MLP Coursework 1: Activation Functions

s1749267

Abstract

This experiment tested different activation functions, depth of hidden layers, and various initializers of the deep neural network (DNN) in training work on MNIST dataset. We chose five kinds of activation functions: sigmoid, rectified linear units (ReLU), leaky rectified linear units (LReLU), exponential linear units (ELU), and scaled exponential linear units (SELU). Compared to sigmoid and ReLU, the other three activation functions had their advantages. Under that, the loss function and the accuracy ratio converged in fewer epochs in the training process. While the number of hidden layers was increasing from 2 to 8, the converging would finish in fewer epochs, but the error of validation set was growing up. We used six initialization strategies: Fan-in, Fan-out, Fan-in & Fan-out, and their Gaussian distribution version in training. From that, we obtain the result that the Fan-in spent fewest epochs to converge, but it led to the highest validation set error.

1. Introduction

Currently, there are many popular activation functions for deep neural networks (DNN). Sigmoid and rectified linear units (ReLU) were widely used in practice. Although ReLU has 6 times(Krizhevsky et al., 2012) faster the converging speed in Stochastic Gradient Descent (SGD) than sigmoid/tanh functions, ReLU could be dying during training – a large gradient would deactivate the neuron since the value of ReLU is zero while input is less than zero.(CS3) To avoid this problem, Leaky ReLU (LReLU)(Maas et al., 2013) was invited to set a small slope while the input is negative. Besides, in the propose for accelerating training speed and improve the performance, exponential linear units (ELU)(Clevert et al., 2015), as a mixture of ReLU and sigmoid, and scaled exponential linear units (SELU) to build a self-normalized network(Klambauer et al., 2017). We used these activation functions in the experiment to compare their behavior in training DNN.

To compare their behavior, we chose a di-hidden-layer and softmax-output-layer network to recognize numbers from MNIST 28*28 handwriting number dataset. We set the batch size to 100 for both training and validation sets in SGD. Other hyperparameters are 0.1 for the learning rate, 100 for the number of epochs, and 100 units for each hidden layer.

In this experiment, we'd like to know how the different activation functions influenced on training, how the performance evolved with the number of hidden layer, and how different types of initializers influenced on the performance.

2. Activation functions

The activation functions in this experiment we used are Sigmoid, Sigmoid and rectified linear units (ReLU), Leaky ReLU (LReLU), exponential linear units (ELU), and scaled exponential linear units (SELU). ReLU and its derivative can be simply expressed as:

$$\mathrm{relu}(x) = \max(0, x), \tag{1}$$

$$\frac{d}{dx}\mathrm{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{2}$$

Adding a small slope while the input is negative, we got LReLU:

$$\mathrm{lrelu}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0, \end{cases} \tag{3}$$

$$\frac{d}{dx}\mathrm{lrelu}(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{4}$$

In the experiment, $\alpha = 0.01$ was used.

Changing the negative part to an exponential function, the activation function became ELU:

$$\mathrm{elu}(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0, \end{cases} \tag{5}$$

$$\frac{d}{dx}\mathrm{elu}(x) = \begin{cases} \alpha \exp(x) & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{6}$$

To make the function smooth, we chose $\alpha = 1$. Multiplying a scale factor to ELU, we got SELU:

$$\mathrm{selu}(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0, \end{cases} \tag{7}$$

$$\frac{d}{dx}\mathrm{selu}(x) = \begin{cases} \lambda\alpha \exp(x) & \text{if } x \leq 0 \\ \lambda & \text{if } x > 0. \end{cases} \tag{8}$$

Suggested by the experiment guide, we used $\alpha = 1.6733$ and $\lambda = 1.0507$.

3. Experimental comparison of activation functions

In this part, we trained the model by using different activation functions introduced above and compared their

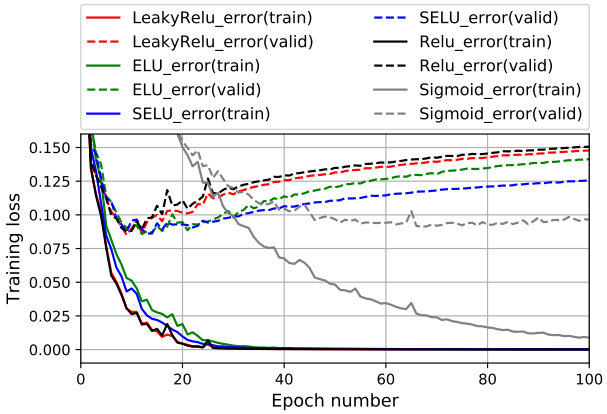

Figure 1. Training error vs Epoch number while using various activation functions

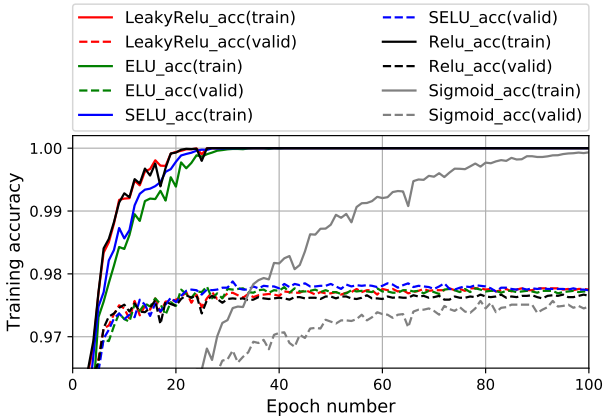

Figure 2. Training accuracy vs Epoch number while using various activation functions

behavior. As shown in figure 1, the model using sigmoid activation function did not converge within 100 epochs, and others had a better performance which converged within 40 epochs. Respecting the validation set, we find that the error of validation set was higher while the model converged faster. Compared to ReLU, LReLU had a similar converging speed, but lower validation set error, which means LReLU could overcome the overfitting issue more or less. As to ELU and SELU, in this experiment, SELU performed better. SELU had faster converging speed and lower variance than ELU model. However, ELU and SELU had smaller validation error although they spent more epoch to converge than ReLU and LReLU.

When comparing the accuracy of various models, SELU model had the highest accuracy, shown in figure 2. Compared to ReLU, all LReLU, ELU, and SELU had a higher accuracy of validation set. These three modified linear units could improve the accuracy.

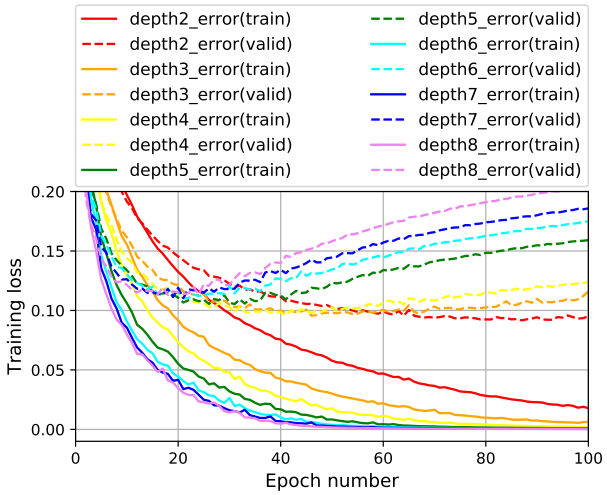

Figure 3. Training error vs Epoch number while using different number of hidden layers

4. Deep neural network experiments

From the last section, we found SELU activation function had an ideal performance in the MNIST task. In this part, we used SELU as activation function and tried different hidden layers quantities from 2 to 8, and then we observe the influence of the initializers in different types.

Firstly, we set the number of hidden layers from 2 to 8 and changed the learning rate to 0.01. The training curve shown in figure 3 could tell the converging speed increased with the increase of the number of hidden layers. The octo-hidden-layer network converged fastest, and the di-hidden-layer network converged slowest. However, the validation error had the opposite tendency – the deepest network had the largest validation error. Apparently, while adding parameters to a neural network, or even using a higher-order polynomial function to fit curves, it is more likely to overfitting the data, which cause the high validation error in large epoch number. In figure 3, we find that the gaps between two validation error curves were different. The difference between tetra-hidden-layer and penta-hidden-layer validation errors was relatively higher than other differences. For faster converging speed, to use 4 hidden layers was enough and more layers would cost higher variance. The tetra-hidden-layer model also performed well to give a high accuracy. In figure 4, there was not much difference among models with 2 to 5 hidden layers. The accuracy was around 0.974.

Then we tried to explore the behavior when choosing different initializers. We used six kinds of initialization strategies in training:

- Fan-in: $w_i \sim \mathrm{U}\left(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}}\right)$,
- Fan-out: $w_i \sim \mathrm{U}\left(-\sqrt{3/n_{out}}, \sqrt{3/n_{out}}\right)$,
- Fan-in and Fan-out: $w_i \sim \mathrm{U}\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right)$,

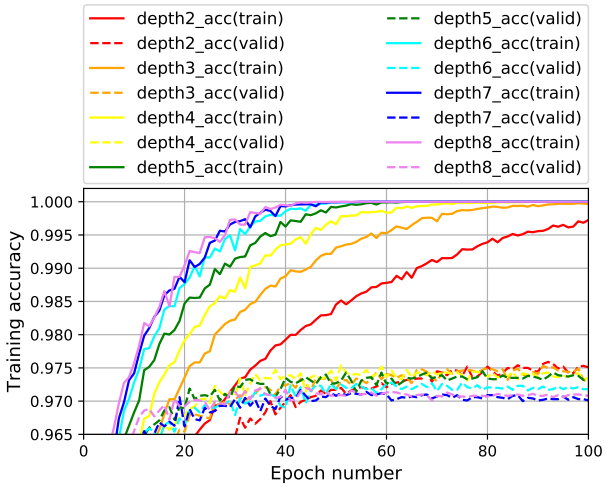

Figure 4. Training accuracy vs Epoch number while using different number of hidden layers

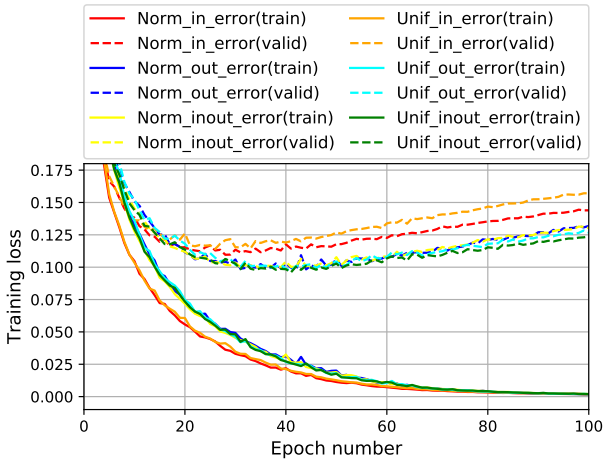

Figure 5. Training error vs Epoch number while using different initializers

- Gaussian Fan-in: $w_i \sim \mathrm{N}\left(0,\ \sqrt{1/n_{in}}\right)$,
- Gaussian Fan-out: $w_i \sim \mathrm{N}\left(0,\ \sqrt{1/n_{out}}\right)$,
- Gaussian Fan-in and Fan-out: $w_i \sim \mathrm{N}\left(0,\ \sqrt{\frac{2}{n_{in}+n_{out}}}\right)$.

For uniform distribution $\mathrm{Uniform}\,(-\mathrm{a}, \mathrm{a})$, it has a mean value 0 and standard deviation $^{a}/_{\sqrt{3}}$, and then the mean value and the standard deviation can be written down as:

- Fan-in: $mean = 0, std = \sqrt{1/n_{in}}$,
- Fan-out: $mean = 0, std = \sqrt{1/n_{out}}$,
- Fan-in and Fan-out: $mean = 0, std = \sqrt{\frac{2}{n_{in}+n_{out}}}$,

while they have the same parameter with Gaussian distribution. Noticeably, the Gaussian Fan-in is the suggested initializer in SELU model.(Klambauer et al., 2017)
Using different initializers, we used the tetra-hidden-layer model with SELU activation function and set the learning rate to 0.01. From the result shown in figure 5 and 6, two kinds of Fan-in initializers had similar converging speed, and the other four were also similar. Gaussian Fan-out, Gaussian Fan-in & out, Fan-out, and Fan-in & out had very close curves in figure 5, and all of them had smaller validation error than Gaussian Fan-in and Fan-in. Different from other Gaussian initializers, Gaussian Fan-in initializer did reduce the validation error, compared to Fan-in. While comparing the accuracy performance, we find Fan-out and Fan-in & out had the highest accuracy at around 0.974.

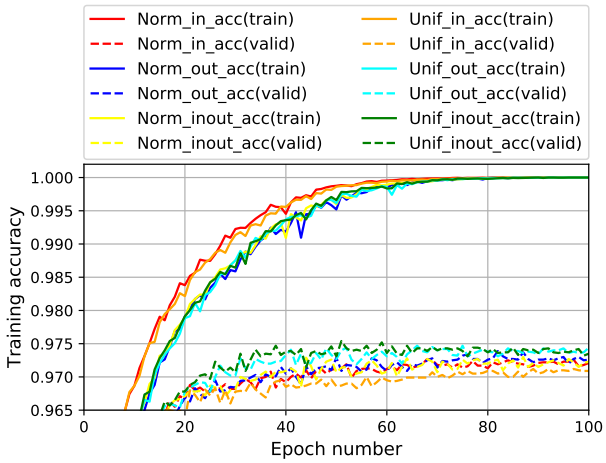

Figure 6. Training accuracy vs Epoch number while using different initializers

5. Conclusions

In this experiment, we compared different activation functions: sigmoid, ReLU, LReLU, ELU, and SELU. Compared to ReLU, LReLU can lower the validation error and improve the accuracy without the price of slowing down converging speed. ELU and SELU can achieve better performance with the price of slowing down converging speed. Compared to ELU, SELU paid less and gained more.
We also used different numbers of layers. With the increase of layer quantities, the model was more overfitted and converged in less epochs, since more parameter will cause overfitting. From the results above, the validation error has the same tendency with the accuracy.
We tested the different initializers. Although $N(0,\ \sqrt{1/n_{in}})$ is suggested for SELU model, it was not the best initialization strategy. It did not slow down converging speed and reduced the validation error. In the accuracy test, Fan-out and Fan-in&out were at the highest level, three kinds of Gaussian initializers were at the moderate level, and Fan-in was the worst.

References

URL http://cs231n.github.io/neural-networks-1/.

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

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.