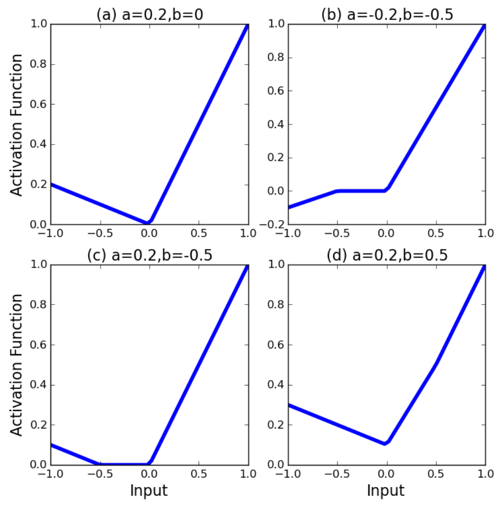


Figure 1: Sample activation functions obtained from changing the parameters. Notice that figure b shows that the activation function can also be non-convex. Asymptotically, the activation functions tend to $g(x) = x$ as $x \to \infty$ and $g(x) = \alpha x - c$ as $x \leftarrow -\infty$ for some α and c. $S = 1$ for all plots.

Proof Let $g(x)$ be piecewise linear with $K + 2$ linear regions separated by ordered boundary points b^0, b^1, ...,b^K, and let a^k be the slope of the k-th region. Assume also that $g(x) = x$ for all $x \geq b^K$.

We show that $g(x)$ can be expressed by the following special case of Equation 1:

$$\begin{aligned} h(x) \equiv & -a^0 \max(0, -x + b^0) \\ & + \textstyle\sum_{k=1}^{K} a^k(\max(0, -x + b^{k-1}) - \max(0, -x + b^k)) \\ & - \max(0, -x) + \max(0, x) + \max(0, -x + b^K), \end{aligned} \tag{2}$$

The first term has slope a^0 in the range $(-\infty, b^0)$ and 0 elsewhere. Each element in the summation term of Equation 2 has slope a^k over the range (b^{k-1}, b^k) and 0 elsewhere. The last three terms together have slope 1 when $x \in (b^K, \infty)$ and 0 elsewhere. Now, $g(x)$ and $h(x)$ are continuous, their slopes match almost everywhere, and it is easily verified that $h(x) = g(x) = x$ for $x \geq b^K$. Thus, we conclude that $h(x) = g(x)$ for all x. $\Box$

2.1 Comparison with Other Activation Functions

In this section we compare the proposed approach to learning activation functions with two other nonlinear activation functions: maxout (Goodfellow et al., 2013), and network-in-network (Lin et al., 2013).

We observe that both maxout units and network-in-network can learn any nonlinear activation function that APL units can, but require many more parameters to do so. This difference allows APL units to be applied in very different ways from maxout and network-in-network nonlinearities: the small number of parameters needed to tune an APL unit makes it practical to train convolutional networks that apply different nonlinearities at each point in each feature map, which would be completely impractical in either maxout networks or network-in-network approaches.

Maxout. Maxout units differ from traditional neural network nonlinearities in that they take as input the output of *multiple* linear functions, and return the largest:

$$h_{\text{maxout}}(x) = \max_{k \in \{1,\ldots,K\}} w^k \cdot x + b^k. \tag{3}$$

Incorporating multiple linear functions increases the expressive power of maxout units, allowing them to approximate arbitrary convex functions, and allowing the difference of a pair of maxout units to approximate arbitrary functions.

Networks of maxout units with a particular weight-tying scheme can reproduce the output of an APL unit. The sum of terms in Equation 1 with positive coefficients (including the initial $\max(0, x)$ term) is a convex function, and the sum of terms with negative coefficients is a concave function. One could approximate the convex part with one maxout unit, and the concave part with another maxout unit:

$$h^{\text{convex}}(x) = \max_k c_k^{\text{convex}} w \cdot x + d_k^{\text{convex}}; \quad h^{\text{concave}}(x) = \max_k c_k^{\text{concave}} w \cdot x + d_k^{\text{concave}}, \tag{4}$$

where c and d are chosen so that

$$h^{\text{convex}}(x) - h^{\text{concave}}(x) = \max(0, w \cdot x + u) + \textstyle\sum_s a^s \max(0, w \cdot x + u). \tag{5}$$

In a standard maxout network, however, the w vectors are not tied. So implementing APL units (Equation 1) using a maxout network would require learning $O(SK)$ times as many parameters, where K is the size of the maxout layer's input vector. Whenever the expressive power of an APL unit is sufficient, using the more complex maxout units is therefore a waste of computational and modeling power.

Network-in-Network. Lin et al. (2013) proposed replacing the simple rectified linear activation in convolutional networks with a fully connected network whose parameters are learned from data. This "MLPConv" layer couples the outputs of all filters applied to a patch, and permits arbitrarily complex transformations of the inputs. A depth-M MLPConv layer produces an output vector f_{ij}^M from an input patch x_{ij} via the series of transformations

$$f_{ijk}^1 = \max(0, w_k^1 \cdot x_{ij} + b_k^1), \ldots, f_{ijk}^M = \max(0, w_k^M \cdot f_{ij}^{M-1} + b_k^M). \tag{6}$$

As with maxout networks, there is a weight-tying scheme that allows an MLPConv layer to reproduce the behavior of an APL unit:

$$f^1_{ijk} = \max(0, c_k w_{\kappa(k)} \cdot x_{ij} + b^1_k), f^2_{ijk} = \textstyle\sum_{\ell|\kappa(\ell)=k} a_k f^1_{ij\ell}, \tag{7}$$

where the function $\kappa(k)$ maps from hinge output indices k to filter indices κ, and the coefficient $c_k \in \{-1, 1\}$.

This is a very aggressive weight-tying scheme that dramatically reduces the number of parameters used by the MLPConv layer. Again we see that it is a waste of computational and modeling power to use network-in-network wherever an APL unit would suffice.

However, network-in-network can do things that APL units cannot—in particular, it efficiently couples and summarizes the outputs of multiple filters. One can get the benefits of both architectures by replacing the rectified linear units in the MLPconv layer with APL units.

3 Experiments

Experiments were performed using the software package CAFFE (Jia et al., 2014). The hyperparameter, S, that controls the complexity of the activation function was determined using a validation set for each dataset. The a^s_i and b^s_i parameters were regularized with an L2 penalty, scaled by 0.001. Without this penalty, the optimizer is free to choose very large values of a^s_i balanced by very small weights, which would lead to numerical instability. We found that adding this penalty improved results. The model files and solver files are available at https://github.com/ForestAgostinelli/Learned-Activation-Functions-Source/tree/master.

3.1 CIFAR

The CIFAR-10 and CIFAR-100 datasets (Krizhevsky & Hinton, 2009) are 32x32 color images that have 10 and 100 classes, respectively. They both have 50,000 training images and 10,000 test images. The images were preprocessed by subtracting the mean values of each pixel of the training set from each image. Our network for CIFAR-10 was loosely based on the network used in (Srivastava et al., 2014). It had 3 convolutional layers with 96, 128, and 256 filters, respectively. Each kernel size was 5x5 and was padded by 2 pixels on each side. The convolutional layers were followed by a max-pooling, average-pooling, and average-pooling layer, respectively; all with a kernel size of 3 and a stride of 2. The two fully connected layers had 2048 units each. We applied dropout (Hinton et al., 2012) to the network as well. We found that applying dropout both before *and* after a pooling layer increased classification accuracy. The probability of a unit being dropped before a pooling layer was 0.25 for all pooling layers. The probability for them being dropped after each pooling layers was 0.25, 0.25, and 0.5, respectively. The probability of a unit being dropped for the fully connected layers was 0.5 for both layers. The final layer was a softmax classification layer. For CIFAR-100, the only difference was the second pooling layer was max-pooling instead of average-pooling. The baseline used rectified linear activation functions.

When using the APL units, for CIFAR-10, we set $S = 5$. For CIFAR-100 we set $S = 2$. Table 1 shows that adding the APL units improved the baseline by over 1% in the case of CIFAR-10 and by almost 3% in the case of CIFAR-100. *In terms of relative difference, this is a 9.4% and a 7.5% decrease in error rate, respectively.* We also try the network-in-network architecture for CIFAR-10 (Lin et al., 2013). We have $S = 2$ for CIFAR-10 and $S = 1$ for CIFAR-100. We see that it improves performance for both datasets.

We also try our method with the augmented version of CIFAR-10 and CIFAR-100. We pad the image all around with a four pixel border of zeros. For training, we take random 32 x 32 crops of the image and randomly do horizontal flips. For testing we just take the center 32 x 32 image. To the best of our knowledge, the results we report for data augmentation using the network-in-network architecture *are the best results reported for CIFAR-10 and CIFAR-100 for any method.*

In section 3.4, one can observe that the learned activations can look similar to leaky rectified linear units (Leaky ReLU) (Maas et al., 2013). This activation function is slightly different than the ReLU because it has a small slope k when the input $x < 0$.

$$h(x) = \begin{cases} x, & \text{if } x > 0 \\ kx, & \text{otherwise} \end{cases}$$

In (Maas et al., 2013), k is equal to 0.01. To compare Leaky ReLUs to our method, we try different values for k and pick the best value one. The possible values are positive and negative 0.01, 0.05, 0.1, and 0.2. For the standard convolutional neural network architecture $k = 0.05$ for CIFAR-10 and $k = -0.05$ for CIFAR-100. For the network-in-network architecture $k = 0.05$ for CIFAR-10 and $k = 0.2$ for CIFAR-100. APL units consistently outperform leaky ReLU units, showing the value of tuning the nonlinearity (see also section 3.3).

Table 1: Error rates on CIFAR-10 and CIFAR-100 with and without data augmentation. This includes standard convolutional neural networks (CNNs) and the network-in-network (NIN) architecture (Lin et al., 2013). The networks were trained 5 times using different random initializations — we report the mean followed by the standard deviation in parenthesis. The best results are in bold.

Method	**CIFAR-10**	**CIFAR-100**
Without Data Augmentation		
CNN + ReLU (Srivastava et al., 2014)	12.61%	37.20%
CNN + Channel-Out (Wang & JaJa, 2013)	13.2%	36.59%
CNN + Maxout (Goodfellow et al., 2013)	11.68%	38.57%
CNN + Probout (Springenberg & Riedmiller, 2013)	**11.35%**	38.14%
CNN (Ours) + ReLU	12.56 (0.26)%	37.34 (0.28)%
CNN (Ours) + Leaky ReLU	11.86 (0.04)%	35.82 (0.34)%
CNN (Ours) + APL units	11.38 (0.09)%	**34.54 (0.19)%**
NIN + ReLU (Lin et al., 2013)	10.41%	35.68%
NIN + ReLU + Deep Supervision (Lee et al., 2014)	9.78%	34.57%
NIN (Ours) + ReLU	9.67 (0.11)%	35.96 (0.13)%
NIN (Ours) + Leaky ReLU	9.75 (0.22)%	36.00 (0.36)%
NIN (Ours) + APL units	**9.59 (0.24)%**	**34.40 (0.16)%**
With Data Augmentation		
CNN + Maxout (Goodfellow et al., 2013)	9.38%	-
CNN + Probout (Springenberg & Riedmiller, 2013)	9.39%	-
CNN + Maxout (Stollenga et al., 2014)	9.61%	34.54%
CNN + Maxout + Selective Attention (Stollenga et al., 2014)	**9.22%**	**33.78%**
CNN (Ours) + ReLU	9.99 (0.09)%	34.50 (0.12)%
CNN (Ours) + APL units	9.89 (0.19)%	33.88 (0.45)%
NIN + ReLU (Lin et al., 2013)	8.81%	-
NIN + ReLU + Deep Supervision (Lee et al., 2014)	8.22%	-
NIN (Ours) + ReLU	7.73 (0.13)%	32.75 (0.13)%
NIN (Ours) + APL units	**7.51 (0.14)%**	**30.83 (0.24)%**

3.2 Higgs Boson Decay

The Higgs-to-$\tau^+\tau^-$ decay dataset comes from the field of high-energy physics and the analysis of data generated by the Large Hadron Collider (Baldi et al., 2015). The dataset contains 80 million collision events, characterized by 25 real-valued features describing the 3D momenta and energies of the collision products. The supervised learning task is to distinguish between two types of physical processes: one in which a Higgs boson decays into $\tau^+\tau^-$ leptons and a background process that produces a similar measurement distribution. Performance is measured in terms of the area under the receiver operating characteristic curve (AUC) on a test set of 10 million examples, and in terms of discovery significance (Cowan et al., 2011) in units of Gaussian σ, using 100 signal events and 5000 background events with a 5% relative uncertainty.

Our baseline for this experiment is the 8 layer neural network architecture from (Baldi et al., 2015) whose architecture and training hyperparameters were optimized using the Spearmint algorithm (Snoek et al., 2012). We used the same architecture and training parameters except that dropout was used in the top two hidden layers to reduce overfitting. For the APL units we used $S = 2$. Table 2 shows that a single network with APL units achieves state-of-the-art performance, increasing performance over the dropout-trained baseline and the ensemble of 5 neural networks from (Baldi et al., 2015).

Table 2: Performance on the Higgs boson decay dataset in terms of both AUC and expected discovery significance. The networks were trained 4 times using different random initializations — we report the mean followed by the standard deviation in parenthesis. The best results are in bold.

Method	**AUC**	**Discovery Significance**
DNN + ReLU (Baldi et al., 2015)	0.802	3.37σ
DNN + ReLU + Ensemble(Baldi et al., 2015)	0.803	3.39σ
DNN (Ours) + ReLU	0.803 (0.0001)	3.38 (0.008) σ
DNN (Ours) + APL units	**0.804 (0.0002)**	**3.41 (0.006)** σ

3.3 Effects of APL unit Hyperparameters

Table 3 shows the effect of varying S on the CIFAR-10 benchmark. We also tested whether learning the activation function was important (as opposed to having complicated, *fixed* activation functions). For $S = 1$, we tried freezing the activation functions at their random initialized positions, and not allowing them to learn. The results show that learning activations, as opposed to keeping them fixed, results in better performance.

Table 3: Classification accuracy on CIFAR-10 for varying values of S. Shown are the mean and standard deviation over 5 trials.

Values of S	**Error Rate**
baseline	12.56 (0.26)%
$S = 1$ (activation not learned)	12.55 (0.11)%
$S = 1$	11.59 (0.16)%
$S = 2$	11.73 (0.23)%
$S = 5$	**11.38 (0.09)%**
$S = 10$	11.60 (0.16)%

3.4 Visualization and Analysis of Adaptive Piecewise Linear Functions

The diversity of adaptive piecewise linear functions was visualized by plotting $h_i(x)$ for sample neurons. Figures 2 and 3 show adaptive piecewise linear functions for the CIFAR-100 and Higgs$\rightarrow \tau^+\tau^-$ experiments, along with the random initialization of that function.

In figure 4, for each layer, 1000 activation functions (or the maximum number of activation functions for that layer, whichever is smaller) are plotted. One can see that there is greater variance in the learned activations for CIFAR-100 than there is for CIFAR-10. There is greater variance in the learned activations for Higgs$\rightarrow \tau^+\tau^-$ than there is for CIFAR-100. For the case of Higgs$\rightarrow \tau^+\tau^-$, a trend that can be seen is that the variance decreases in the higher layers.

4 Conclusion

We have introduced a novel neural network activation function in which each neuron computes an independent, piecewise linear function. The parameters of each neuron-specific activation function are learned via gradient descent along with the network's weight parameters. Our experiments demonstrate that learning the activation functions in this way can lead to significant performance improvements in deep neural networks without significantly increasing the number of parameters. Furthermore, the networks learn a diverse set of activation functions, suggesting that the standard one-activation-function-fits-all approach may be suboptimal.

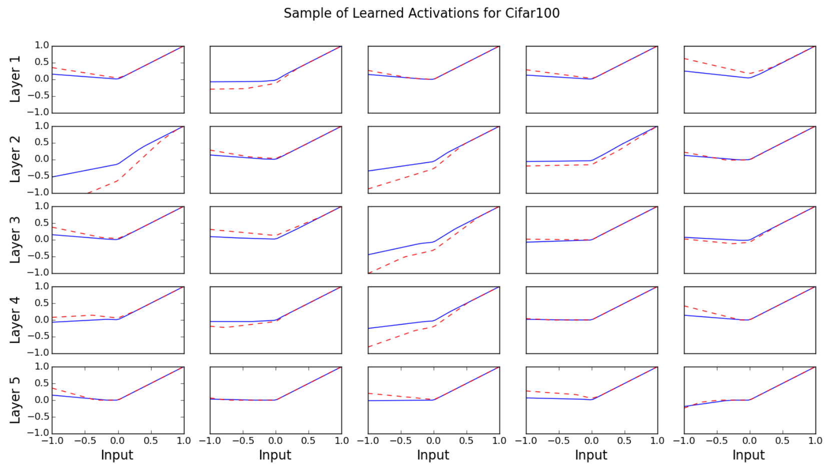


Figure 2: CIFAR-100 Sample Activation Functions. Initialization (dashed line) and the final learned function (solid line).

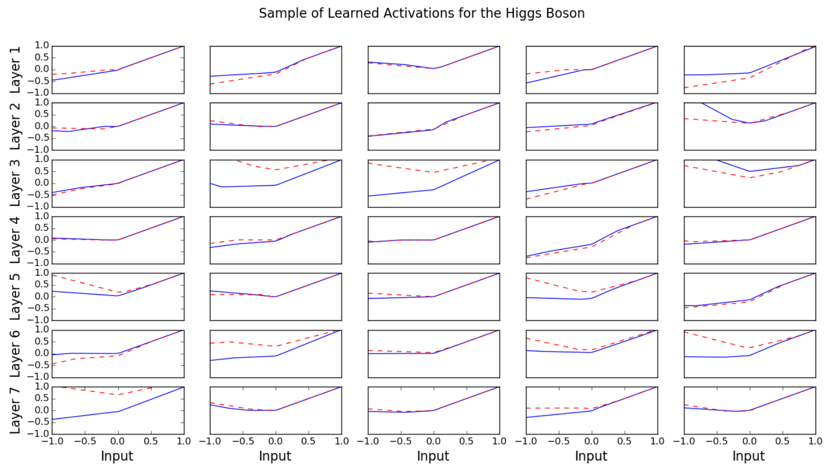


Figure 3: Higgs$\rightarrow \tau^+\tau^-$ Sample Activation Functions. Initialization (dashed line) and the final learned function (solid line).

Acknowledgments

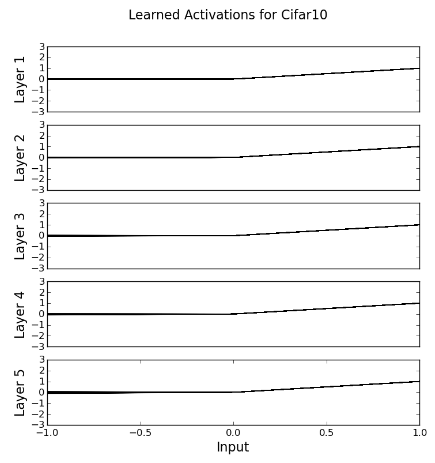
F. Agostinelli was supported by the GEM Fellowship. This work was done during an internship at Adobe. We also wish to acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research, NSF grant IIS-0513376, and a Google Faculty Research award to P. Baldi, and thanks to Yuzo Kanomata for computing support.

References

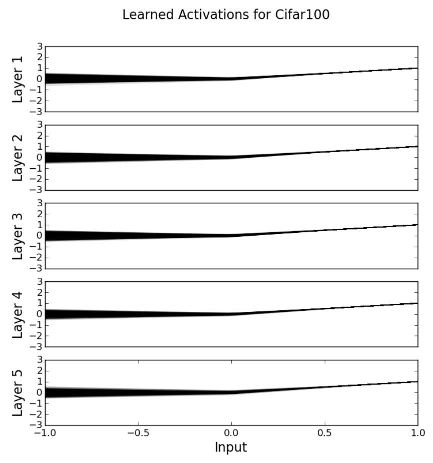
Baldi, P, Sadowski, P, and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5, 2014.

Baldi, Pierre, Sadowski, Peter, and Whiteson, Daniel. Enhanced higgs to $\tau^+\tau^-$ searches with deep learning. *Physics Review Letters*, 2015. In press.

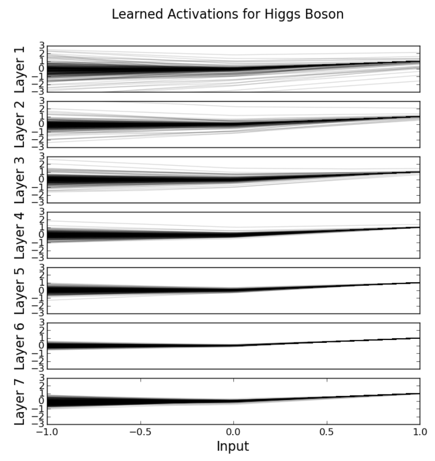
Cho, Youngmin and Saul, Lawrence. Large margin classification in infinite neural networks. *Neural Computation*, 22(10), 2010.



(a) CIFAR-10 Activation Functions.



(b) CIFAR-100 Activation Functions.



(c) Higgs→ $\tau^+\tau^-$ Activation Functions.

Figure 4: Visualization of the range of the values for the learned activation functions for the deep neural network for the CIFAR datasets and Higgs→ $\tau^+\tau^-$ dataset.

Cowan, Glen, Cranmer, Kyle, Gross, Eilam, and Vitells, Ofer. Asymptotic formulae for likelihood-based tests of new physics. *Eur.Phys.J.*, C71:1554, 2011. doi: 10.1140/epjc/s10052-011-1554-0.

Di Lena, P., Nagata, K., and Baldi, P. Deep architectures for protein contact map prediction. *Bioinformatics*, 28:2449–2457, 2012. doi: 10.1093/bioinformatics/bts475. First published online: July 30, 2012.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, volume 15, pp. 315–323, 2011.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Gulcehre, Caglar, Cho, Kyunghyun, Pascanu, Razvan, and Bengio, Yoshua. Learned-norm pooling for deep feedforward and recurrent neural networks. In *Machine Learning and Knowledge Discovery in Databases*, pp. 530–546. Springer, 2014.

Hannun, Awni Y., Case, Carl, Casper, Jared, Catanzaro, Bryan C., Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, and Ng, Andrew Y. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014. URL `http://arxiv.org/abs/1412.5567`.

Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Hornik, Kurt, Stinchcombe, Maxwell, and White, Halbert. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Jarrett, Kevin, Kavukcuoglu, Koray, Ranzato, M, and LeCun, Yann. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2146–2153. IEEE, 2009.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

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

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Lusci, Alessandro, Pollastri, Gianluca, and Baldi, Pierre. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. *Journal of chemical information and modeling*, 53(7):1563–1575, 2013.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

Springenberg, Jost Tobias and Riedmiller, Martin. Improving deep neural networks with probabilistic maxout units. *arXiv preprint arXiv:1312.6116*, 2013.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Stollenga, Marijn F, Masci, Jonathan, Gomez, Faustino, and Schmidhuber, Jürgen. Deep networks with internal selective attention through feedback connections. In *Advances in Neural Information Processing Systems*, pp. 3545–3553, 2014.

Turner, Andrew James and Miller, Julian Francis. Neuroevolution: Evolving heterogeneous artificial neural networks. *Evolutionary Intelligence*, pp. 1–20, 2014.

Wang, Qi and JaJa, Joseph. From maxout to channel-out: Encoding information on sparse pathways. *arXiv preprint arXiv:1312.1909*, 2013.

Yao, Xin. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.