

Planning and Risks

Exam Number: B032374

February 21, 2017

1 Introduction

Investigate the effect of activation function on a multi-layer neural network.

This part of the report investigates how the choice of activation function affect-training set performance of a small multi-layer model. Three transformation functions with different characteristics were tested and evaluated. Additionally

How does the number of hidden layer and hidden units affect the training of a deep neural network?

2 Methodology

As the methodology used for the three experiments were similar, this section will cover the overall structure of the code and neural networks used throughout the investigation. Instead of building three isolated models and corresponding methods to run the experiments, some effort was put into building a system that would accommodate all the intended trials.

By looping over arrays of experimental parameters, the actual experiments were easily controlled and initiated once the necessary code was in place.

2.1 The Core Model

A simple three-layer model, interleaved with non-linear transformations, was chosen as a basis for this study.

To investigate the aforementioned questions, only specific aspects of the model, or training procedure, were changed during the experimentation. The weights for the fully connected layers were initialized by a random distribution, using the `tf.truncated_normal` function from TensorFlow. Then the biases are initialized with `tf.zeros` to ensure they start with all zero values, and their shape

is simply the number of units in the layer to which they connect. A simple schematic representation of the model can be seen in Figure 1.

The model was trained on image classification, using labeled images from the CIFAR-10 data set. To accommodate the CIFAR-10 image size of 32x32, and three color channels, the model's input dimension was 3x32x32. Furthermore the base model had 200 hidden units in the hidden layers, and a 10 dimensional output layer (10 classes). A cross entropy error function was used with the model, which was implemented using `softmax_cross_entropy_with_logits` from TensorFlow.

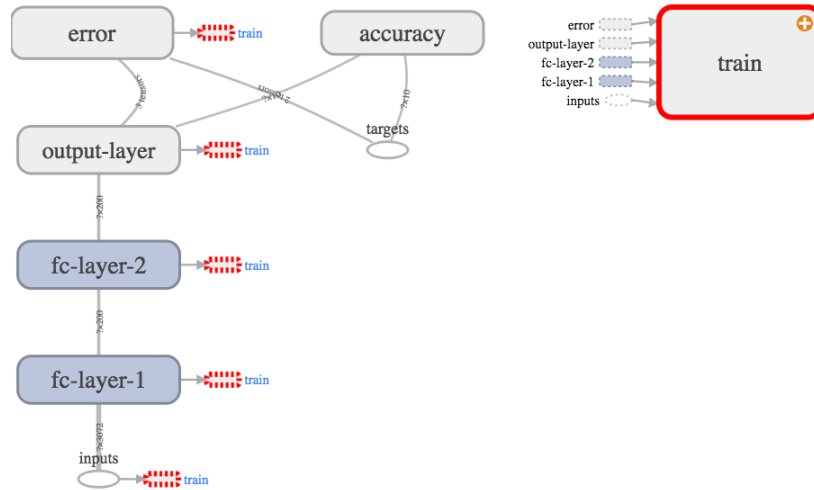


Figure 1: Showing a visual representation of the model used, from TensorBoard.

2.2 The Training

The model was trained for 40 epochs, or 32,000 steps, with batch size of 50, in all three experiments. The `AdamOptimizer` was used as the default optimizer, with a learning rate of 0.01. The Summary Operator from TensorFlow was used to record the data from the experiments. The training data was recorded every step, whereas the validation data was recorded every 100 steps.

2.3 The Experiments

2.3.1 Activation Functions

Four activation functions were compared. The logistic sigmoid $f(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent $f(x) = \tanh(x)$ the rectified linear (ReLU) $f(x) = \max(0, x)$ and the exponential linear unit (ELU):

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

where α is a hyperparameter that decides the range of negative values for which the activation function saturates.

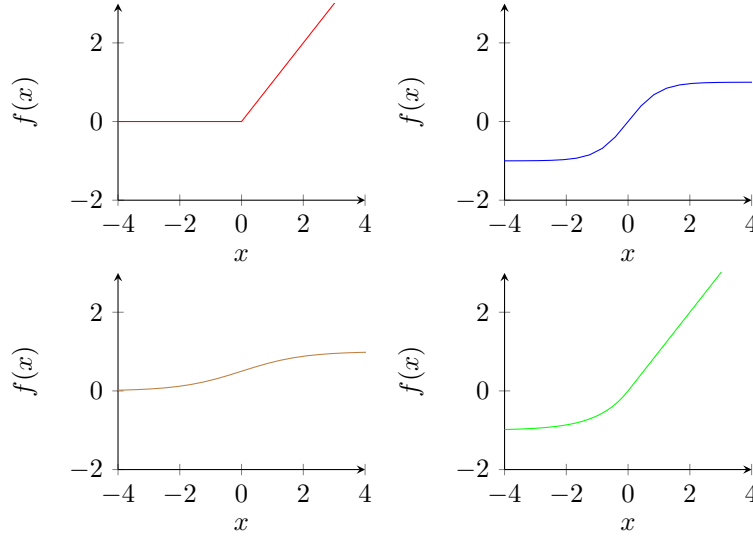


Figure 2: Showing the plots of the activation functions. Top Left: Relu. Top Right: Tanh. Bottom Left: Sigmoid. Bottom Right: ELU

Both the hyperbolic tangent and the sigmoid functions compress their input values between $(-1, 1)$ and $(0, 1)$ respectively. A result of such saturation will cause their gradients to approach zero, which means smaller parameter changes and slower training. Two key differences between the two are that the hyperbolic tangent allows for both positive and negative outputs, and that the hyperbolic tangent is centered around zero. The sigmoid transformation function is centered around 0.5, which is not ideal when training models.

The ReLU transformation function looks and behaves quite differently from the two before mentioned alternatives, see Figure 2. The ReLU function has a constant gradient of 1 for activations greater than zero, and it will not suffer

from diminishing gradients. It can however kill all the units, by responding to everything with 0. Furthermore, as it is unbounded, it will be more sensitive to learning rates than the sigmoid and hyperbolic tangent.

In addition to ReLu, the Exponential Linear Unit (ELU), was also tested. Although unlike ReLu, the ELU responds to negative values, which pushes the mean of the activations closer to zero. Mean activations that are closer to zero enable faster learning as they bring the gradient closer to the natural gradient [1].

The four activation functions were used in turn to train the three-layer model presented above. The only difference, the new activation function would override the default (ReLu) non-linearity.

2.3.2 Hidden Layers

depths 1 2 4 5 widths 50 100 200 400

2.3.3 Learning Rate Schedules

The third experiment investigates the implementation of a time-dependent learning rate schedule. This report uses an exponential learning rate schedule

$$\eta(t) = \eta_0 \gamma^{(t/t_{decay})} \quad (2)$$

where $\eta(t)$ is the learning rate after t steps, η_0 the initial learning rate and γ is the decay rate. t_{decay} is the number of steps required for one decay-cycle. The value of γ and t_{decay} determine the rate of decay of the learning rate.

The learning schedule was implemented using already built functionality from TensorFlow. The optimiser updates the step, which changes the learning rate, which goes back into the optimiser. The `tf.train.exponential_decay`

The decay rate and the decay steps were only changed during some initial testing, and were set to 0.96 and 10,000 respectively. The initial learning rates used were 0.02, 0.03, 0.05, 0.07 and 0.1.

3 Results and Discussion

3.1 Activation Functions

After the model was trained with all the activation functions, the ELU performed significantly better than the other activation functions. The final classification error for ELU was 0.96, whereas the hyperbolic tangent performed

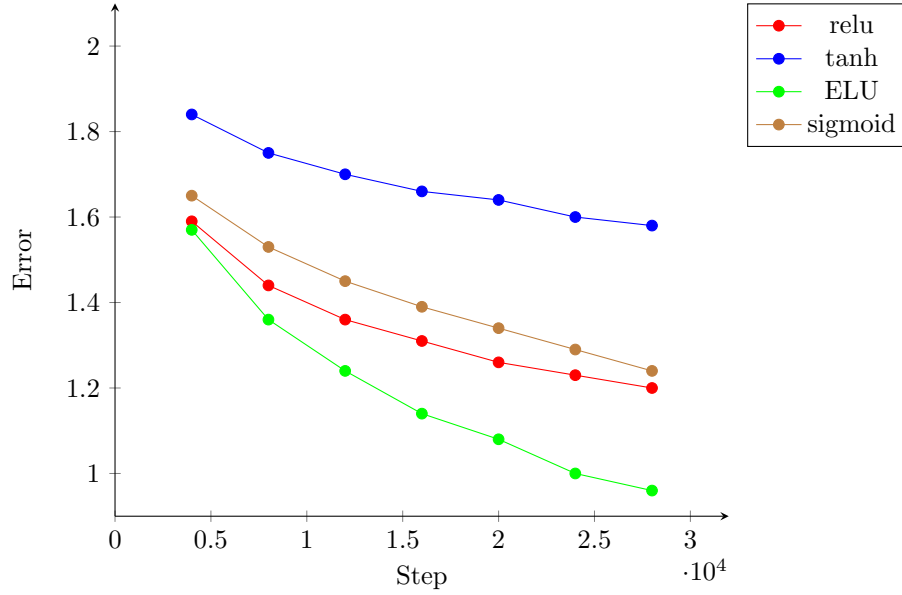


Figure 4: Showing the evolution of the error function values across the training steps, for the different activation functions.

Table 1: Displaying the final classification accuracies for the activation functions.

	Final training set accuracy	Final validation set accuracy
ELU	0.669	0.505
ReLU	0.553	0.483
Sigmoid	0.537	0.481
Tanh	0.472	0.401

scenarios with different models or learning rules, an initial trial using both non-linear functions is advisable.

3.2 Learning Rate Schedules

3.3 Hidden layers

write here

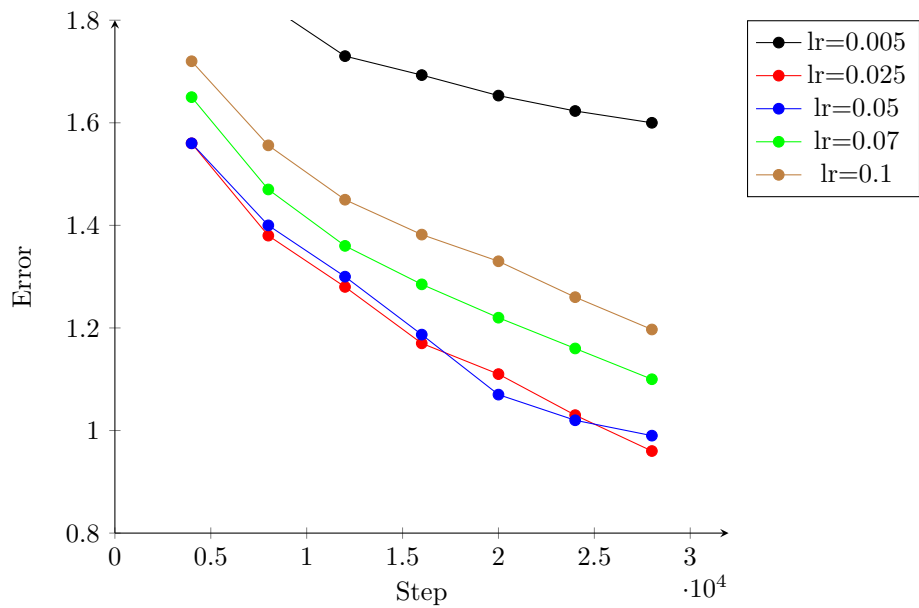


Figure 5: Showing the evolution of the error function values across the training steps, for some of the initial learning rates.

4 Conclusion

longer trials

5 Future Work

Convolutional neural network, drop out, max pool Continue to investigate ELU, in more complex models.

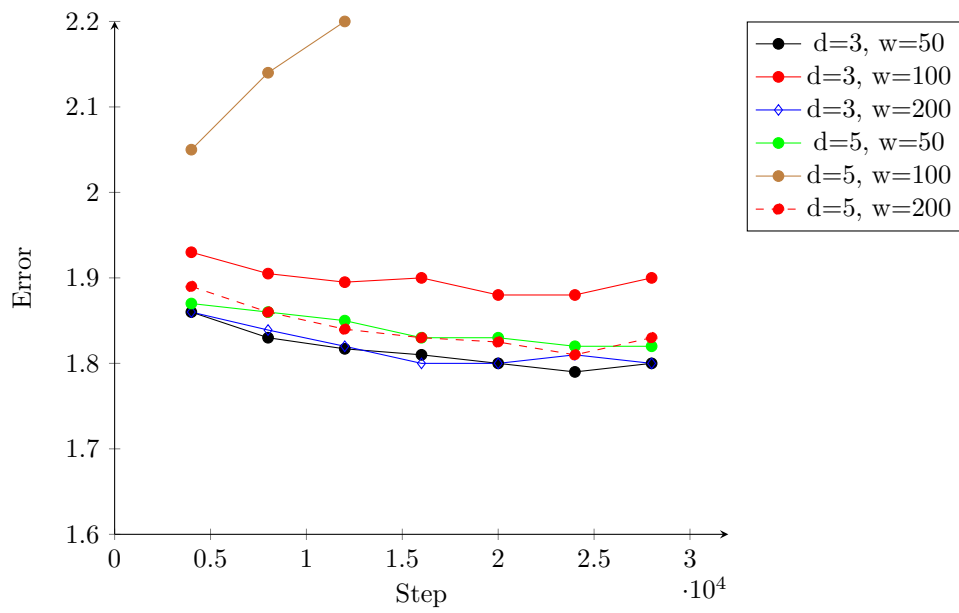


Figure 6: Showing the evolution of the error function values across the training steps, for some of the initial learning rates.

References

- [1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *CoRR*, vol. abs/1511.07289, 2015.