

Investigating Network Architectures and Activation Functions in Neural Networks

Nikhil Adhe

University of Massachusetts, Amherst

nadhe@umass.edu

Abstract

In this project, I have investigated different activation function like ReLu (Rectified Linear Units), ELU (Exponential Linear Unit), Softplus and Leaky ReLu across different network architectures like fully connected networks, convolution neural networks. The effect of techniques like batch normalization and dropout was also studied across different activation functions. It was found that the Softplus activation function is the ideal choice for fully connected networks while the leaky ReLu activation function was the best choice for Convolution neural networks. Also batchnorm and dropout resulted in improved performance across all activation functions.

The effect of a random weight update schedule, where in the weights of a particular layer are updated according to an assigned probability instead of updating them after every iteration was also investigated. It was found that updating the weights of a layer according to chance hinders the performance of neural networks with lower chance of an update resulting in inferior performance. Also it was shown that doing so slows the 'learning' of the network and hence it is advisable to update the weights of all layers after every iteration.

Introduction

Architecture of a neural network defines how various layers, each consisting of several units or neurons are arranged and how they interact with each other through various parameters like

weights, biases and activation functions. Tinkering with various neural network architectures gives a fundamental understanding about the workings of neural networks which is crucial in determining the best possible choice of a particular architecture for a particular application. Among the various components of neural networks, choice of activation functions and interactions of these functions with different optimization techniques were explored in this project. Also, a different update schedule for the weights of a neural network was investigated in this project.

Background

Activation functions map the inputs to a neuron to produce outputs such that they introduce non-linearity which allows neural networks to act as universal approximators and learn non-linear functions. Various types of activation functions exist. Nair et al^[1] popularized the ReLu activation function which happens to be the most commonly used activation function in neural networks. But the ReLu activation function suffers from the 'dying ReLu' problem in which a large gradient flowing through a neuron alters the weights such that the neuron does not activate from that point on and 'dies'. To address the shortcomings of the ReLu activation function, several other activation functions have been explored. Mass et al^[2] introduced the leaky ReLu activation function which allows a small non-zero gradient to flow through the networks when the neurons are saturated. They observed that leaky ReLus converge faster than ReLu

activation functions. Another attempt to address the dying ReLu problem was made by Glorot et al^[3] when they introduced the Softplus activation function which is the smooth version of the ReLu activation function, the effectiveness of which is explored in this project. Clevert et al^[4] introduced the Exponential Linear Unit (ELU) activation function in which the activations have negative values in contrast to ReLu activation functions. They found that ELUs have better learning characteristics and that batch normalization did not improve the performance of ELU networks.

To the best of my knowledge, the effect of a different weight updation schedule to every layer of a neural network has not been investigated in the current literature.

Approach

Activation functions determine the input to a particular layer of a neural network. Forming and choosing appropriate activation functions is crucial to train a good network. Among several candidates for the activation functions, ReLu^[1] (Rectified Linear Unit) , Leaky ReLu^[2], Softplus^[3] and Elu^[4] (Exponential Linear Units) have been investigated in this project (Table 1) . These activation functions were incorporated in both fully connected neural networks and convolution neural networks which were coded from scratch in python. This was done to allow for flexibility in the experimentation process to achieve the objectives of the project. The interactions of these activation functions with techniques like batch normalization^[5], dropout^[6] and different techniques of hyperparameter update like using update ‘schedules’ wherein some neurons have their weights updated periodically in place of every iteration were also explored. The dataset used is CIFAR – 10. The dataset has been divided into 49,000 images for training, 1000 images for validation and 1000 images for testing.



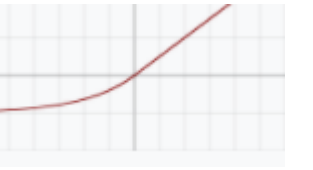
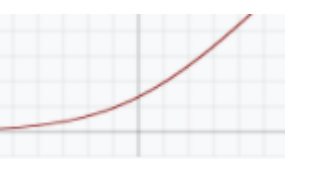
Activation function	F(x)
ReLu	Max (0,x) 
Leaky ReLu	Max (0.01x ,x) 
Leaky ReLu With varying slope	αx for $x < 0$ x for $x \geq 0$ $\alpha = 0.2, 0.25, 0.3$
ELu	$0.1 (e^x - 1)$ for $x < 0$ 1 for $x \geq 0$ 
Softplus	$\log (1 + e^x)$ 

Table 1 (images from Wikipedia)^[8]

Experiments

Initially, a fully connected five layer neural network with 100 neurons in each hidden layer has been used. The update rule is Stochastic Gradient Descent. The network was trained for 20 epochs with a batch size of 250 images.

The preliminary network has been deliberately kept simple in order to judge the merits of the activation functions without enablers like dropout, sophisticated update rules like ADAM and data normalization techniques like batch normalization. A total of 13 models were compared with the same network architecture as described earlier. For models with Elu

activation functions, the learning rate was varied in the range of $[1 \times 10^{-2}, 7 \times 10^{-2}]$.

The results for the best model for each activation function were obtained for the training and validation accuracies and tabulated as below.

Model	Validation acc. %	Testing acc. %
ReLU	51.3	47.0
Leaky ReLU, $\alpha = 0.01$	49.0	47.8
Leaky ReLU, $\alpha = 0.20$	52.7	48.1
Leaky ReLU, $\alpha = 0.25$	51.5	50.4
Leaky ReLU, $\alpha = 0.30$	52.4	49.4
Softplus	50.2	48.1
ELU	51.7	49.1

Table 2

It can be seen from Table 2 that the validation accuracy for model using Leaky ReLU with α (slope parameter) = 0.25 is much better than using ReLU. It can be seen from Table 2 that Leaky ReLU, Softplus and ELU activation functions are definitely better than plain ReLU function. Leaky ReLU with $\alpha = 0.25$ has a testing accuracy which is 3.4 % better than ReLU activation function.

In the next step, normalization technique like BatchNorm, optimization techniques like ADAM and regularization technique like dropout were applied to these fully connected neural networks. Hyperparameter optimization included experimenting with different learning rates and dropout fractions. Dropout rate of 0.15 was found optimal over a broad range. After experimenting over 43 models, the best results for each activation functions were obtained as follows :

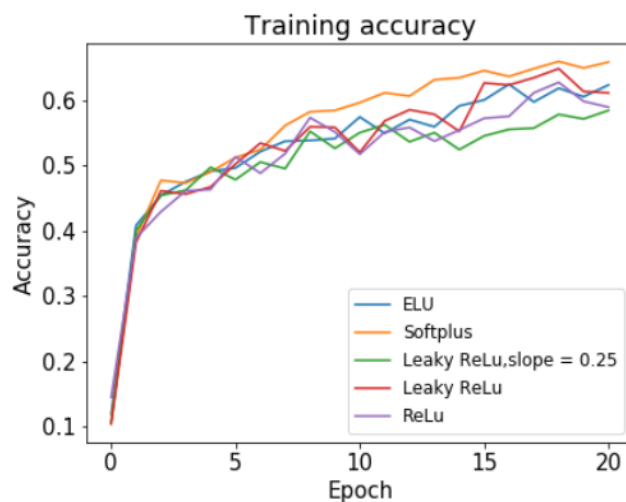


Figure 1

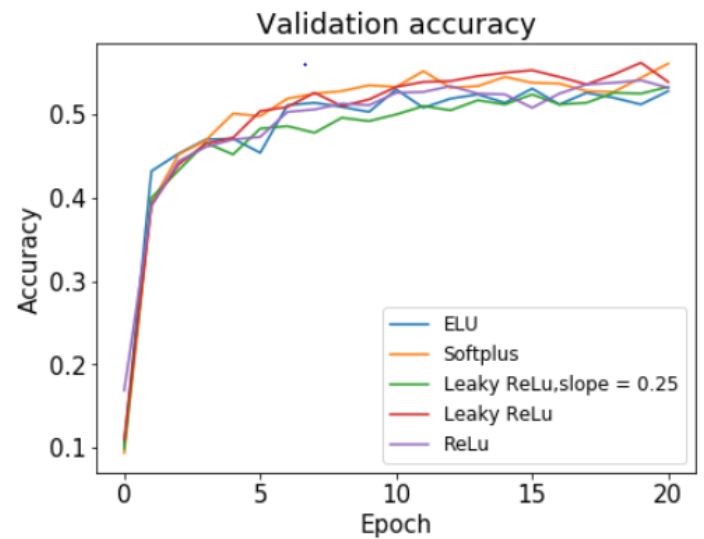


Figure 2

Model	Validation acc. %	Testing acc. %
ReLU	51.9	50.8
Leaky ReLU, $\alpha = 0.01$	55.8	55.0
Leaky ReLU, $\alpha = 0.25$	53.3	52.5
Softplus	56.1	54.7
ELU	52.4	49.1

Table 3

As evident from the results obtained in figures 1,2 and table 3 incorporation of the aforementioned techniques results in a substantial improvement in the performance of these networks. A notable exception is the ELU activation function which seems to resonate the claim of Clevert et al ^[4] that batch normalization does not seem to improve the performance of ELU networks. Also as seen in figure 1, the Softplus activation function seems to converge a lot faster as compared to other activation functions. In the results Leaky ReLU with a slope of 0.01 seems to perform the best among the other activation functions in consideration followed closely by the Softplus activation function.

In the next part, these activation functions were used in a Convolution Neural network. The network was hand coded from scratch and had 3 convolution-non linearity-2x2 pooling layers followed by 2 affine- non linearity layers and ending with a affine-softmax function for classification. The non-linearities investigated were all functions that were used previously. After optimizing over a wide range, a filter size of

3 and 16 filters were used in each convolution layer. The following results were obtained :

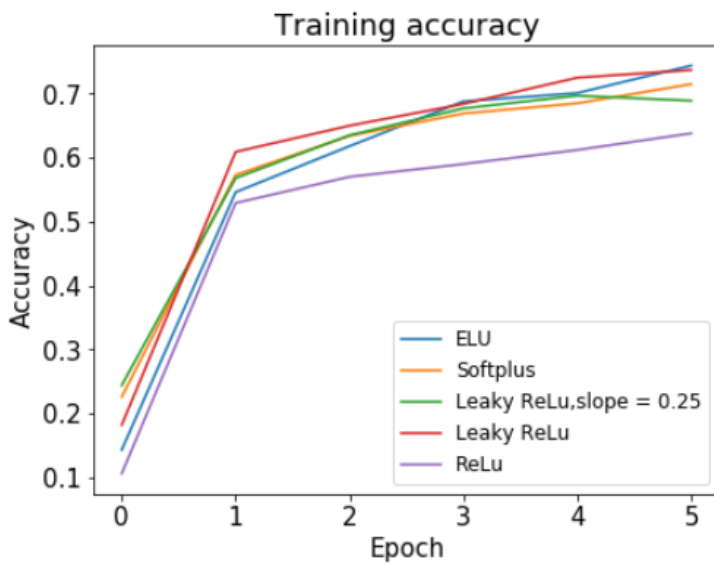


Figure 3

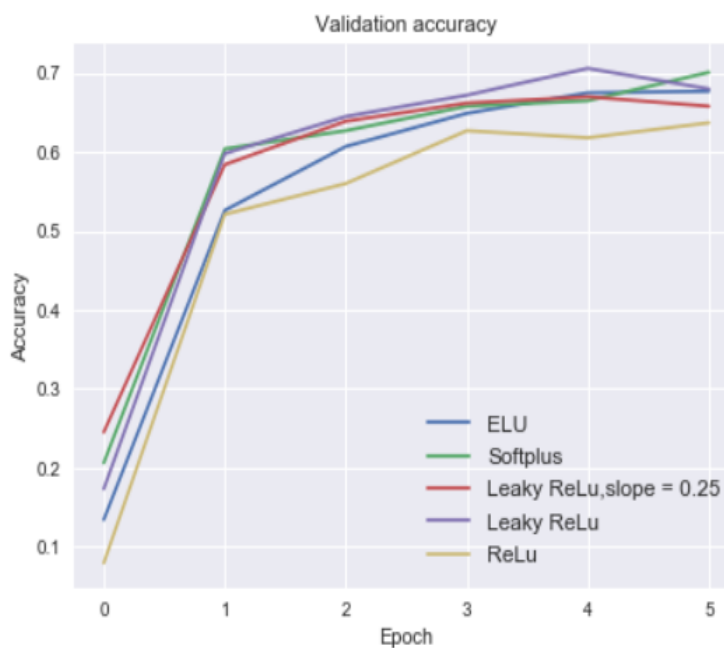


Figure 4

As seen in figures 3 and 4 , the leaky ReLU activation function performs much better than the other activation functions on the training set as well as the validation set. It is also interesting to note that all other activation functions performed much better than the normal ReLU activation function in convolution networks as compared to their relative performance vs the normal ReLU activation function in the fully connected networks.

Effect of a random update schedule

In the dropout technique, randomly selected neurons from a layer are 'dropped' out of the network according to a given dropout probability. This gives rise to an 'ensemble' of neural networks improving the generalization performance.

Instead of switching off entire neurons, I have tried to investigate the effects of updating the weights of a particular layer according to different probabilities for each update. Hence although the forward pass and backward pass of the gradients flows freely throughout the network, the weights for a particular layer are only updated with a certain probability every time.

A fully connected 5 layer neural network with ReLU non-linearity was chosen. The weights of these layers were randomly updated according to a set probability. So for example, after a particular iteration the weights for layers 1,3 and 5 could be updated while they may not be updated for layers 2 and 4.

Three variations of these networks were chosen. In the first network , for every layer, the weights were only updated 50% of the time. In the second network the layer weights had a 75% chance of update every time and the third network was a regular network in which the weights were updated every single time.

As seen in figure 5 , updating weights according to a random update schedule adversely affects the performance of a neural network. Generally speaking, more the chance of an update after every iteration, better is the performance of a network, with a 100% chance of an update performing the best.

I tried to establish the reason for the inferior performance of the network in case of a random update schedule. Two networks were trained : one with a 100% chance of a weight update which was trained for 20 epochs and the other network with a 50% chance of an update which

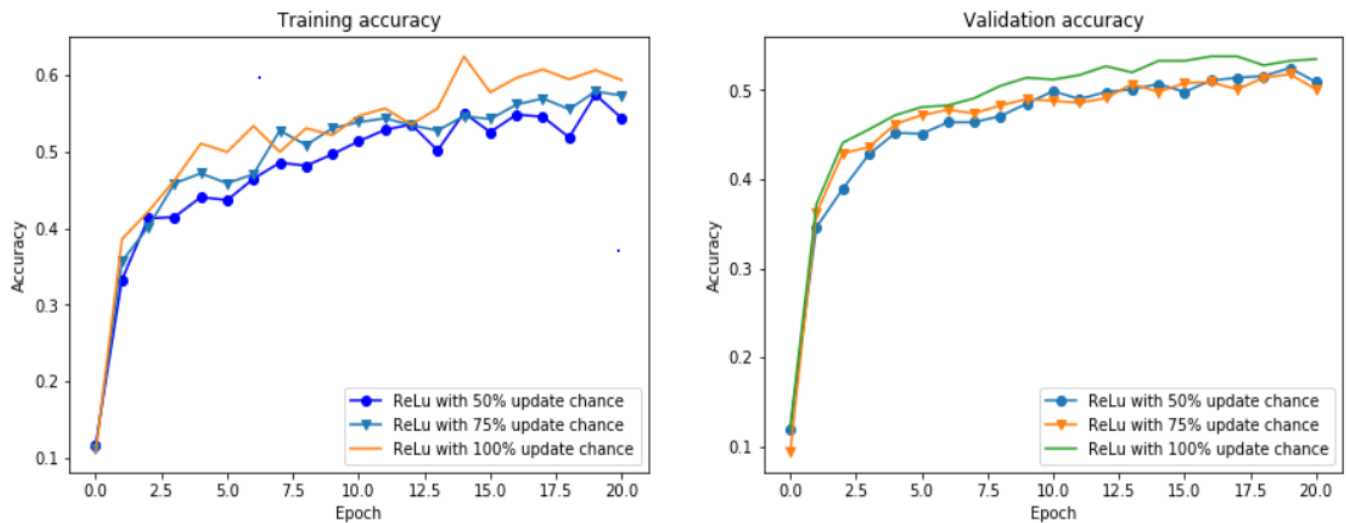


Figure 5

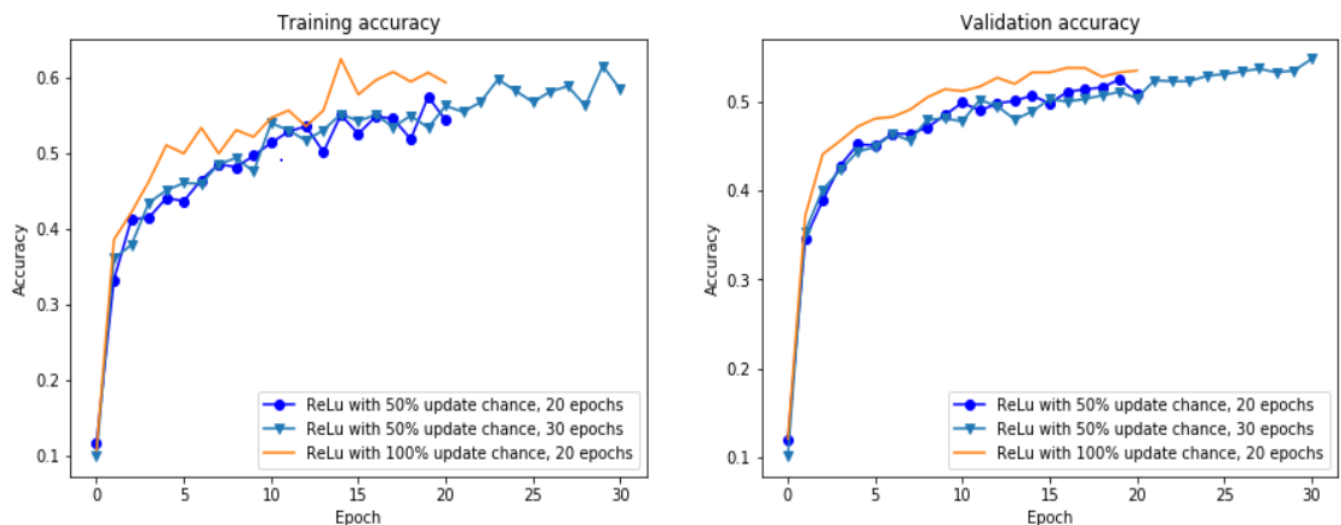


Figure 6

was trained for 30 epochs. As established above the network with a 50% chance of an update performs inferiorly as compared to the network with a 10% chance of an update. But once it is trained for more epochs (30 in this case), figure 6, it 'catches' up with the better network. This seems to suggest that randomly not updating the weights of a particular network slows the learning of the network.

Hence it can be concluded that introducing a random weight update schedule for every layer in a network hampers the performance of the neural network.

Conclusion

After performing numerous experiments with various activation functions like ReLu, ELU, Softplus and Leaky ReLu across various network configurations it can be concluded that better activation functions are available than the standard activation function ReLu. The choice of the activation function, of course depends on the network configuration. For example, the experiments indicate that for a fully connected neural network the Softplus activation function appears to give the best results while for the convolution neural networks the leaky ReLu activation function seems to be a favorable choice.

Also as expected, techniques like Batch normalization and dropout enhance the

performance across all activation functions as seen in table 3.

The experiments involving the introduction of a random weight update schedule proves that introducing an element of chance in updating the weights of a particular layer after every iteration in the neural network diminishes the performance of the neural network considerably. Hence it is advisable to update the weights after every iteration as is the established practice.

[8] https://en.wikipedia.org/wiki/Activation_function#cite_note-12

References

- [1] Nair, Vinod; Hinton, Geoffrey E. (2010), "Rectified Linear Units Improve Restricted Boltzmann Machines", 27th International Conference on International Conference on Machine Learning, ICML'10, USA: Omnipress, pp. 807–814, ISBN 9781605589077
- [2] Maas, Andrew L.; Hannun, Awni Y.; Ng, Andrew Y. (June 2013). "Rectifier nonlinearities improve neural network acoustic models" (PDF). Proc. ICML. 30 (1).
- [3] Glorot, Xavier; Bordes, Antoine; Bengio, Yoshua (2011). "Deep sparse rectifier neural networks" (PDF). International Conference on Artificial Intelligence and Statistics.
- [4] Clevert, Djork-Arné; Unterthiner, Thomas; Hochreiter, Sepp (2015-11-23). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". arXiv:1511.07289
- [5] Sergey, Lofe; Szegedy, Christian (2015-02-11). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". arXiv:1502.03167
- [6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; (2014-06-15). "Improving neural networks by preventing co-adaptation of feature detectors". arXiv:1207.0580
- [7] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2015-02-06). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". arXiv:1502.01852