Report TP1 & TP2

OPT6 : Deep Learning

YANG Mo

13 décembre 2017

1 TP1 : Single Layer Neural Network with Softmax

The first tp use a single layer neural network with no hidden layer to cope with Minst Database. The whole program contains some fundamental functions which could be used to tp2 (multi-layer neural network). The most important functions are forward, output layer softmax, loss and gridient descend. Foward function : $z_j = \sum_{i \to j} W_{ij} x_i + b_j$, it calculate the product between input layer and weights and add bias. Softmax takes the result as inputs and output the normalized probability of wach class. Loss function is defined as sum of minus negative log of each probability. To train parameters (weights and bias), we use gradient descend to optimize loss function, which is called back propagation. Back propagation try to find optima of loss function to adjust expected output. To accelerate training process, training data is dealt with mini batch.

Results : Final accurency on valid set : 0.88

Loss curve graph and error rate graph :

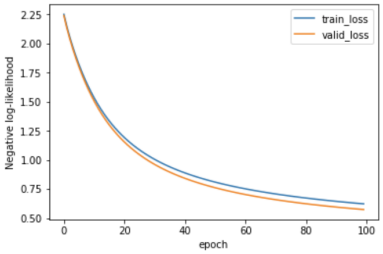

FIGURE 1 – Loss function curve

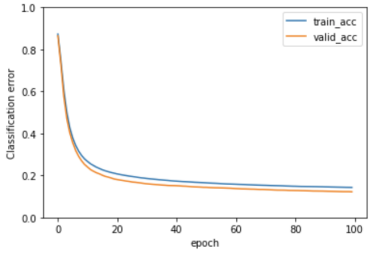

FIGURE 2 – Error rate curve

2 TP2 : Multi-layers neural network with hidden layers - Mnist database

The objective of tp2 is to implement hidden layers to neural network. In a deep neural network, forward and backward are processed layer by layer, and an avtivation function is to implement to cope with non-linear complex functions. So the fundamental task is to establish a proper structure to store output and derivative of each, and to implement activation functions properly.

Yet 3 common activation functions are implemented : sigmoid, tanh, relu. For a easy task as classification of Mnist dataset, using all 3 activation functions can get a very good result. Here are the results of using different activation functions :

Sigmoid :

hyper parameters : n_hidden = [128,64], act_func = 'sigmoid', eta = 0.1, batch_size = 50, n_epoch = 20

Results : accurency = 0.968

Loss curve graph and error rate graph :

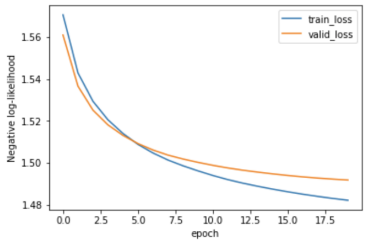

FIGURE 3 – Loss function curve of sigmoid activation function

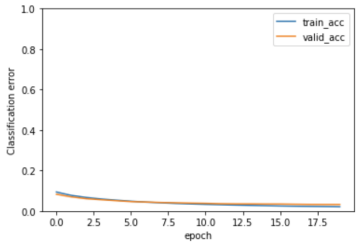

FIGURE 4 – Error rate curve of sigmoid activation function

Tanh :

hyper parameters : n_hidden = [128,64], act_func = 'tanh', eta = 0.1, batch_size = 50, n_epoch = 20

Results : accurency = 0.974

Loss curve graph and error rate graph :

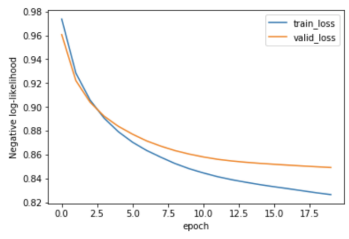

FIGURE 5 – Loss function curve of tanh activation function

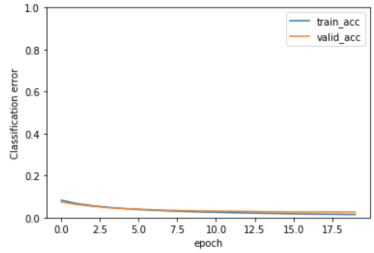

FIGURE 6 – Error rate curve of tanh activation function

Relu :

hyper parameters : n_hidden = [128,64], act_func = 'relu', eta = 0.1, batch_size = 50, n_epoch = 20

Results : accurency = 0.978

Loss curve graph and error rate graph :

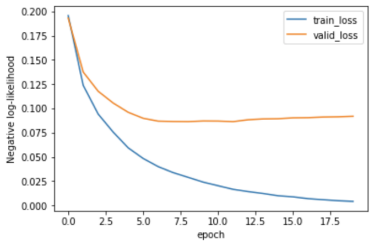

FIGURE 7 – Loss function curve of relu activation function

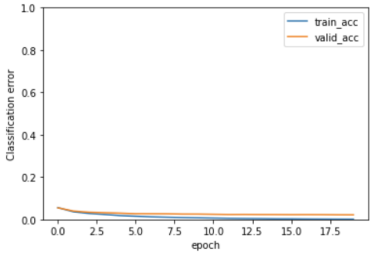

FIGURE 8 – Error rate curve of relu activation function

Resume :

We can see that with two hidden layer the result is huge boosted compared to single layer neural network(which is simply a multi labels logistic regression) We can also see that relu has learned faster than sigmoid and tanh. The result of relu is also the best. But we have not seen yet more property of each activation function since Mnist is a very simple task. So in the next part I have tried to challenge a more complex dataset : Cifar10

3 TP2 : Multi-layers neural network with hidden layers - Cifar10 database

To continue evaluate the multi-layer neural network in more complex tasks, I have chosen using the program to train Cifar10 dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. It is much more complex. Today the best conv neural network can get a result over 0.9 accurency, however in this part we start from simple feedback neural network with two hidden layers. The objective of this part is not only to evaluate the correctness of our neural network, but also learn more about the effects of each hyper pa-

rameters, some tricks to improve performance, the differences between different activation functions..... First we compare the result of different activition functions without setting anything special about hyper parameters :

hyper parameters : n_hidden = [128,64], eta = 0.1, batch_size = 50, n_epoch = 20

sigmoid : Result : accurency = 0.464

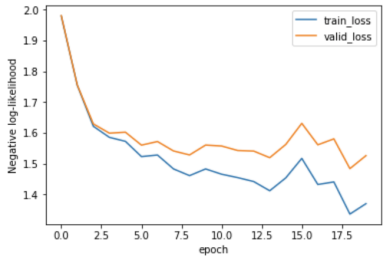

FIGURE 9 – Loss function curve : sigmoid

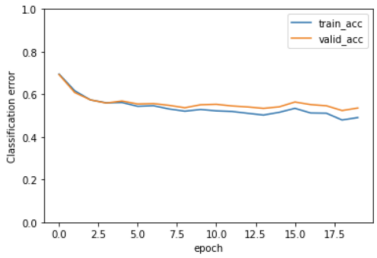

FIGURE 10 – Error rate curve : sigmoid

sigmoid : Result : accurency = 0.456

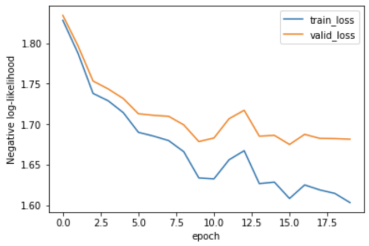

FIGURE 11 – Loss function curve : tanh

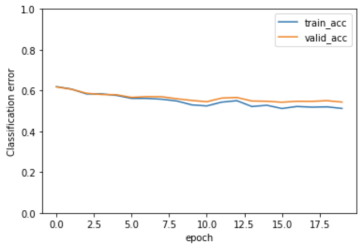

FIGURE 12 – Error rate curve : tanh

relu : Result : accurency = 0.485

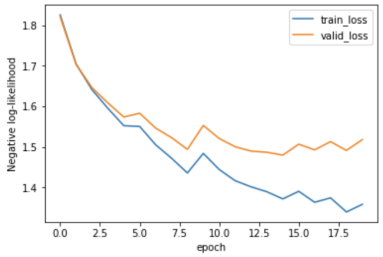

FIGURE 13 – Loss function curve : relu

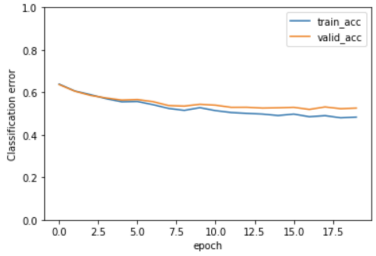

FIGURE 14 – Error rate curve : relu

3.1 learning rate decay

We can resume from the results above that the descend of loss function is rather stable at first and gradually become unstable. That is because when optimizer is close to optimal, we need to shrink step length to approximate optimal. Here a simple algorithm to shrink learning rate step by step is implemented and is tested on relu. The learning rate will decay over each epoche. $eta_{t+1} = eta_t * e^{-k}$. There are a lot of ways to control step length and this is only a simple one. Here is the comparison before and after implementing learning rate decay :

Results :

Fixed learning rate : accurency = 0.485

Learning rate decay : accurency = 0.514

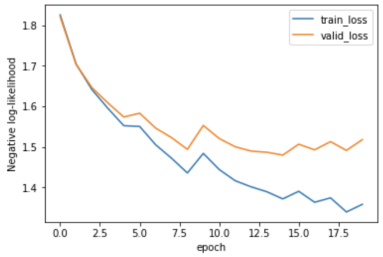

FIGURE 15 – Loss function curve : relu, fixed learning rate 0.1

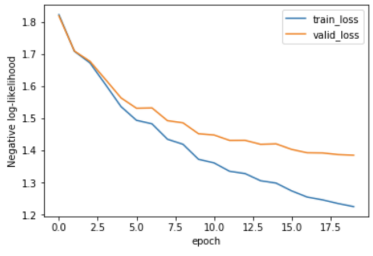

FIGURE 16 – Error rate curve : relu, learning rate decay

We see that learning rate decay is useful to approximate optimal and make

gradient descend more stable.

3.2 regularization

Regularization can cope with over-fitting issues. There is no obvious over-fitting problem in this case since the training set is very rich and sufficient. After testing on L1 regularization and L2 regularization, I find out that both works very well, but both have minor influence to classifier's performance. There is this trade-off, regularization can help to avoid over-fitting, but it might hurt the performance of the classifier. So if the learning is troubled by over-fitting seriously we should use regularization, other wise should not use it or lower the hyper-parameter λ.

3.3 Resume

So far, the most tedious work to improve classifier's performance is to adjust the hyper parameter, including learning rate, number of hidden layers, hidden units, choice of activation functions,regularization parameters, iterations, mini-batch size..... Here are some experiences when optimizing the classifier :

1 : relu normally is faster than sigmoid and tanh, the reason is that sigmoid and tanh's gradient descend very slow when their values is too high or too low. relu just ignore whatever its value

2 : relu is more unstable and more sensitive to learning rate. The most important thing is to try out the suitable learning rate, better to introduce a proper learning rate decay mechanism. Setting a upper bound to the value of relu can also make relu more stable.

3 : When having a over-fitting probelm, we can try out regularization. The parameter λ should be tuned to get balance between overcomming over-fitting and do not hurt performance.

4 : Deep neural network works better but also have a higher computential cost. It is helpful to start from shallow and simple neural networks, tune hyper parameters, then use those experience to go deeper or get a bigger network.