MLP Coursework 1: Activation Functions

s1773005

Abstract

The abstract should be 100–200 words long, providing a concise summary of the contents of your report.

1. Introduction

Deep learning neural network is a very popular and practical model to simulate complex non-linear relationships. We construct a multi-layer networks to classified the data from MNIST digit. The MNIST data contains thousands of scanned-images of handwritten digits from 0 to 9. The images are grey-scale and 28 * 28 pixels in size. There are two parts of data extracted from the MNIST data: the training set and validation set. The training set is aimed to train the model while validation set is aimed to measure the error and accuracy of the model. We use 50000 data points training set and 10000 data points training set and use a total of 100 epochs with a batch size of 50 to train the model in order to have computationally efficient and less noisy.

The core of the networks is a four-layer structure, consisting of 784 input units, two hidden layers with 100 units, and 10 output units. The input units represent the images from MNIST as training inputs x_i, while i is from 1 to 784. The output of the input units is the sum of $w_{ij}x_i + b_{ij}$, while w_{ij} means the weight between the i_{th} input units and the j_{th} hidden units from the first hidden layer. Then, we give a activation function, such as Sigmoid function to the sum value and put it as the input of the first hidden layer. We will talk more about the activation function in the following section. We then do the same thing to forward propagate the activations value from input layer to the output layer.

As to the training part, we initial the value of weights with the Glorot initialisation scheme in part1 and part2A of the experiment and the uniform distribution initialisation scheme in part2B which will be discussed in the third part. The initial value of the bias is consistent equal to one. Then we use back propagation algorithm to back propagate the gradient through a layer so that we can use stochastic gradient descent learning rule with a learning rate to update the weights and biases of the model. We also need to calculate the error value to get the gradient. Here we considered a multi-class cross entropy error with soft max applied to outputs as the error value.

There are many free-parameters in this model, such as initial-scale from the random number generator, learning rate from the learning rule and the number of hidden layers. However, we still have no knowledge about weather this parameters influence the error and accuracy of our model. Therefore, we designed an experiment to fix these kinds of parameters and try to figure out how these parameters influence the result of the neural networks.

In the following, we will talk about activation functions of the network, using different activation functions and comparing the error and accuracy of them. Then we choose different numbers of hidden layers and values of learning rate as long as different initialisation scheme of the weights to find the best model. At last we make a conclusion about the results and discuss the effect of the parameters.

2. Activation functions

We implement different activation functions to constrain the value: Sigmoid units and Restricted Linear Unit(ReLU) as the baseline systems, Leaky ReLU, Exponential Linear Unit(ELU) and Scaled Exponential Linear Unit(SELU). **ReLU** has the following form:

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

which has the gradient:

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

Leaky ReLU(lrelu(x)) has the following form:

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

which has the gradient:

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

Where α is a constant equalling to 0.01.

ELU(elu(x)):

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

which has the gradient:

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

where α =1.

SELU(selu(x)):

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

which has the gradient:

$$\frac{d}{dx}\text{selu}(x) = \lambda \begin{cases} \alpha exp(x) & \text{if } x \leq 0 \\ 1 & \text{if } x > 0, \end{cases} \tag{8}$$

where $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$. We then compare the error and accuracy of training set and validation set.

3. Experimental comparison of activation functions

We implemented all the activation functions mentioned above to forward propagate the activations from input units to output units. Then we compared the error and accuracy of all the three models themselves and also compared them with baseline separately. The results are showing in `figure1` and `table1`.

The accuracy and error from training set of all activation function are all very low, so we did not put them in the `table1`. From the `table1` we can find that LeakyReLU has the lowest error from validation data set. However, from the `figure1` we can find that the model is over-fitting as the error from validation data set increased when the epoch number is larger than 20. Therefore, we may stop the training early in order to get a better validation set error.

Furthermore, we compared the three activation functions with the two baselines. As to Sigmoid Unit, the error from validation data set of three functions are all larger than Sigmoid Unit. However, from the `figure1`, we can find that if we stop the training early, then we can get a great error smaller than Sigmoid Unit. As to ReLU Unit, they are all very similar. They have the same tuning curve and nearly error and accuracy from the validation data and the validation set error of LeakyReLU is smaller while others are larger. Also, compare to the Sigmoid Unit, the ReLU Unit necessarily stop the training early in order to get better validation set error. Therefore, ReLU Units is a better baseline than Sigmoid Units. At last, LeakyReLU has the fast speed than others which will beneficial to our following experiment.

Activation	Error(Valid)	ACC(Valid)	run time/epoch
Sigmoid	8.83e-02	9.77e-01	2.17
ReLU	1.17e-01	9.80e-01	1.55
LeakyReLU	1.12e-01	9.80e-01	2.03
ELU	1.23e-01	9.79e-01	2.55
SELU	1.25e-01	9.79e-01	2.98

Table 1. Error and accuracy of validation data set and the run time per epoch.

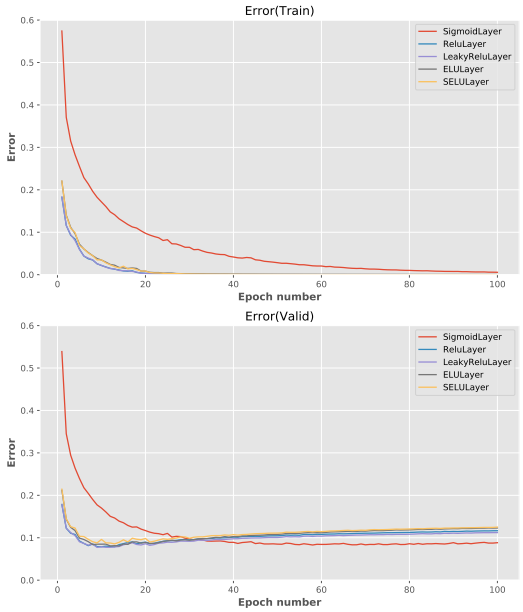

Figure 1. The error of the 2 hidden layers networks with five different activation function: Sigmoid Units, ReLU, Leaky ReLU, ELU and SELU.

Due to the information mention above, it is reasonable to choose the LeakyReLU as the best activation functions.

4. Deep neural network experiments

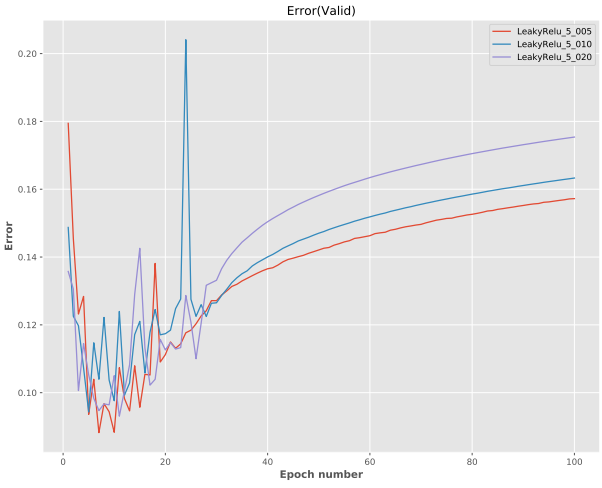

Figure 2. The error of the 5 hidden layers networks with three different learning rate

We would like to fix the learning rate and the number of hidden layers during this chapter. We choose the number of hidden layers from 2 to 8 and the learning rate from the

0.05, 0.1, 0.2 and 0.5. The best way to fix the learning rate is to vary it during every possible number of hidden layers, but it will cost so much time and seems not very practical. However, we can choose the 5 hidden layers model to fix learning rate. When the number of hidden layers is very large, the model will very easily over-fitting because they simulated too much features. At this time, the small learning rate will be more likely to give a good result. When the number of hidden layers is very small, the model will be more easily stuck in a local minimum. Then, a large learning rate can help much about this model. In order to avoiding this biases, we choose the middle number of layers to properly fit every situations.

As to the `figure2`, although the error of the validation set is over-fitting, all three learning rate has the same accuracy. Therefore, we just chose the learning rate 0.05 giving the smallest validation set error and being the fastest one among three learning rate. Then we compared models having different number of layers from 2 to 8 with learning rate 0.05.

LearningRate	Error(Valid)	ACC(Valid)	run time/epoch
0.05	1.57e-01	9.80e-01	2.73
0.10	1.63e-01	9.80e-01	2.87
0.20	1.75e-01	9.80e-01	3.09

Table 2. Error and accuracy of validation data set and the run time per epoch.

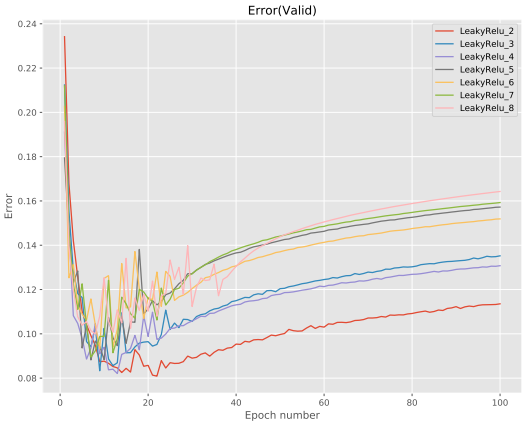

Figure 3. The error of different number of hidden layers networks with three different learning rate

From the `figure3`, we can find that all number of hidden layers are over-fitting with 100 epochs. Therefore, we consider the accuracy`table3`. It shows that 7 layers have the highest accuracy overall. Furthermore, we can find that with more number of layers, the speed will become slower. This is because more number of layers means the model represent more features from the output and need to calculate gradient and output of these all feaers. Also, we can find from `figure3` that with more number of layers, the early stage of training will contain much more unstable, as in general, the more number of hidden layers, the more flexible of model, which needs more number of data to train.

NumOfHidden	Error(Valid)	ACC(Valid)	run time/epoch
2	1.14e-01	9.78e-01	1.91
3	1.35e-01	9.79e-01	2.18
4	1.31e-01	9.80e-01	2.59
5	1.57e-01	9.80e-01	2.64
6	1.52e-01	9.79e-01	3.08
7	1.59e-01	9.81e-01	3.46
8	1.64e-01	9.79e-01	5.00

Table 3. Error and accuracy of validation data set and the run time per epoch.

Then, we focus on the scale of initial, which also influences the error and accuracy of the model. First, we use uniform distribution to generate random number in order to initialize the weights of the model instead of GlorotUniform. We use $-\sqrt{\frac{3}{n_{in}}}$ and $\sqrt{\frac{3}{n_{in}}}$ or $-\sqrt{\frac{3}{n_{out}}}$ and $\sqrt{\frac{3}{n_{out}}}$ to decide the lower and upper bound of interval, while n_{in} means the dimensions of the input units and n_{out} means the dimensions of the output units. We only train the data with 5 and 6 hidden layers because large number of layers are sensitive to the change of initial scale and they have a great speed.

NumOfHidden	Error(Valid)	ACC(Valid)	run time/epoch
5	1.52e-01	9.82e-01	8.59
6	1.69e-01	9.80e-01	10.50

Table 4. Error and accuracy of validation data set and the run time per epoch of the FanIn method

NumOfHidden	Error(Valid)	ACC(Valid)	run time/epoch
5	1.66e-01	9.80e-01	9.03
6	1.57e-01	9.81e-01	8.93

Table 5. Error and accuracy of validation data set and the run time per epoch of the FanOut method

We can see from `table4` and `table5` that with input or output dimensions initialized weights, the validation set error is lower than before, which means correlated the input and output dimensions to the initialization of weights have beneficial to the error and accuracy of the model. However, it has a bad impact on the speed of the running, as it need to reinitialization the weights during every step through layers.

5. Conclusions

We create a multi-layer neural networks to classify the scanned hand-written digit from MNIST with low error and high accuracy. We first compare different activation function and find the LeakyReLU is the best activation function for our model. Then we fixed the learning rate by using 5 hidden layers neural networks and choose 0.05 as the best learning rate. Furthermore, we compare different number of hidden layers and we find 7 hidden layers is the best number to classify the data. At last, we change the scale of initial that make it correlated to the input and output dimensions, and also use Uniform random number generator instead of GlorotUniform to initialize the weights of our model.

We can have an eye on every validation set error curve, which shows that the model is over-fitting every time. As 7 hidden layers have the best accuracy and correlate the input and output to the initialization of weights are beneficial to the error of the model, we can fixed these two parameters and change the learning rate or use the some skill such as early stopping to avoid over-fitting. Also, we can give every number of hidden levels of our model a individual optimized learning rate and compare their performances afterwards.

At last, when we used the Glorot and BengioâĂŹs combined initialisation and Gaussian distribution instead of uniform distribution, it always show the overflow problem so that we can not perfectly compare the difference among different type of correlations to the input and output dimensions. It probably some restrict of these kind of methods to train these neural networks as they will output a large number of big numbers.