Hyperactivations for Activation Function Exploration

Conner Joseph Vercellino
College of Creative Studies
University of California
Santa Barbara, CA 93106
cjv@ucsb.edu

William Yang Wang
Department of Computer Science
University of California
Santa Barbara, CA 93106
william@cs.ucsb.edu

Abstract

Typically, when designing neural network architectures, a fixed activation function is chosen to introduce nonlinearity between layers. Various architecture agnostic activation functions have been proposed. However a less explored area of research, is how to learn task specific activation functions while training the network. Activation functions (even as fixed nonlinearities) can affect the network's capacity, learning dynamics and convergence properties. Allowing for varied activations for each layer or neuron could have an even more pronounced affect on the stated properties. Furthermore, it is difficult to form intuition around what properties are desirable in a static activation function, and even more difficult to design task specific activation functions that differ on each layer or neuron. Allowing the network to discover activations during training bypasses the need to fully understand the activation function's interactions. To explore the search space of activation functions, we propose hyperactivations, a novel approach to learning task specific activations while maintaining training stability. Using hyperactivations we beat commonly used activation functions — both in training speed and test performance — and learn activations that deviate wildly from the norm.

1 Introduction

A common practice for designing neural network models is to use a human-engineered activation function (e.g., Tanh, ReLU, sigmoid) between layers. The activation functions selected are generally non-parameterized and created to generalize well across different network architectures. While generally seen as a minor detail, activation functions strongly affect the network's speed of convergence, capacity, and overall performance.

Many activation functions have been proposed with the intent of obtaining strong performance cross architecture, especially for convolutional neural networks. Some recent examples of activation functions include: ELU (Clevert et al., 2015), PELU (Trottier et al., 2016) and Swish[1] (Ramachandran et al., 2017). Use of these activation functions varies, as their performance can highly depend on the architecture and task, despite the intention that they would easily transfer to new architectures.

There have also been attempts to find activation functions with desirable convergence properties. Klambauer et al. (2017) proposed SELU, an activation function that acts similarly to batch normalization (Ioffe and Szegedy, 2015), given some assumptions about the weight initiation. Ideally, we can formulate activation functions that act as new forms of normalization, especially if the activation function is learned for each layer. However, properties that make an activation function desirable are difficult to discern. SELU for instance, required laborious analysis to discern a seemingly "magic" value in order to have its beneficial properties.

[1]While the activation that was proposed by Ramachandran et al. (2017) was not novel, we include it to demonstrate a recent attempt to find a generalizable activation function.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Proposals for nonlinearities are also biased for interpretable activation functions, as it's difficult to know how small tweaks to an activation function will affect it's performance. Without an immense improvement in our understanding of neural networks, researchers are unable to find activation functions optimal for their architectures. It is an open question how varying an activation function between layers affects the network (let alone varying many different activations throughout the network).

With these problems in mind, we propose hyperactivations, a way to learn activations during the training of the model, permitting an exploration of the activation search space. Given that an activation function can affect many properties of a neural network, changing the activation as the network trains seems potentially optimal for model performance. Hyperactivations are based around using hypernetworks, utilizing them for their normalization properties. We show that hyperactivation functions learn novel activation functions not found in human-designed networks, while maintaining training stability. We summarize our contributions as two fold: we are the first to consider a hypernetwork-based meta-learning approach for learning activation functions, and empirically, our approach learned task-specific activation functions that outperformed models using standard activation functions.

2 Prior Work

Agostinelli et al. (2014) proposed using a sum of hinge functions to learn an activation functions, whereby each activation function is learned per a neuron. This approach is admirable in its simplicity, but it has two problems: the method assumes a function family, and it results in rough function surfaces. Eisenach et al. (2016) approach did not assume a function family. We question the strength of the activation functions learned by Eisenach et al. (2016) as they seem overnormalized (hence the indentations near each side of the learned activation).

Learning an activation function per a neuron greatly simplifies the problem of training instability. We found stability very difficult to maintain when the activation function was parameterized, as a small change in the activation function will affect every output of a layer leading to training instability. However, if you learn activations per a neuron, the method must be computationally and parameter efficient, as you need to scale per a neuron not per a layer. Given that our method has higher computational and parameter requirements than previous methods, and given that we wanted to learn comparable functions to commonly used human-designed activation functions, we chose to learn activations per a layer.

Another method for attempting to discover novel nonlinearities is to use reinforcement learning to generate nonlinearities from the results of attempting to train many networks. This approach, used to find Swish, is very similar to the one presented by Zoph and Le (2017), as it uses a fixed vocabulary to create an activation function, tests the activation on various network structures, and then trains itself on a reward; a reward based on the activation function's performance. This approach, while resulting in more comprehensible, generalizable activation functions, also has a few clear downsides. The activation functions in the search space are limited to those that can be expressed with a small vocabulary of mathematical operators, and no guarantees exist that the optimal generalizable function exists in such a rigid space. Furthermore, this method requires an extreme amount of computational power, limiting its accessibility to researchers. Finally, the method used in Swish is aimed at finding a generalizable activation function, not activation functions that are task specific.

3 Hyperactivations

One hyperactivation takes the place of all activations in the network. A hyperactivation is constructed from two parts: an activation network and a hypernetwork. The activation network is a shallow forwardfeed neural network that predicts one output for each input $i_j \forall o_j$. The flattening of the activation network's input allows a hyperactivation to be placed anywhere a normal activation function would. It also allows the activation network to be visualized on a 2D graph. The activation network needs to have an activation function to bootstrap from (we attempted nested hyperactivations, but found this did not work as a result of training instability). With the vector and reshape operations, the activation network is defined as:

$$AN(x) = \text{reshape}(A(\text{vec}(x), W_a), x_{(wj,hj)})$$

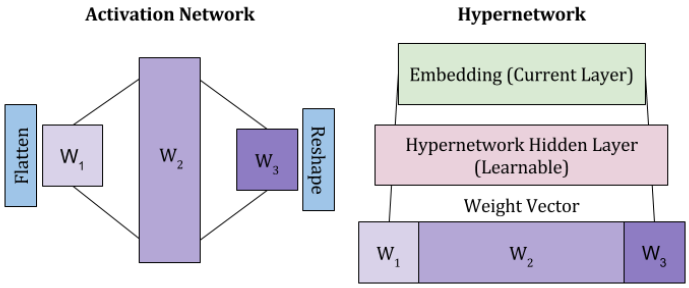

Figure 1: A diagram of the two parts of a hyperactivation. We use W_n to show how the weight vector is used for the activation network's weight matrices.

Where A is the nonlinearity (for our experiments ReLU), W_a is the activation network weight matrix, and x is the input. However, using this small activation network is not enough to learn activations per a layer, as normalization was found essential for even toy tasks. Since we are learning activation per a layer, normalization is required, as a small shift in the learned activation completely shifts the network's output. If the inputs fluctuate wildly (for instance, upstream activations changing), it is hard to learn anything. We initially attempted to use batch normalization and later, layer normalization (Lei Ba et al., 2016) in the construction of the activation network. Batch normalization resulted in decent performance for small to medium networks. However, it was unbelievably slow as it needed to operate on potentially large flattened batches given to the activation network. We attempted layer normalization, but found it didn't work at all. We also attempted to combine the regularization methods, which yielded no benefit to train time or accuracy.

In an attempt to find a better normalization strategy, we turned to hypernetworks. Hypernetworks are a type of neural network that produce weights for another network. If we incorporate a hypernetwork into our previously defined activation network:

$$HA(x) = \text{reshape}(A(\text{vec}(x), H(e, W_h)), x_{(wj,hj)})$$

Here, e is the embedding of the current layer that the hypernetwork uses as context of what weight vector to output for use in the activation network and H is the hypernetwork that produces the weights. The hypernetwork produces W_a when given the current embedding. The implementation of hyperactivations, while easy to express notationally, can be frustrating to implement for generalizability over any given activation network. We provide our code for all experiments.[2] Our hypernetwork structure is very similar to the static one proposed by Ha et al. (2016). Hypernetworks have a few different attractions, but our main interest was using it as a form of normalization for our activation network. Each activation of the hyperactivation is given an embedding, and the hypernetwork produces weights for the activation network. Each activation network has a loose form of weight sharing, with each of the other activations in the network on account of using a hypernetwork to create each of their weights.

4 Experiments

4.1 MNIST

We first tested on MNIST (Lecun et al., 1998) with a small convolutional network trained using Adam (Kingma and Ba, 2014). We find that the smaller the network, the more nonlinear the nonlinearities. We assume that this result is because the larger network needs less nonlinearity to learn the dataset. We outperform all other tested activations, both in convergence speed (Figure 2) and test accuracy (Table 1). We show the learned activations in figure (Figure 3) and find that they are radically different from popular activation functions. Our hyperactivation had both an embedding and hidden size of 10 and the activation network had a hidden size of 8.

[2] `www.github.com/c0nn3r/learned_activations`

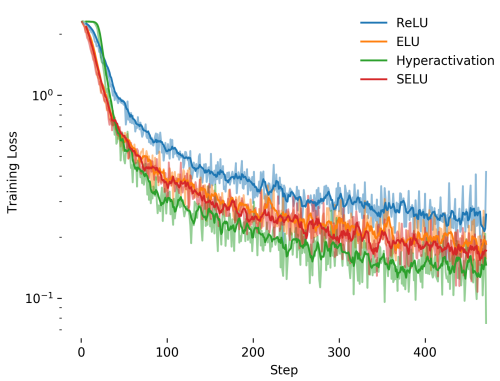

Figure 2: Losses of common activation functions compared to a Hyperactivation.

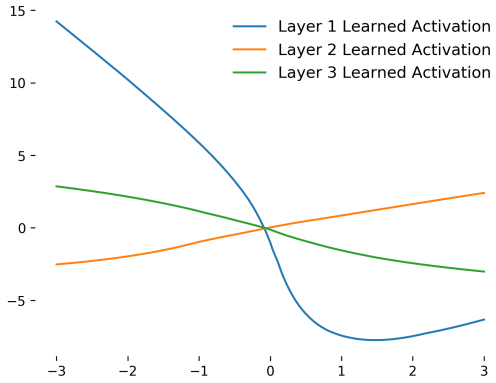

Figure 3: The activation functions learned during the training of the network.

Table 1: Test Loss and Accuracy for MNIST.

Activation	Loss	Test Acc.
SELU	0.0906	97.04%
ELU	0.0773	97.42%
ReLU	0.0833	97.45%
Hyperactivation	**0.0641**	**98.36%**

Table 2: Test Loss and Accuracy for CIFAR-10.

Activation	Loss	Test Acc.
SELU	0.590	79.70%
ELU	0.552	81.62%
ReLU	0.426	85.80%
Hyperactivation	**0.371**	**88.66%**

4.2 CIFAR-10

We also tested our hyperactivations on CIFAR-10 using the ResNet-16 architecture, replacing the 9 ReLUs with hyperactivations. The hypernetwork having a hidden size of 10, an embedding size of 10 and a activation hidden size of 10. The network was trained for 10 epochs with Adam. We saw faster convergence speed and accuracy using hyperactivations (Table 2).

We would like to note that no parameter tuning was done from the network optimized for ReLU. We created a PyTorch script that replaced the activations used in a given network with hyperactivations. We assume there is improvement to be found by finding optimal hyperparameters for the hyperactivation, but also for the general model hyperparameters, optimized for having hyperactivations.

5 Conclusion

In this paper we introduce hyperactivations, a novel way of learning nonlinearities for each layer of a neural network. We show that we learn various activations that differ from current convention, and that these learned activations converge faster and result in higher test accuracy. We believe hyperactivations are an important step in the process of parameterizing more of the network's design. In the future, we want to look at the activations used for LSTMs, GRUs, and other types of recurrent cells to see if the activations normally used are optimal. We feel there is a lack of research into how choice of activations traditionally recurrent cells. We also want to apply our hyperactivations to larger datasets.

Acknowledgements

The authors would like to thank Mike Wu and Jen Selby.

References

Agostinelli, F., Hoffman, M. D., Sadowski, P. J., and Baldi, P. (2014). Learning activation functions to improve deep neural networks. *CoRR*, abs/1412.6830.

Clevert, D., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289.

Eisenach, C., Liu, H., and Wang, Z. (2016). Nonparametrically learning activation functions in deep neural net.

Ha, D., Dai, A. M., and Le, Q. V. (2016). Hypernetworks. *CoRR*, abs/1609.09106.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-Normalizing Neural Networks. *ArXiv e-prints*.

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

Lei Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer Normalization. *ArXiv e-prints*.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for Activation Functions. *ArXiv e-prints*.

Trottier, L., Giguère, P., and Chaib-draa, B. (2016). Parametric exponential linear unit for deep convolutional neural networks. *CoRR*, abs/1605.09332.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning.