MLP Coursework 3

Eskil Joergensen

February 21, 2017

1 Introduction

The report aims to investigate, present and discuss three simple questions: How does the choice of activation function affect the performance of a multi-layer neural network? How does the number of hidden layer and hidden units affect the training of a deep neural network? Can a learning rate scheduler increase the performance of a simple feed-forward network? Using a simple model, CIFAR-10 image classification and TensorFlow, a number of small experiments related to the training of deep neural networks.

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.

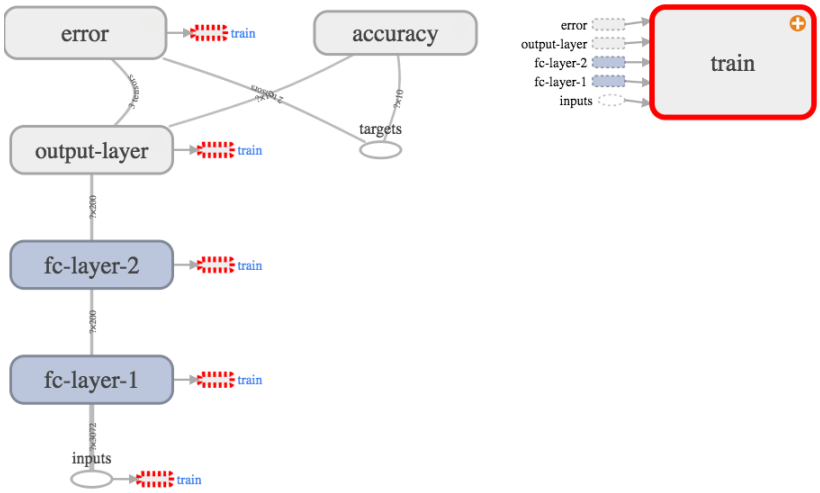

Figure 1: Showing a visual representation of the model used, from TensorBoard.

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):

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

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

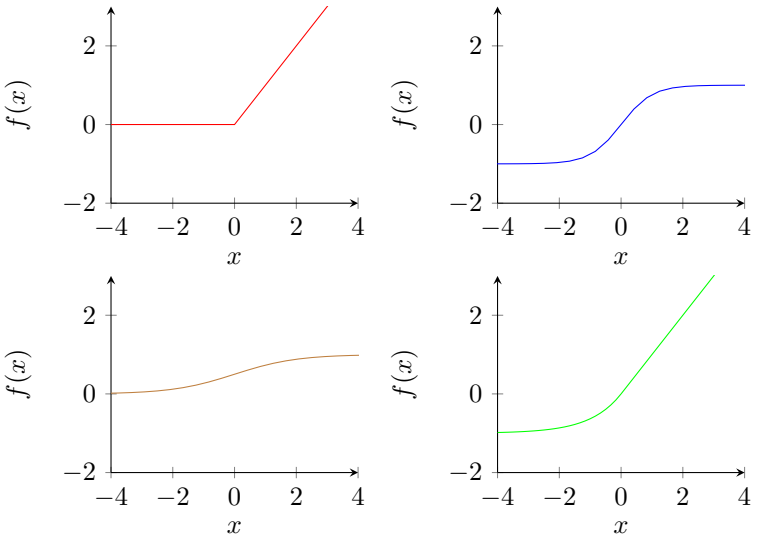

Figure 2: Showing the plots of the activation functions. Top Left: Relu. Top Right: Tanh. Bottom Light: Sigmoid. Bottom Right: ELU

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 may not be ideal.

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 their 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

The second experiment involved investigating how the arrangement of hidden layers and hidden units would effect the performance of the training of a model. A series of depths and widths were tested using two nested loops.

The main obstruction was creating a system that would dynamically allocate the desired number of hidden layers in the model. A dictionary was used to store the fully connected layers by their respective TensorFlow name scope identifiers. As the number of hidden units was the same for all the hidden layers, it was easily implemented.

A number of combinations of hidden units and layers were integrated into the base model. The number of hidden layers tested in the model were 1, 2 and 4, combined with 50, 100, 200 and 400 hidden units.

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

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

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. For the learning rate experiments, the `GradientDescentOptimizer` replaced the Adam optimizer.

The learning schedule was implemented using already built functionality from TensorFlow. In addition to minimizing the training error, the optimizer can also be used to control the learning rate, see Figure 3. The optimizer iterates over the steps taken, which changes the learning rate as the training progresses.

TensorFlow offers a number of learning rate schedulers, in this investigation the exponential version, `tf.train.exponential_decay`, was used.

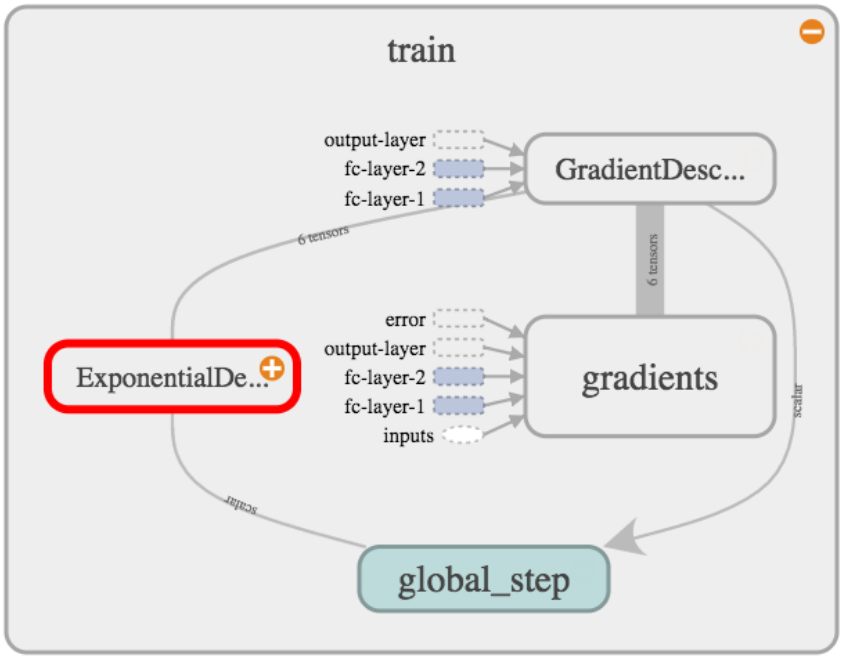

Figure 3: Showing a schematic representation of how the learning rate is updated within the graph, from TensorBoard.

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 0.005, 0.02, 0.025, 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, see Figure 4. The final classification error for ELU was 0.96, whereas the hyperbolic tangent performed worst, with a final classification error of 1.58. The training with ReLu and sigmoid showed similar results.

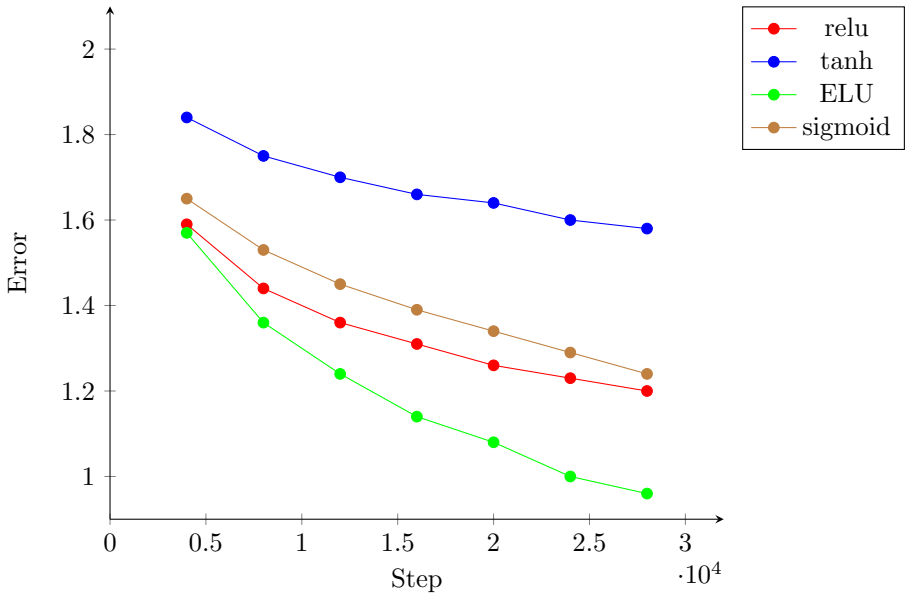

Figure 4: Showing the evolution of the error function values across the training steps, for the different activation functions.

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

The results from the training errors are also reflected in the final classification accuracies, see Table 1. The ReLu and the sigmoid performed very similarly. Having to choose between the two is more difficult, but taking into account that the ReLu is computationally more efficient, that should break the tie. However, in other scenarios with different models or learning rules, an initial trial using both non-linear functions is advisable.

Based on the results presented here, it is clear that the ELU activation function outperformed the other three. As the first use of the ELU was published quite recently, it is interesting to see how well it performs. Although the network used in this experiment was quite simple, the findings are consistent with those of the original publishers [1].

3.2 Hidden layers

For the second experiment, the results were mostly very similar. In the case of a single hidden layer, no interesting level of learning was observed. With a classification accuracy around 0.08, the three trials with a single hidden layer (the non-deep networks) were dismissed.

Besides the one outsider, the training data shows a very similar performance for remaining five configurations. The two best trials were both from the models with three hidden layers, see Figure 5.

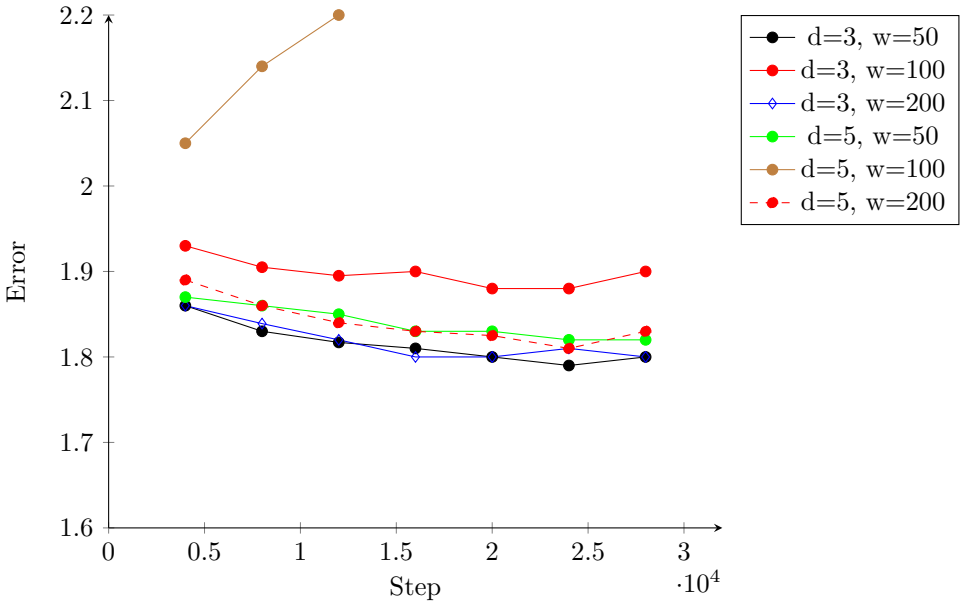

Figure 5: Showing the evolution of the error function values across the training steps, for different depths and widths of the hidden layers.

Even though the results may indicate better performance using fewer hidden layers, one must realize that additional hidden layers, may require other changes in the network or training process to benefit from the additional layers.

3.3 Learning Rate Schedules

Seven initial learning rates were tested. However, as the trials for 0.02, 0.025 and 0.03 were visually identical, the results from 0.02 and 0.03 have been omitted from this section. Figure 6 shows the relevant findings.

The results show that the training with the initial learning rate of 0.025 and 0.05 performed significantly better than the other initial learning rates (besides

0.02 and 0.03). As the error increases for learning rates lower than 0.025, as well as for learning rates over 0.05, it seems plausible that the optimal initial learning rate lies between 0.025 and 0.05 for the current model.

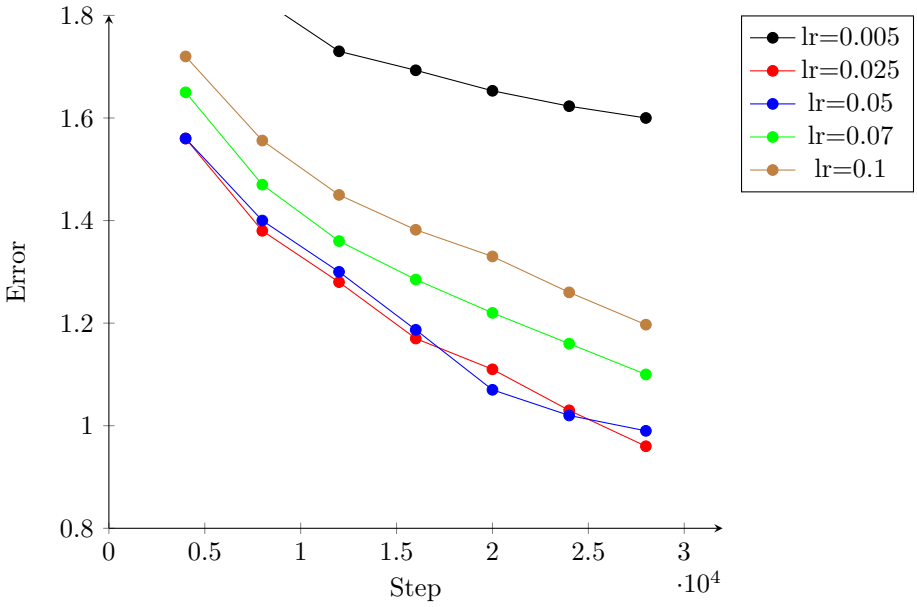

Figure 6: Showing the evolution of the error function values across the training steps, for some of the initial learning rates.

4 Conclusion

In conclusion, the results of this investigation were not revolutionary. Although the reason behind the simple feed-forward model was to more easily observe changes in performance, it may have been too simplistic. However, experimenting with the ELU activation function was interesting, and it will be interesting to see if will be adopted by the community.

Despite the lack of new revelations, having set up a system that allows for a versatile and dynamic way to set up and train models, may help further experimentation in the future.

5 Future Work

Following on the progress and findings of this investigation, the next step involves building on the current setup so it can handle more complex scenarios.

Initially a procedure for setting up and running convolutional neural networks will be implemented. Secondly, other complimentary features like drop out and max pooling are still needed, and will be even more relevant with convolutional neural networks.

Lastly, some investigation will go into the use of ELU as a part of a convolutional neural network.

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.