Trainable activation functions in neural networks
Tanítható aktivációs függvények neuron hálózatokban
(December 2018)

Gergely Karz and András Kovács, students, *BME*

***Kivonat*— Az aktivációs függvény egy fontos hiperparamétere a mély neurális hálózatoknak, melynek jelentős hatása van teljesítményükre. Manapság a leginkább elterjedtebb függvény a Rectified Linear Unit (ReLU)[1], habár egyes megoldásokhoz a sigmoid and hyperbolic tangent[2] optimális (például kimeneti rétegeken). A jelen munkánkban megkíséreljük ennek a hiperparaméternek tanítható komponensé tételét a hálózatban. A kutatásunk egy első lépés, ennek az ötletnek a hatékonyságának a bizonyítására. Az alap koncepció szerint, néhány, széles körben alkalmazott aktivációs függvényeknek vesszük lineáris kombinációjukat és ezek súlyait tanítható paraméterként valósítjuk meg a hálózatban, erre Weighted Superposition Gated Function (WSGAF) néven hivatkozunk a továbbiakban. További kutatásra lesz szükség az elv finomítására, melyre a cikk végén részletesen kitérünk.**

***Abstract*—The activation function is vital hyperparameter of deep neural networks, which has significant impact on the performance of these structures. The widely-used and one of the most effective activation function is the Rectified Linear Unit (ReLU)[1], however many solutions use sigmoid and hyperbolic tangent[2] as well. In our work we combined these functions to convert this hyperparameter into a trainable part of the network. Our research explores the first steps in verifying the effectiveness of such an idea. The basic concept is that we use these activation functions in a linear combination with learnable weights. We name this structure Weighted Superposition Gated Function (WSGAF). Additional research is needed to explore this topic, at the end of this article we recommend some ideas on how to further explore this concept.**

I. Introduction

Activation functions are necessary to enable neural networks to fit non-linear surfaces, which are quite common in real life problems. Currently there exists no general function, which could solve this issue and it is ambiguous what the most suitable choice in a given case is, however it has an outstanding impact on the performance of the model. Latest research suggests significant improvement[3] in this topic using gated activation functions[4,5] such as Swish[6] and SoftExp[7]. Other research also shows significant improvement with activation functions that have trainable weights[14,15].

II. Concept of WSGAF

The topic of this homework is to evaluate the effectiveness of a new type of activation function which is a superposition of previously established activation functions (tanh, ReLU [1], sigmoid) with learnable weights. From now on referred to as WSGAF (Weighted Superposition of Gated Activation Functions)

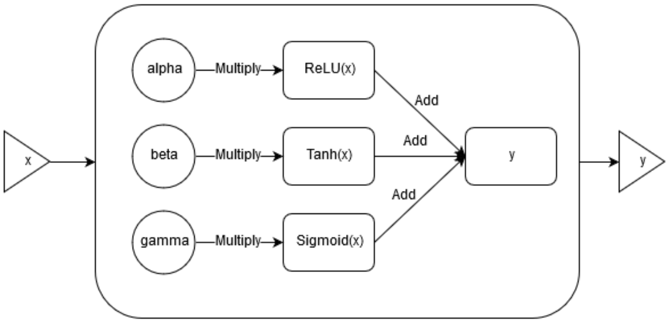

Fig. 1 . WSGAF base concept

We expect this new method to outperform previous heuristics to determine the best activation to use in a certain layer, because this method gets rid of this hyperparameter for every layer and makes the network learn it for itself.

We will evaluate four ways of doing this. The first method (WSGAF-1) is to use separate weights for each neuron. (This method should provide more flexibility for the network, but is also expected to substantially increase learning times, because of the extra weights.) The second method (WSGAF-2) uses the same weights for each neuron per layer (This format is similar to existing structures). In the other two variants, once we multiply the activation functions directly with their respective weights, and in the other, we pass these weights through sigmoid, similarly to other composite activation functions[8].

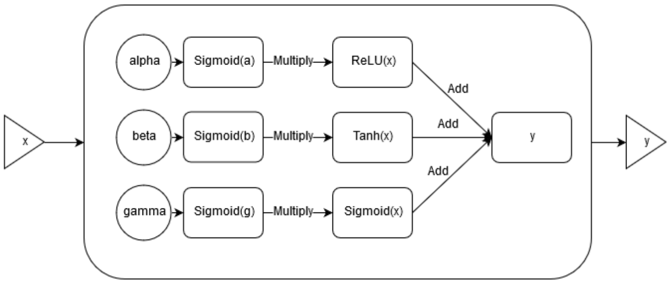

Fig. 2. WSGAF with sigmoid

III. Experiment with WSGAF

During our experimentation, we noticed that initialization of these new weights had a great effect on the loss curves (bad initializations leads to a major plateau at the beginning of training)[9]. For example, using a normal distribution initializer with zero mean and 0.1 deviation yielded a loss curve that did not start to converge before about twenty to forty training steps. This result is not controversial however, as it is expected that with such small values, the information is “lost” in the network before anything meaningful reaches the output layer.

Considering this, we decided to initialize the weights, so our activation function initially matches the behaviour of the established activation functions. To do this we set a single weight’s initializer’s mean close to unit and set all the other weight’s initializers to zero mean. This way the network has a default activation function, from which it can deviate, if the gradients make it necessary.

To evaluate the viability of this new activation function, using the TensorFlow framework[10], we try the following settings on three datasets (MNIST[13], Fashion-MNIST[11] and Cifar10):

TABLE I
TESTED MODEL SETTINGS

Weights per	Layer / neuron
Function applied to weights	Linear / sigmoid
High initial mean for weight of	ReLU / Tanh / Sigmoid / All
Standard deviation of initial weights	Large / Small

Tested model settings
Note: large / small standard deviation is 0.5 and 0.05 when the activation functions are directly multiplied by the weights; 1.0 and 0.1 when weights are passed through sigmoid before multiplying the activation functions with them.

To filter noise produced by random initialization of weights, we ran each model five times, and used the average accuracy to score the model.

TABLE 2
NETWORK STRUCTURE FOR MNIST DATABASES

Layer type	Parameters
conv	1@28x28,f:5x5@32,z:2x2
max_pool	28x28 > 14x14
conv	32@14x14,f:5x5@64,z:2x2
max_pool	14x14 > 7x7
flatten	7x7x64 > 1024
dropout	50%
output (softmax)	1024 > 10

For both of the MNIST databases, we use the same network structure.

TABLE 3
NETWORK STRUCTURE FOR CIFAR10 DATABASE

Layer type	Parameters
conv	3@32x32,f:5x5@64,z:2x2
max_pool	32x32 > 16x16
local_response_normalization	radius=5
conv	64@16x16,f:5x5@64,z:2x2
local_response_normalization	radius=5
maxpool	16x16 > 8x8
flatten	8x8x64 > 384
fully_connected	384 > 192
output (softmax)	192 > 10

This structure is from the TensorFlow implementation. Every layer, unless otherwise noted, uses the currently tested activation function.

We also ran each network with the ReLU activation function for a baseline score to compare our results. This means we trained 2*2*4*2*5, that is 160 models not counting the baseline ReLU models.

Besides using accuracy values at the end of each training session for evaluation, we also graphed loss values, calculated every 10 training steps on the current training batch and the entire validation dataset for MNIST and Fashion-MNIST, however, as the validation set of the cifar10 dataset didn’t fit into memory, we only calculated validation loss on a random subset of the validation set. This causes the validation graph of the cifar10 dataset to be rougher than that of the MNIST graphs.

We did not mix the training batches, before passing them to the network, so as to not make any more differences in learning curve between each test. This causes the train loss graphs to be highly correlated between different activation functions.

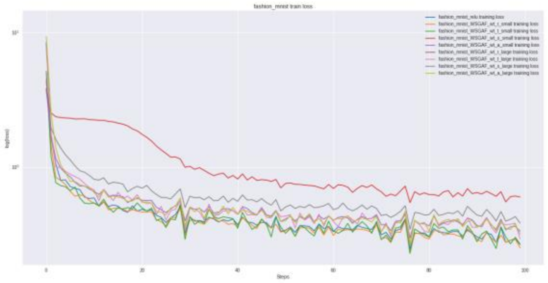

Fig. 3. Train loss of the first test (separate weights by neuron, activation functions directly multiplied by weights) Notice that every model dipps at the same time, because of the same training batch being evaluated.

IV. Results

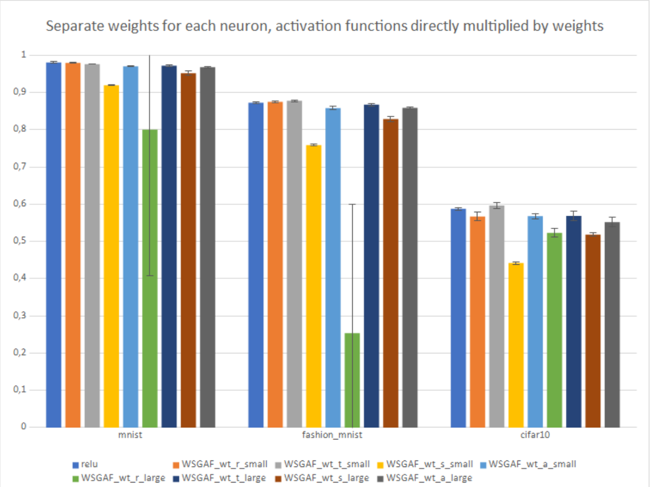

Fig. 4. Separate weights for each neuron, activation functions directly multiplied by weights. Error bar shows one standard deviation of the five tests' accuracies.

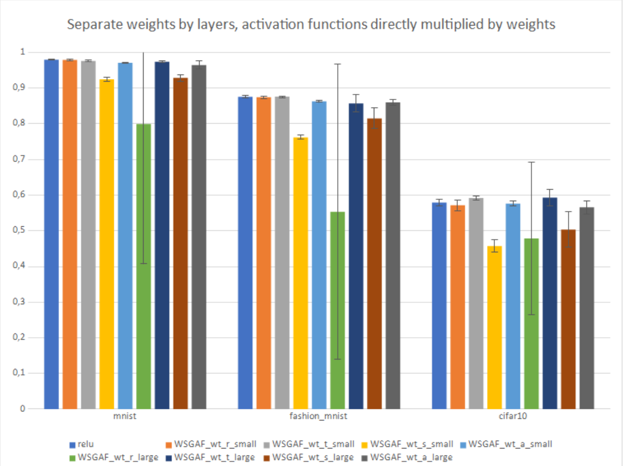

Fig. 5. Separate weights by layer, activation functions directly multiplied by weights. Error bar shows one standard deviation of the five tests' accuracies.

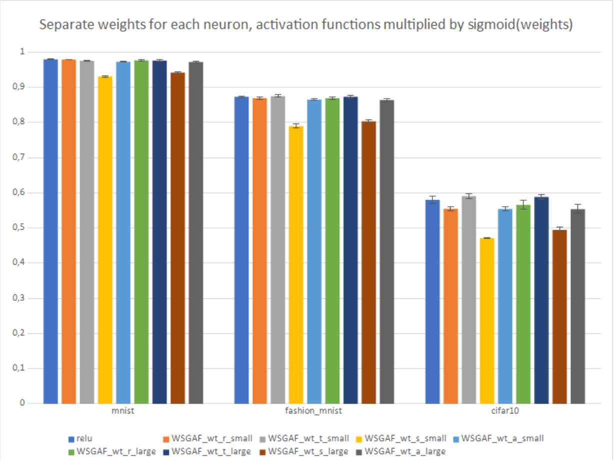

Fig. 6. Separate weights for each neuron, activation functions multiplied by sigmoid(weights). Error bar shows one standard deviation of the five tests' accuracies.

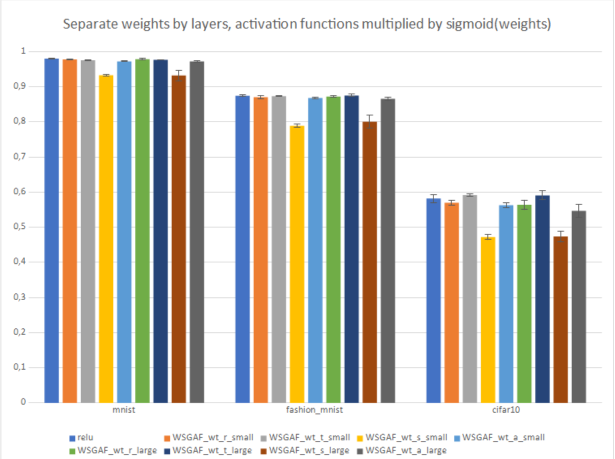

Fig. 7. Separate weights by layer, activation functions directly multiplied by sigmoid(weights). Error bar show one standard deviation of the five tests' accuracies.

The unusually high standard deviation values for some of the WSGAF activations initialized to ReLU with a large standard deviation occur because, there were a couple of training sessions with this setting that did not converge. This issue is not present when we pass the weights through sigmoid before multiplying the activation functions with them. This lets us conclude that this problem is caused by exploding gradients in the network.

In all the tests we did, initializing the weights of WSGAF to mimic the sigmoid activation function lagged behind other approaches, both with small and large standard deviations. This result is peculiar, because even though the sigmoid activation is known to saturate, thus a bad choice as an activation function especially for deep networks, our method multiplies it by a weight which can be greater than one. Combined with the biases in the network, the model should be able to approximate the much better performing TanH function

Based on our results, ReLU performs best on the mnist database, suggesting that a good network architecture effects performance more than fine-tuning the activation functions. We see minor improvements in performance on both the fashion mnist and cifar10 datasets. The best scores are achieved by selecting the TanH activation, using a small standard deviation when initializing weights of WSGAF and using separate weights for each neuron. This advantage is barely significant: for the fashion mnist dataset, +0.16% using sigmoid(w), +0.51% using w, for the cifar10 dataset, +1.03% using sigmoid(w) and +0.92% using w (all values are accuracy percentage points). This improvement in accuracy might be due to the extra entropy that the network can fit caused by the larger amount of trainable weights and might be lost in deeper models, as the TanH activation function tends to underperform compared to ReLU due to the vanishing gradient problem[12].

Using shared weights within layers, we get a slightly different result as the selected TanH using large standard deviation wins over using small deviation in a couple of tests, but this is advantage is not statistically significant.

It is disappointing to observe, that initializing weights to equal mean for all three activation functions (WSGAF_wt_a_*) only performs on par with other approaches, as this is the case where we would expect large improvement because it is only in this case that the network truly has unbiased control over which activation function it uses for the given neuron / layer.

To confirm the new weights do in fact learn, we printed their values before and after a training session for WSGAF_wt_r_small (setting the mean for the ReLU weights to unit, the rest to zero, and the standard deviation for all the weights to 0.05) on the MNIST database, in the first layer of the network. The values did change, this change was minor, however. The mean absolute change of the ReLU weights before and after training is only $3.9*10^{-3}$, for TanH weights it is $4.3*10^{-3}$ and the sigmoid weights $3.3*10^{-3}$. These change values are sub par as they imply that the gradients of the weights between neurons is much greater than that of the weights inside the neurons. This means the number one factor in determining effectiveness of WSGAF is weight initialization as the network barely deviates from the initial values. To verify this result, more testing is needed, for example inspecting statistical parameters of weights of other WSGAF versions and generating more meaningful representations of the data using tools like TensorBoard.

V. Further research

Due to resource and time constraints, significant further research is required to verify our findings. To train the models, we used the google colaboratory tool, in which we were unable to accurately measure training time, which could be impacted by the extra weights. The limited memory of this tool also prohibited us from testing deeper models which tend to work better with ReLU than with the TanH activation. During the course of this semester we also did not have the time to test different models and other activation functions such as PReLU, Swish and other, wilder ones such as Sinusoid, Sinc and the Gaussian. Another promising approach might be using a single weight to interpolate between these activation functions, similar to the one used in the SoftExponential. Further testing needs to be done to determine the extra amount of information the network is able to store in these weights, and also to test the viability of this method on other, non-image datasets.

References

[1] Abien Fred M. Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). arXiv:1803.08375v1 [cs.NE] 22 Mar 2018

[2] Bing Xu, Ruitong Huang, Mu Li. 2016. Revise Saturated Activation Functions. arXiv:1602.05980v2 [cs.LG] 2 May 2016

[3] Jos van der Westhuizen, Joan Lasenby, 2018. The Unreasonable Effectiveness of The Forget Gate. arXiv:1804.04849v3 [cs.NE] 13 Sep 2018

[4] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, 2017. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. arXiv:1711.10433v1 [cs.LG] 28 Nov 2017

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv:1412.3555v1 [cs.NE] 11 Dec 2014

[6] Prajit Ramachandran, Barret Zoph, Quoc V. Le, 2017. Searching for Activation Functions. arXiv:1710.05941v2 [cs.NE] 27 Oct 2017

[7] Luke B. Godfrey and Michael S. Gashler, 2016. A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks. arXiv:1602.01321v1 [cs.NE] 3 Feb 2016

[8] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, Dzmitry Bahdanau, Yoshua Bengio, 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. arXiv:1406.1078v3 [cs.CL] 3 Sep 2014

[9] Siddharth Krishna Kumar, 2017. On weight initialization in deep neural networks. arXiv:1704.08863v2 [cs.LG] 2 May 2017

[10] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, Google Brain, 2016. TensorFlow: A System for Large-Scale Machine Learning Available: https://ai.google/research/pubs/pub45381

[11] Han Xiao, Kashif Rasul, Roland Vollgraf, 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747v2 [cs.LG] 15 Sep 2017

[12] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, 2013. On the difficulty of training Recurrent Neural Networks. arXiv:1211.5063v2 [cs.LG] 16 Feb 2013

[13] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, The Mnist Database of handwritten digits. Available: http://yann.lecun.com/exdb/mnist/

[14] M. Solazzi, 2000. Artificial neural networks with adaptive multidimensional spline activation functions. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium

[15] Masayuki Tanaka, 2018. Weighted Sigmoid Gate Unit for an Activation Function of Deep Neural Network. arXiv:1810.01829v1 [cs.CV] 3 Oct 2018