Deep Learning with Exponential Linear Units (ELUs)

Sagar Patni, Prashant Gonarkar

Abstract— The paper 'Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) [1] by Djork-Arn´e Clevert, Thomas Unterthiner & Sepp Hochreiter introduces an activation function ELU which provides a significant speed up in learning rate over traditionally used activation functions (ReLU, eReLU, lReLU) by alleviating the bias shift effect and pushing the mean of the activation function to zero. In the experiments section the paper proposes,'ELUs lead not only to faster learning, but also to significantly better generalization performance than sReLU and lReLUs on networks with more than 5 layers'. In this project we examine the mathematical expressions and properties of different activation functions and we try to reestablish the results achieved in this paper by training a 6 Layer feed forward neural network classifier on MNSIT dataset using different activation functions.

Index Terms—Neural Networks, Deep Learning, Activation Function, ELU, MNSIT, Tensorflow.

1 Introduction

Increasing the speed of training the neural network is always been challenging problem. There has been multiple ways through which the network training is improvised. One of such method is defining a good activation function. In this paper we will examine one of such approach proposed by Djork-Arn´e Clevert, Thomas Unterthiner & Sepp Hochreiter [1] which uses a new activation function called Exponential Linear Unit (ELUs) while training. Currently the most popular activation function that has been used by researchers is ReLU which alleviates the vanishing gradient problem as it's derivative is one for positive values in not contractive. However, ReLu's are defined positive values hence they have mean activation larger than zero.

The new proposed ELU's overcome the vanishing gradient problem as their positive part is the identity function just like ReLUs, LReLU and SReLU. The negative part in the ELU's push the mean activation towards zero which in turn resemble more natural gradient, hence the speed up in learning. We are going examine the speed and accuracy of ELUs on MNIST dataset.

Rest of the Paper is organized as follows Section 2 gives background on neural network and activation functions. Section 3 describes the newly proposed activation function ELU and idea behind it. Chapter 4 talks about implementation and setup of experiments and chapter 5 gives interpretation of experimental results. Finally, last chapter 6 concludes the paper with closing remarks.

2 Background

2.1 Neural Networks:

An Artificial Neural Network (ANN) is a computational model motivated by the way natural neural systems in the human brain works. The basic unit of the neural network is neuron. The working of ANN takes its roots from the neural network residing in human brain. They operates on hidden State which are transient between input and output layers.

One of the key aspect of a neural network is its ability to learn. A neural network is not only a complex system, but adaptive system too, meaning it can change its internal structure based on the information flowing through it. Internally, this is achieved through the adjusting of weights.

Neural systems are organized as a series of layers, each made out of at least one neurons. Every neuron delivers a yield, or activation, in view of the yields of the past layer and set of weights.

Simulated Neural Networks have produced a great deal of fervor in Machine Learning exploration and industry, on account of numerous leap forward outcomes in discourse acknowledgment, PC vision and content handling.

Some standard uses of neural networks in software today are Pattern Recognition, Time Series Prediction,Signal Processing,Anomaly Detection.

2.2 Backpropogation Algorithm:

Backpropogation algorithm is one of the several ways in which an artificial neural network (ANN) can be trained.

In Backpropogation, Initially all the weights in neural network are randomly initialized. For every input in the training dataset, weighted sum of output is calculated along with bias & ANN is activated and its output is observed. This output is compared with the given output that we already know.

Error is computed based on this and it's propagated back to the previous layer. This error is taken into consideration and the weights are "adjusted" accordingly. This process is iterated until the output error is below a predetermined threshold value.

2.3 Activation Functions:

Activation functions are one of the key concept ine neural

network. Generally smoother & differentiable functions are choosen for activation function which makes them ideal for gradient based optimization algorithms.

2.3.1 Need of activation function:-

A typical artifical neural network calculates weighted sum of it's input along with bias and then pass it on next level.

$$Y = \sum (weight * input) + bias$$

As the output Y could be of any value between –infinity to +infinity, so we don't know which neuron to activate. Hence we need a decider function which would give the metric for the activation of neural network.

Some of the popular activation functions are

2.3.2 Sigmoid

The sigmoid function maps the output of the neuron within range [0,1].

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

The sigmoid activation function is monotonous, and it has a positive first derivative which makes it suitable function to use with neural network. The sigmoid function suffers from the vanishing gradient problem which slows down the learning.

2.3.3 Rectified Linear Unit (ReLUs):-

$$f(x) = \max(0, x)$$

The ReLU activation function maps the output of neuron to true value if the the value is greater than zero and it maps all the negative output values to zero. The ReLU activation function address vanishing grading and exploding gradient problems while using sigmoid function.

2.3.4 Softmax

The output of the softmax function gives us categorical distribution, it can be used as a output function to classify the categorical data.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

3 Exponential Linear Units

3.1 Problems with existing activation function:

Current most populor choice of activation function is ReLU which was originally proposed for restricted boltzman machines & then later used for neural networks. One of the major advantage of using ReLUs was it avoid vanishing gradient problem which the previously popular activation fuction is sigmoid suffers from, as the derivative of ReLU is 1 for positive values. But ReLU's are non-negative and their mean activations is not close to zero but greater than zero.

It has been learnt that less bias shift moves standard gradient closer to the natural gradient and fasten the learning process. Hence the activation function should push the activation means closer to zero in order to decrease the bias shift effect. To achieve decrease in bias shift effect following solutions were proposed.

1. *Centering the activations to zero so that off-diagonal entries of fisher information matrix is small [8]*
2. *Using batch normalization which also centers the activation [9]*
3. *Projected Natural Gradient Descent algorithm centers activation by whitening [10]*

Researchers [link] proposed new activation function Exponentail Linuear Unit (ELU) which can return negative values, but quickly saturates over the negative values.

3.1 ELU:-

The exponential linear unit is defined as

$$f(x) = \begin{cases} x & if\ x > 0 \\ \alpha(e^x - 1) & if\ x \leq 0 \end{cases}$$

where α controls the value to which the activation function saturates for negative inputus.

The function maps output of neuron to it's true value when the value is greater than zero. It maps the value to an exponential value, controlled by hyper parameter a when the output of neuron is negative value. The ELU function try to make means activation function to zero which speeds up the learning.

ELU's overcome the vanishing gradient problem as their positive part is the identity function just like ReLUs. The negative part in the ELU's push the mean activation towards zero which in turn resemble more natural gradient, hence the speed up in learning. ELU saturates to a negative value with much smaller arguments and the saturation decreases the variation of the units if deactivated, hence ELUs can encode degree of presence of input concepts, but they do not quantitatively describe the degree of absence. Because ELU's saturate the negative values leading to smaller gradient, they are most noise robust to LeakyReLUs and LeakyReLUs.

4 Implementation

4.1 Building neural network from scratch :

To get understanding of how internally the neural network works we created a vanilla implementation of the neural network in python numpy.

The algorithm performs below taks

1. Forward pass
2. Calculate the error in the prediction.
3. Backpropagation

In step 1, Each hidden unit takes the sum of its inputs * weights (z) and passes that through the activation functions(a). The data and the weights of the network are used to compute a prediction (y).

In step 2, the error is computed based the true labels provided by user and prediction computed in step 1.
In this experiment we used root mean squared loss function,

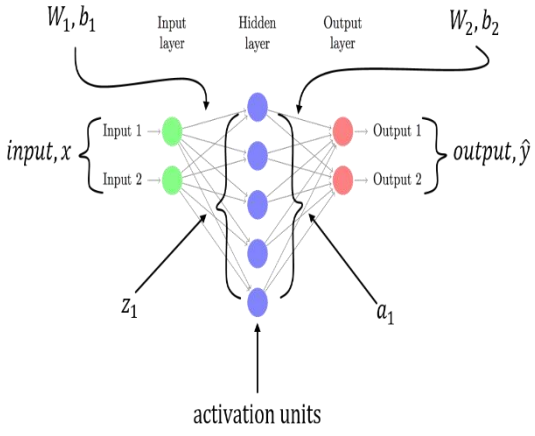


Fig. 1 Neural Network
(Source: https://i.stack.imgur.com/iHDtO.png)

In step 3, the weights in each layer are adjusted based on error computed in step 2. The derivative of sigmoid at output layer is calculated to determine the value of slope and that is multiplied by the error. This give the magnitude of the error and the direction in which the weights needs to be changed.

The equations for backpropagation algorithm, are given below and the mathematical foundations and current improvements to the algorithm can be found at[5]

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$$\frac{\partial C}{\partial w^l_{j\,k}} = a^{l-1}_k \delta^l_j$$

4.2 Neural Networks using TensorFlow:

The aim of this project is to re establish the result presented in ''. In previous sections we developed multi layer basic neural network from scratch to get intuition of neural network working in forward pass and backward pass. We got similar performances using the neural network we built. But in order to standardize our result in this we leveraged deep learning library tensorflow. And we used mnist dataset provided by tensorflow

from tensorflow.examples.tutorials.mnist import input_data
mnist=input_data.read_data_sets("/tmp/data/", one_hot=True)

Tensorflow has in buit implementations of activations function ELU which is inspired from the same research paper we used for experimentation in this project.

The tensorflow works by creating a dataflow graph of the neural network functions. In the graph each node is represented as 'tensor' where the tensor is basic unit of tensorflow. First we defined hyperparameters of the network and we experimented with them to get best results. Below are the hyperparameter we used in our final experiment. The configuration we have used for building the neural network in tensorFlow framework is as follows:

learning_rate = 0.01
training_epochs = 300
batch_size = 64
hidden_layers = 8
units = 128

We defined placeholder for out data input. Placeholder's are data units in tensorflow which can be initialized when tensorflow program runs. We defined tensorflow variables for each of our weight and bias vectors with a sizes of 784*128 for input layer, 128*128 for each hidden and 128*10 for the output layer. Then we built our neural network using mathematical forms explained in forward propagation in previous section.

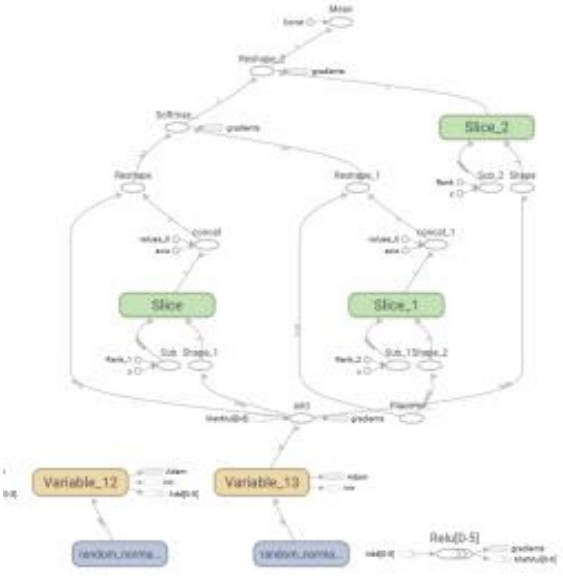


Fig. 2 dataflow graph of the program

We used softmax cross entropy function with root mean squared error as a loss function for our output layer, this function calculates error at the output layer converting

the output into categorical distribution.

softmax_cross_entropy_with_logits(
_sentinel=None,
labels=None,
logits=None,
dim=-1,
name=None)

We used Adam optimizer to update the network weights in an iterative fashion. In the paper[5] adam algorithm was applied to the logistic regression algorithm on the MNIST character recognition, a Multilayer Perceptron algorithm on the MNIST dataset. They conclude that "Using large models and datasets, we demonstrate Adam can efficiently solve practical deep learning problems."

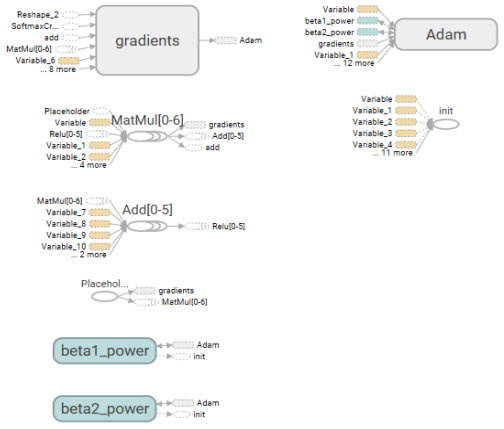


Fig. 3 dataflow units of the program

The tensorflow program can be run by establishing a session and calling session.run method providing the tensor of the output function. The tensorflow automatically finds the dependencies required to compute the output function from the dataflow graph and execute those. We ran our optimization for the epoch of 300 and calculated total running time and the cost corresponding to each epoch.

4.3 MNIST Dataset:

MNIST [4] is well know dataset of handwritten digits and widely used by researchers for pattern recognition methodoliges. The dataset contains handwritten images of 0 to 9 digits along with their labels. It has training set of 60,000 samples and test set of 10,000 samples. Each image in dataset is 28 pixel by 28 pixel and contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm.

5 Experimental Setup:

We have used Python interfacing with TensorFlow framework. We built three different networks each separate for ELU, ReLU and sigmoid as activation function and measured the results.

One of the goal was to examine whether the mean activation keep closer to zero than other units. We trained the neural network model on MNIST digit classification dataset with ELU and the unit activation was tracked in each hidden state. The network was built with eight hidden layers which consist of 128 units each. We have trained this network for 300 epoches by stochastic gradient descent with learning rate 0.01. The batch size was kept 64.

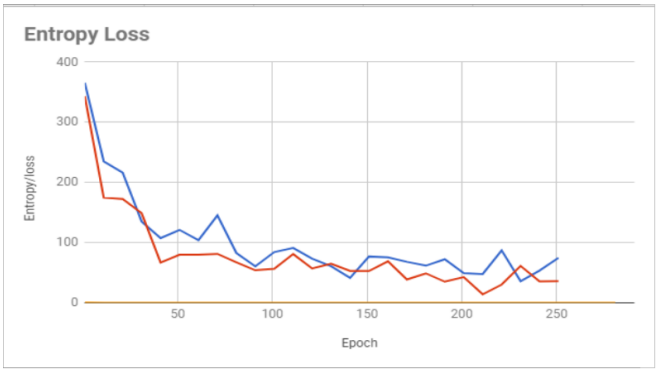
6 Results

The results of our experiment clearly shows that the learning time for ELU is least and the accuracy is highest among other activations.

300 epoch	ELU	ReLU	Sigmoid
Accuracy	0.9609	0.9577	0.936
Time(sec)	798.123498	897.79063	857.382674

Time and Accuracy

Below is the graph for number epochs vs loss, We can see that the ELU (red) converges much faster, we can clearly see the vanishing gradint issue with the Sigmoid(yellow) as the loss remains constant over the time.



Number of epochs vs loss

7 Conclusion

The newly introduced exponential linear units (ELUs) perform faster and precise on neural networks. This can be well evident on previous result section. Hence the propostion that ELUs allows the network to push the mean activations closer to zero which decreases the gap between the noramal gradient and the unit natural gradient fasten the learning holds true.

8 Acknowledgment

We wish to thank Prof Baoxin Li for the valuable guidance throughout the course and authors of the paper [1].

9 References

[1] Djork-Arn_e Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015.

[2] W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)

[3] https://arxiv.org/abs/1412.6980" Diederik P. Kingma, Jimmy Ba

[4] MNIST http://yann.lecun.com/exdb/mnist

[5] "https://link.springer.com/chapter/10.1007/978-3-642-30223-7_87"

[6] https://www.tensorflow.org/get_started/mnist/mechanics

[7] Vinod Nair Geoffrey E. Hinton http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.6419&rep=rep1&type=pdf

[8] Raiko, T., Valpola, H., and LeCun, Y. Deep learning made easier by linear transformations in perceptrons. In Lawrence, N. D. and Girolami, M. A. (eds.), Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS12), volume 22, pp. 924–932, 2012.

[9] LeCun, Y., Kanter, I., and Solla, S. A. Eigenvalues of covariance matrices: Application to neural-network learning. Physical Review Letters, 66(18):2396–2399, 1991.

[10] Desjardins, G., Simonyan, K., Pascanu, R., and Kavukcuoglu, K. Natural neural networks. CoRR, abs/1507.00210, 2015. URL http://arxiv.org/abs/1507.00210.