EE599: Homework #3

Yifan Wang
wang608@usc.edu #3038184983

February 5, 2019

Problem 1

For Softmax, its output is:

$$a = \begin{bmatrix} \frac{e^{s_1}}{\sum_{i=1}^{n} e^{s_i}} \\ \cdots \\ \cdots \\ \frac{e^{s_n}}{\sum_{i=1}^{n} e^{s_i}} \end{bmatrix} \tag{1}$$

cross-entropy cost is:

$$C = -\sum_{i=1}^{n} y_i \ln a_i \tag{2}$$

if y is one-hot label, for one particular data belongs to k^{th} class, this can be reduced to:

$$C = -\ln a_k \tag{3}$$

Then, derivatives of C is:

$$\nabla C = \begin{bmatrix} 0 \\ \cdots \\ \frac{\partial C}{\partial a_k} \\ \cdots \\ 0 \end{bmatrix} \tag{4}$$

derivatives of softmax is :

$$H' = \begin{bmatrix} \frac{\partial a_1}{\partial s_1} & \cdots & \frac{\partial a_n}{\partial s_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial a_n}{\partial s_1} & \cdots & \frac{\partial a_n}{\partial s_n} \end{bmatrix} \tag{5}$$

error vector $\delta = H'^T \nabla C$ becomes:

$$\delta = \begin{bmatrix} \frac{\partial a_k}{\partial s_1}\frac{\partial C}{\partial a_k} \\ \cdots \\ \cdots \\ \frac{\partial a_k}{\partial s_n}\frac{\partial C}{\partial a_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial s_1} \\ \cdots \\ \cdots \\ \frac{\partial C}{\partial s_n} \end{bmatrix} \tag{6}$$

where

$$C = -\ln \frac{e^{s_j}}{\sum_{i=1}^{n} e^{s_i}} = -[s_j - \ln(e^{s_j} + \sum_{i=1, i\neq j}^{n} e^{s_i})] \tag{7}$$

so that,

$$\frac{\partial C}{\partial s_j} = \frac{e^{s_j}}{\sum_{i=1}^{n} e^{s_i}} = a_j, (j \neq k) \tag{8}$$

$$\frac{\partial C}{\partial s_j} = \frac{e^{s_j}}{\sum_{i=1}^{n} e^{s_i}} = a_j - y_j, (j = k) \tag{9}$$

combine $(8)\&(9)$:

$$\delta = a - y \tag{10}$$

Proof done.

Problem 2

In this problem, I use the same parameter as the net provided in homework2 problem 1:

Layer	Neuron	input	output	activation
Input	-	-	784	-
Hidden1	200	784	200	Relu/tanh
Hidden2	100	200	100	Relu/tanh
Output	10	100	10	Softmax

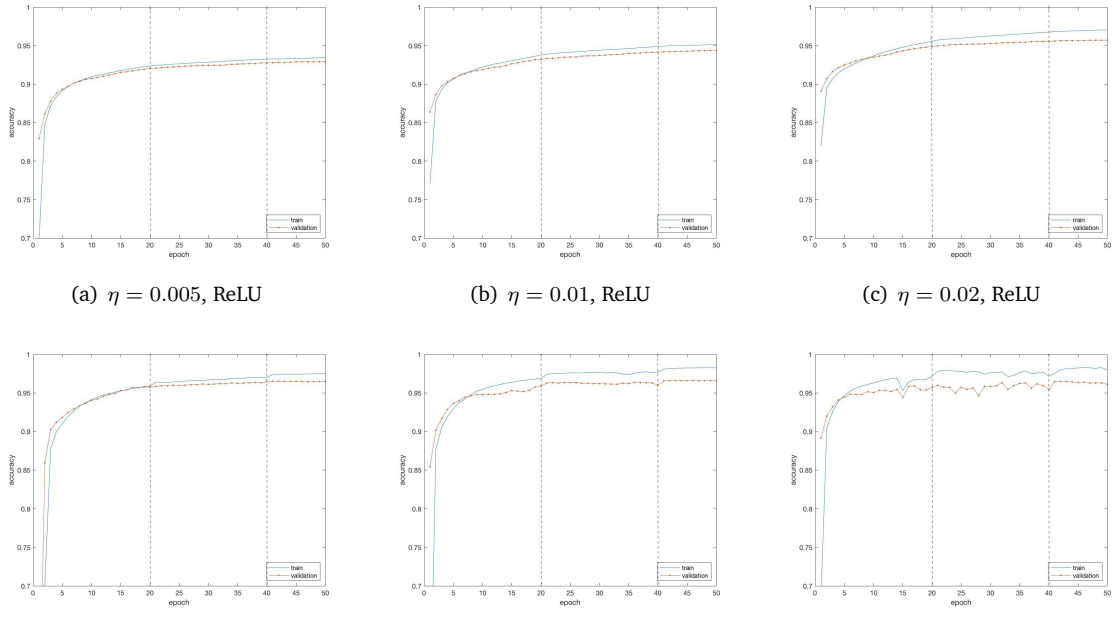
Table 1: MLP structure

learning rate	epoch	half decay	minbatch	initialization
0.005/0.01/0.02	50	20/40	100	$Equation(11)$

Table 2: MLP parameters

My network structure and parameters are shown in $Table(1,2)$. Initialization method coms from [1]:

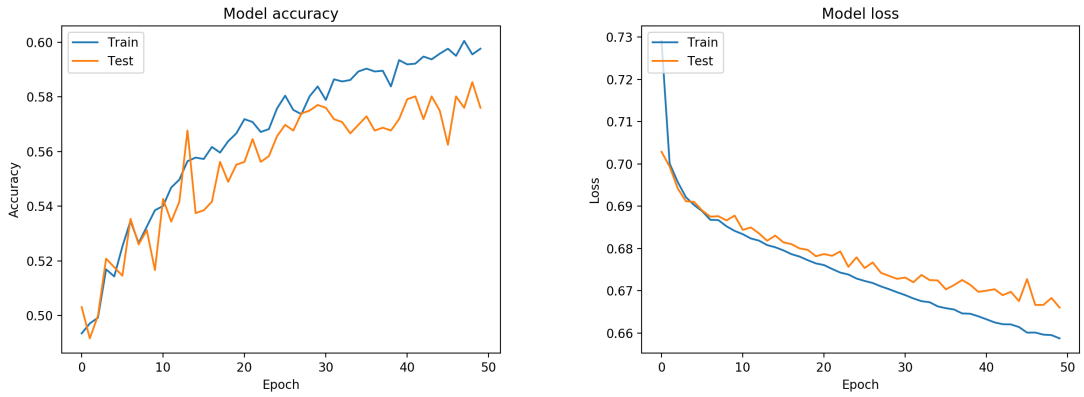
$$W = \frac{\sqrt{2}Gaussian : N(0,1)}{\sqrt{\#Node_{output} + \#Node_{input}}} \tag{11}$$



(a) $\eta = 0.005$, ReLU (b) $\eta = 0.01$, ReLU (c) $\eta = 0.02$, ReLU

(d) $\eta = 0.005$, tanh (e) $\eta = 0.01$, tanh (f) $\eta = 0.02$, tanh

Figure 1: Train Accuracy and Validation Accuracy in different learning rate



(a) Learning Cure (b) Loss

Figure 2: SGD(lr=0.005, decay=0, momentum=0.9, nesterov=True)

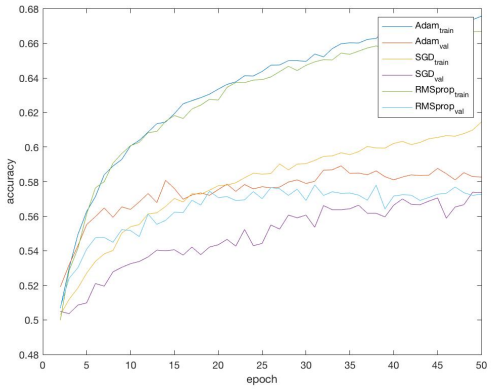


Figure 3: Different optimizers using default setting

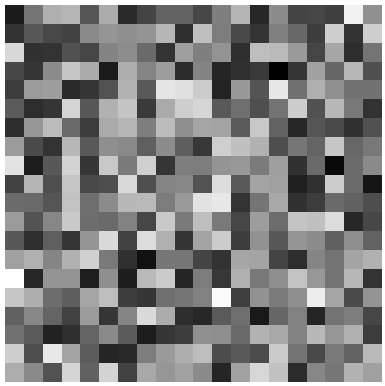


Figure 4: $20 * 20$ weight visualisation

Random initialization is needed, otherwise the whole network would be symmetric and can be regard as one single neuron.

Learning rate decay happens after epochs 20 and 40 in testing of $Figure1$. But for small task like this, decay can not have much impact on final result.

Considering activation function, it is necessary, otherwise a multi-layer network without an activation function is the same as a one layer network, since all the operations are linear, and can be combined. $tanh$ is zero mean, which is good. But it would has saturate problem as $sigmoid$. $ReLU$ do not have saturate problem, however it is not zero mean.

From the test, it can be found when using $ReLU$, a small carefully chose learning rate is needed, otherwise, it would be unstable and might overflow or lots of neurons would die, due to its character that it would never saturate when input is positive and would pass gradient with out any decay during backward.

While $tanh$ needs a relative larger learning rate to get similar result, and it would learn much slower than $ReLU$, since it has saturate problems. But one good thing about $tanh$ is that it is zero mean, which make whole system more robust when cope with large gradient update.

Best accuracy my network achieved on test data is:

$$testloss : 0.08092$$

$$testaccuracy : 0.9788$$

which use $ReLu$ as activation and learning rate is 0.1.

Problem 3

For different optimizer, their training accuracy and validation accuracy is shown in $Figure4$ (each curve is average of 5 run), they all use default setting as keras provided. it can be found that compare with SGD, Adam and RMSprop would converge much faster, which means that they required less time to get the same training effect. What's more, using SGD requires carefully choose learning rate.

The weight ($Figure5$) tends to cluster together, for example, a weight tends to have similar value as it's neighbor, I think that may represent that in the sequence, one input binary number is strongly correlated to its neighbor.

References

[1] Glorot, Xavier & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track. 9. 249-256.