10707 Homework 1

David S. Hippocampus*
Department of Computer Science
Cranberry-Lemon University
Pittsburgh, PA 15213
hippo@cs.cranberry-lemon.edu

Problem 1 (6 pts)

This question will test your general understanding of overfitting as they relate to model complexity and training set size. Consider a continuous domain and a smooth joint distribution over inputs and outputs, so that no test or training case is ever duplicated exactly.

- For a fixed training set size, sketch a graph of the typical behavior of training error rate (y-axis) versus model complexity (x-axis). Add to this graph a curve showing the typical behavior of the corresponding test error rate versus model complexity, on the same axes. (Assume that we have an infinite test set drawn independently from the same joint distribution as the training set). Mark a vertical line showing where you think the most complex model your data supports is; chose your horizontal range so that this line is neither on the extreme left nor on the extreme right. Indicate on your vertical axes where zero error is and draw your graphs with increasing error upwards and increasing complexity rightwards.

- For a fixed model complexity, sketch a graph of the typical behavior of training error rate (y-axis) versus training set size (x-axis). Add to this graph a curve showing the typical behavior of test error rate versus training set size, on the same axes (again on an iid infinite test set). Indicate on your vertical axes where zero error is and draw your graphs with increasing error upwards and increasing training set size rightwards.

- One of the commonly used regularization methods in neural networks is *early stopping*. Argue qualitatively why (or why not) early stopping is a reasonable regularization metric.

Your answer here

- As the model becomes more complex, the training error will decrease. And the testing error will decrease at first, then increase later. The most complex model will be at the point where the testing error starts to increase. And my sketch is in figure 1.
- From figure 2, when the training set size becomes larger and larger, the training error will increase. And the testing error will decrease at first, and then increase.
- The early stop uses the validation data set. As training data, people will observe the loss of the validation data. If the loss starts to increase, people will stop training the network. I think that this method is a reasonable regularization method. Because it prevent the models becoming too complex. Sometimes the network may get better if people insist to training even though they observe the validation loss increases. However, at the point that people doing early stop, the model has been trained relatively well, which is acceptable.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

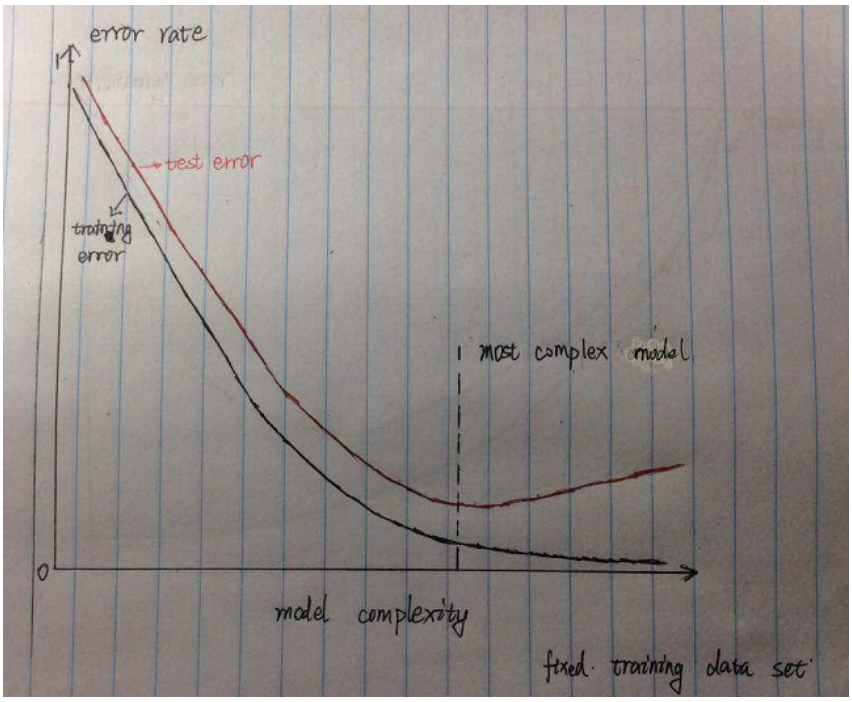


Figure 1: Error rate against complexity of the model with the fixed training data set

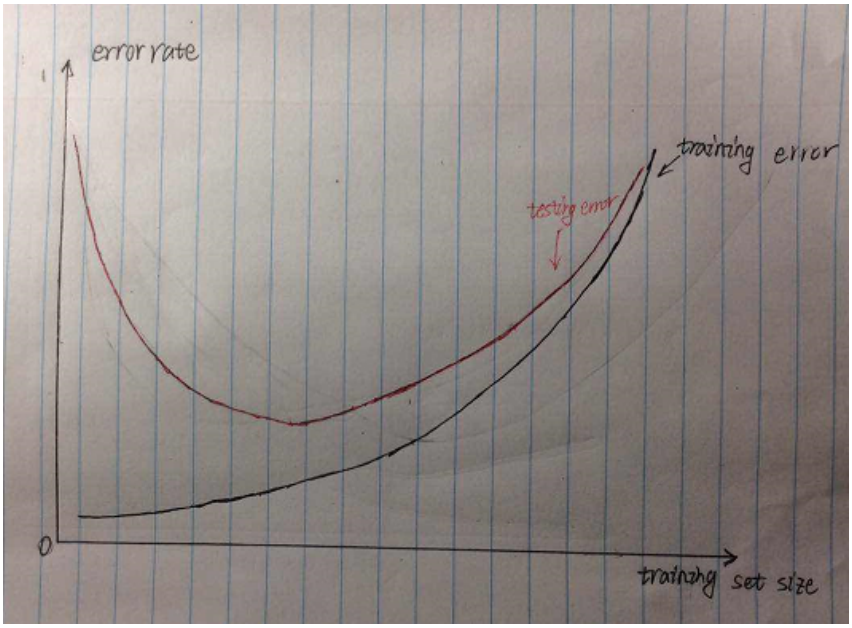


Figure 2: Error rate against the size of the training data set with fixed complexity of the model

Problem 2 (8 pts)

Consider N training points (x_i, y_i) drawn i.i.d from a distribution that is characterized as:

$$x_i \sim h(x) \tag{1}$$
$$y_i = f(x_i) + \epsilon_i \tag{2}$$
$$\epsilon_i \sim (0, \sigma^2) \tag{3}$$

where f is the regression function. An estimator for this data, *linear* in y_i is given by

$$\hat{f}(x^*) = \sum_{i=1}^{N} l_i(x^*; \mathcal{X}) y_i, \tag{4}$$

where $l_i(x^*; \mathcal{X})$ depends on the entire training sequence of x_i (denoted by $\mathcal{X}$) but do not depend on y_i. Show that k-nearest-neighbor regression and linear regression are members of this class of estimators. What would be the $l_i(x^*; \mathcal{X})$ in both these regressions (knn and linear)?

Your answer here

For the k-nn regression, after choosing the k value, people just calculate the distance between the new data and all the data in the training set. And count the number points belong to each classification, then choose the classification with the most points. And the prediction formula can be presented like:

$$f(x^*) = \frac{1}{K} \sum_{i}^{N} y_i$$

where y_i is the ith case of the examples sample. And $l_i(x^*; \mathcal{X}) = \frac{1}{K} \sum_i^N 1$

For the linear regression, the prediction has the form:

$$y_i = \sum_{i=1}^{n} \beta_i x_i + b,$$

where $\boldsymbol{\beta}$ is a set of parameters trained by the data set. And we can know that $\boldsymbol{\beta}$ can be calculated with the formula:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y},$$

which is the estimator. Then we can do the prediction with the estimator like

$$\hat{f}(x^*) = \boldsymbol{\beta} \boldsymbol{X^*}$$

Then transfer the matrix form to the sum form, we can get

$$f(x^*) = \sum_{i=1}^{N} (x^* (x_i x_i^T)) x_i y_i$$

where $l_i(x^*; \mathcal{X}) = (x_i x_i^T) x_i$

Problem 3 (6 pts)

The form of Bernoulli distribution, given by:

$$\text{Bern}(x|\mu) = \mu^x (1-\mu)^{1-x},$$

is not symmetric between the two values of $x \in \{0, 1\}$. Often, it will be convenient to use an equivalent formulation for which $x \in \{-1, 1\}$, in which case the distribution can be written as:

$$p(x|\mu) = \left(\frac{1-\mu}{2}\right)^{(1-x)/2} \left(\frac{1+\mu}{2}\right)^{(1+x)/2},$$

where $\mu \in [-1, 1]$. Show that this new distribution is normalized and compute its mean, variance, and entropy.

Your answer here

Mean calculation:

$$E(x) = 1 * p(x = 1) + (-1) * p(x = -1)$$

plug x into the formula:

$$E(x) = \left(\frac{1-\mu}{2}\right)^{(1-1)/2}\left(\frac{1+\mu}{2}\right)^{(1+1)/2} - \left(\frac{1-\mu}{2}\right)^{(1+1)/2}\left(\frac{1+\mu}{2}\right)^{(1-1)/2}$$

Then we can get:

$$E(x) = \mu$$

Variance calculation:

$$\sigma(x) = E(x^2) - (E(x))^2$$

First we calculate $E(x^2)$:

$$E(x^2) = 1^2 * p(x = 1^2) + (-1)^2 * p(x = (-1)^2)$$

And we can get:

$$E(x^2) = 1 + \mu$$

The we calculate $(E(x))^2$:

$$(E(x))^2 = \mu^2$$

Then we can get variance:

$$\sigma(x) = 1 + \mu - \mu^2$$

Cross Entropy calculation:

$$\begin{aligned} L(x) &= 1 * \log(p(x = 1)) + (-1) * \log(p(x = -1)) \\ &= \log(\frac{1+\mu}{2}) - \log(\frac{1-\mu}{2}) \\ &= \log(1+\mu) - \log(1-\mu) \end{aligned}$$

Problem 4 (12 pts)

Consider a binary classification problem in which the target values are $t \in \{0, 1\}$, with a neural network output $y(x, w)$ that represents $p(t = 1|x; w)$, and suppose that there is a probability ϵ that the class label on a training data point has been incorrectly set. Assuming independent and identically distributed data, write down the error function corresponding to the negative log likelihood. What is the error function when $\epsilon = 0$? Note that this error function makes the model robust to incorrectly labelled data, in contrast to the usual error function.

Your answer here

The error function is:

$$Error(w) = t\ln(y(1-\epsilon) + (1-y)\epsilon) + (1-t)\ln((1-y)(1-\epsilon) + y\epsilon)$$

When $\epsilon = 0$

$$\begin{aligned} Error(w) &= t\ln(y(1-0) + (1-y) * 0) + (1-t)\ln(1-y)(1-0) + y * 0 \\ &= t\ln(y) + (1-t)\ln(1-y) \end{aligned}$$

Problem 5 (8 pts)

Consider a two-layer network function in which the hidden-unit nonlinear activation functions are given by logistic sigmoid functions of the form:

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{5}$$

Show that there exists an equivalent network, which computes exactly the same function, but with hidden unit activation functions given by $\tanh(a)$ where the tanh function is defined by:

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{6}$$

Your answer here

First we can know that there's relationship between the sigmoid function and tanh function, that is:

$$2\sigma(2a) - 1 = \frac{2}{1 + e^{-2a}} - 1 = \frac{2e^{2a}}{e^{2a} + 1} - \frac{e^{2a} + 1}{e^{2a} + 1} = \frac{e^2 a - 1}{e^2 a + 1} = tanh(a)$$

that means if we scale the input and sigmoid result by 2 times, then minus a constant, we can get the same activation value as the tanh function. After we getting the same activation value, we can apply the same output function of the neural network

Problem 6 (60 pts)

For this question you will write your own implementation of backpropagation algorithm for training your own neural network. Please do not use any toolboxes. We recommend that you use MATLAB or Python, but you are welcome to use any other programming language if you wish.

The goal is to label images of 10 handwritten digits of "zero", "one",..., "nine". The images are 28 by 28 in size (MNIST dataset), which we will be represented as a vector x of dimension 784 by listing all the pixel values in raster scan order. The labels t are 0,1,2,...,9 corresponding to 10 classes as written in the image. There are 3000 training cases, containing 300 examples of each of 10 classes, 1000 validation (100 examples of each of 10 classes), and 3000 test cases (300 examples of each of 10 classes). they can be found in the file digitstrain.txt, digitsvalid.txt and digitstest.txt: http://www.cs.cmu.edu/∼rsalakhu/10707/assignments.html

Format of the data: digitstrain.txt contains 3000 lines. Each line contains 785 numbers (comma delimited): the first 784 real-valued numbers correspond to the 784 pixel values, and the last number denotes the class label: 0 corresponds to digit 0, 1 corresponds to digit 1, etc. digitsvalid.txt and digitstest.txt contain 1000 and 3000 lines and use the same format as above. As a warm up question, load the data and plot a few examples. Decide if the pixels were scanned out in row-major or column-major order.

Backpropagation Algorithm

Implement backpropagation algorithm with sigmoid activation function in a single-layer neural network. The output layer should be a softmax output over 10 classes corresponding to 10 classes of handwritten digits. Your backprop code should minimize the cross-entropy entropy function for multi-class classification problem.

a) Basic generalization [5 points]

Train a single layer neural network with 100 hidden units (with architecture: $784 \rightarrow 100 \rightarrow 10$). You should use initialization scheme discussed in class as well as choose a reasonable learning rate (e.g. 0.1). Train the network repeatedly (more than 5 times) using different random seeds, so that each time, you start with a slightly different initialization of the weights. Run the optimization for at least 200 epochs each time. If you observe underfitting, continue training the network for more epochs

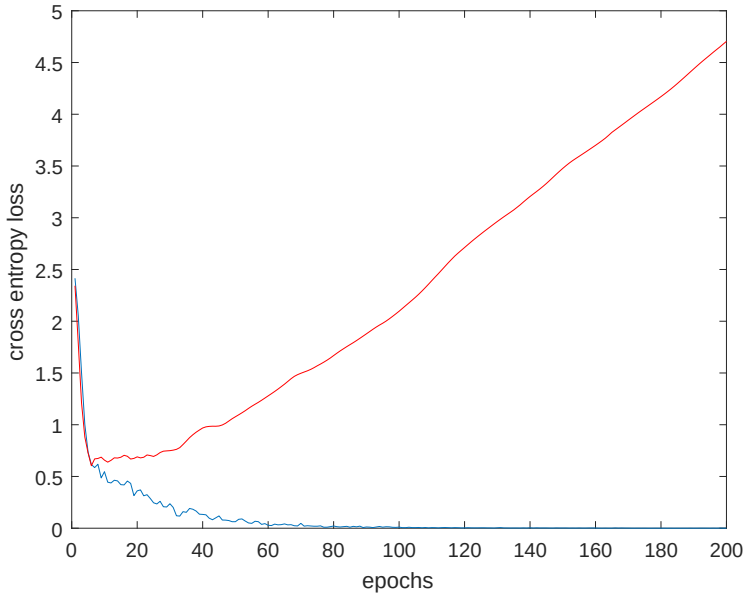


Figure 3: The cross entropy in 200 epochs

until you start seeing overfitting.

Plot the average training cross-entropy error (sum of the cross-entropy error terms over the training dataset divided by the total number of training example) on the y-axis vs. the epoch number (x-axis). On the same figure, plot the average validation cross-entropy error function.

Examine the plots of training error and validation error (generalization). How does the network's performance differ on the training set versus the validation set during learning? Use the plot of error curves (training and validation) to support your argument.

Your answer here

After running five times. I found the figures are quite similar, so I just put one in the report, and other four pictures are in the folder for your reference. And all the results used the network without any optimization, such as regularization or momentum. From the figure 3, I found that, the loss of the training data keeps going down(the blue line is the loss of the training data, and the red line is the validation data). However the loss of the validation data first going down, then going up. As we all know, when the loss of the validation data goes up, it is the start of the over fitting. And we can tell from the figure 3, after training for some epochs, the loss of the validation data starts to go up.

b) Classification error [5 points]

You should implement an alternative performance measure to the cross entropy, the mean classification error. You can consider the output correct if the correct label is given a higher probability than the incorrect label. You should then count up the total number of examples that are classified incorrectly (divided by the total number of examples) according to this criterion for training and validation respectively, and maintain this statistic at the end of each epoch. Plot the classification error (in percentage) vs. number of epochs, for both training and validation. Do you observe a different behavior compared to the behavior of the cross-entropy error function?

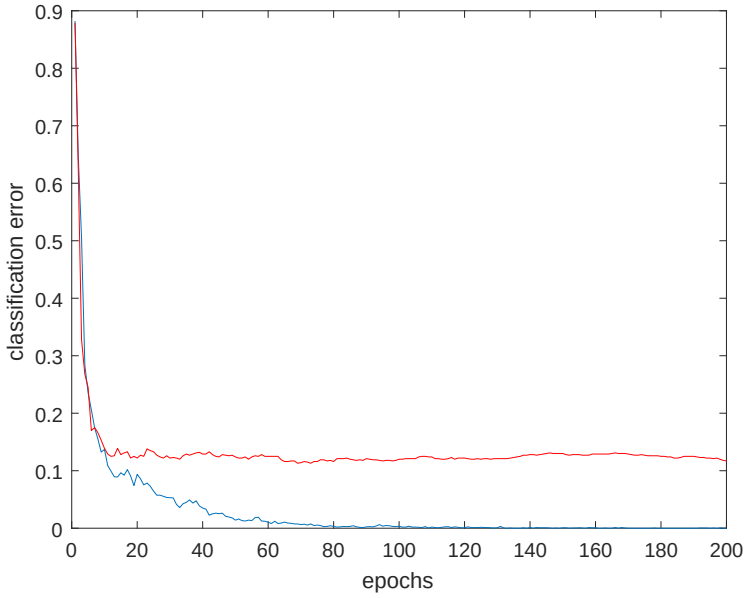


Figure 4: The classification error in 200 epochs

Your answer here

The classification error is shown in figure 4. Similar to the figure 3, the blue line is for the training data, and the red line is for the validation data set. From the picture we can see that the classification errors for both training data keep going down. And there's a different behavior compared to cross entropy picture for validation data. We can see that although the classification error keeps going down for the validation data, the cross entropy loss stills going up. To explain this, we can consider how the classification error be calculated. It tries to find the right classification by checking the possibility generated by the softmax function and finding the largest possibility, then outputing the classification corresponding to the largest probability. However the cross entropy consider the value of the probability, when the largest probability approaching 0.5. The cross entropy loss will increase, but the classification is still right. So we can see that the classification error of the validation data keeps going up, but the cross entropy after going up will go down.

c) Visualizing Parameters [5 points]

Visualize your best results of the learned W as 100 28x28 images (plot all filters as one image, as we have seen in class). Do the learned features exhibit any structure?

Your answer here

From the figure 5, we can see that the learned features exhibit some structures.

d) Learning rate [5 points]

Try different values of the learning rate ϵ. You should start with a learning rate of 0.1. You should then reduce it to $.01$, and increase it to 0.2 and 0.5. What happens to the convergence properties of the algorithm (looking at both average cross entropy and % Incorrect)? Try momentum of $\{0.0, 0.5, 0.9\}$. How does momentum affect convergence rate? How would you choose the best value of these parameters?

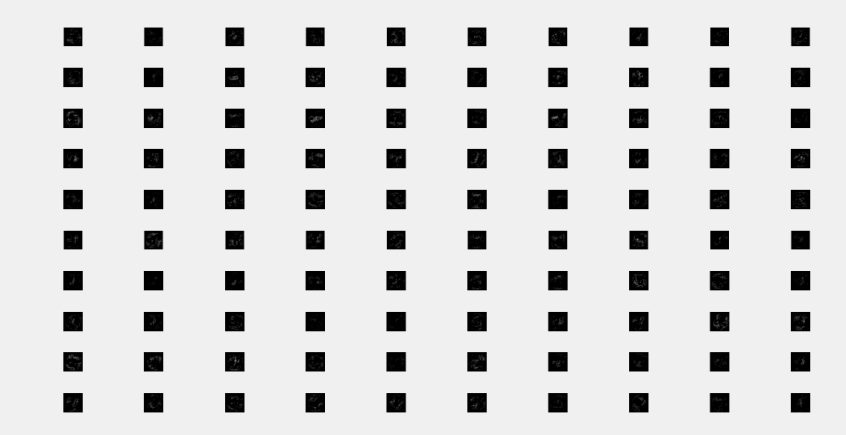
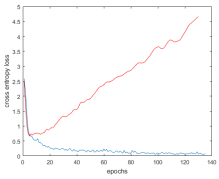
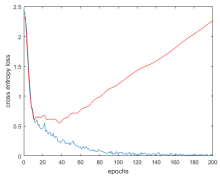


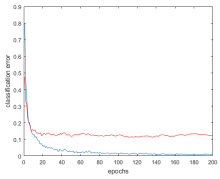
Figure 5: Visualization



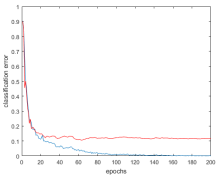
(a) The loss of the learning rate = 0.1



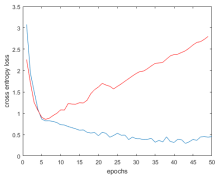
(b) The loss of the learning rate = 0.01



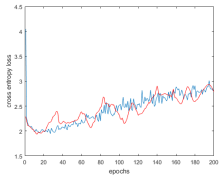
(c) The error of the learning rate = 0.1



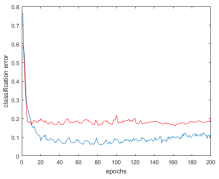
(d) The error of the learning rate = 0.01



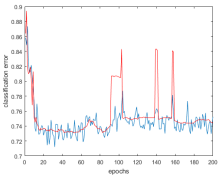
(e) The loss of the learning rate = 0.2



(f) The loss of the learning rate = 0.5



(g) The error of the learning rate = 0.2



(h) The error of the learning rate = 0.5

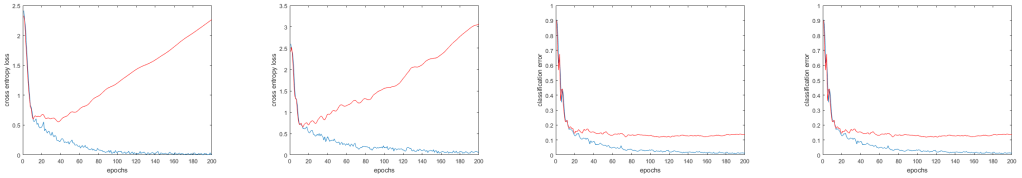
Figure 6: learning rate comparison

Your answer here

Start with the learning rate 0.1, from figure 6, we can see that when learning rate goes down, the converge speed goes down, but both the loss line and error line become more smooth. And when learning rate goes up, the training error can not be as small as before, the training process even cannot converge. And both error line and loss line bounce up and down. We can observe the result from figure 7.
And when using momentum, we can see that when momentum is proper, it can help the training converge faster. But when momentum is very big, it will lead the network can not converge, which we can tell from figure 7.

e) Number of hidden units [5 points]

Set the learning rate ϵ to .01, momentum to 0.5 and try different numbers of hidden units on this problem. You should try training a network with 20, 100, 200, and 500 hidden units. Describe the effect of this modification on the convergence properties, and the generalization of the network.

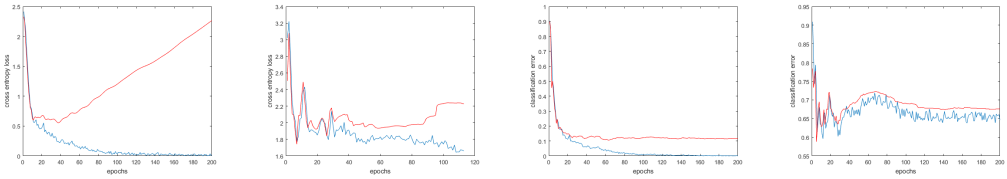


(a) The loss of the learning rate = 0.01

(b) The loss of the learning rate = 0.01 and momentum = 0.05

(c) The error of the learning rate = 0.01

(d) The error of the learning rate = 0.01 and momentum = 0.05



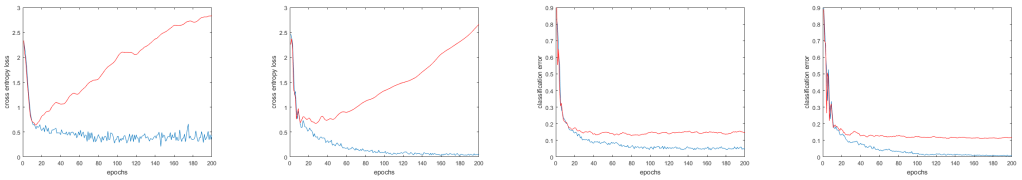
(e) The loss of the learning rate = 0.01

(f) The loss of the learning rate = 0.01 and momentum = 0.09

(g) The error of the learning rate = 0.01

(h) The error of the learning rate = 0.01 and momentum = 0.09

Figure 7: momentum comparison

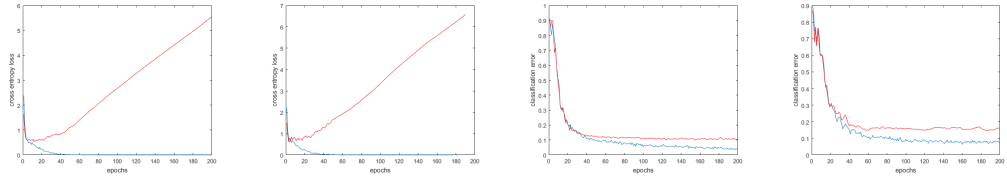


(a) The loss of a 20-neurons network

(b) The loss of a 100-neurons network

(c) The error of a 20-neurons network

(d) The error of a 100-neurons network



(e) The loss of the learning rate = 0.01

(f) The loss of the learning rate = 0.01 and momentum = 0.09

(g) The error of the learning rate = 0.01

(h) The error of the learning rate = 0.01 and momentum = 0.09

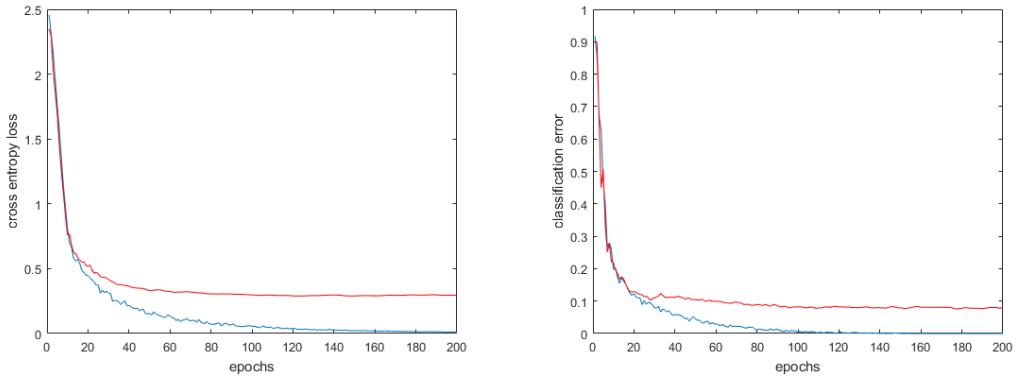
Figure 8: Different number of hidden neurons comparison

Your answer here

When there are 20 hidden units, both training error and validation error becomes bigger. And as the hidden units become more and more, when hidden units equal 100 and 200, the training error keeps going down, even to 0. However when hidden units equal 500, the training error goes up. The network becomes not very stable.
And we can see that as the number of neurons increases, the speed of converge is faster and the generalization becomes easier. The results can been seen in figure 8. And as the hidden units increase, training requires more time.

f) Best performing single-layer network [10 points]

Cross-validation: Explore various model hyper-parameters, including



(a) Best performance for loss (b) Best performance for error

Figure 9: Loss and classification error of best performance

- learning rates
- momentum
- number of hidden units in each layer
- number of epochs (early stopping)
- L_2 regularization (weight decay)

to achieve the best validation accuracy. Briefly describe your findings.

Given the best found values, report the final performance of your 1-layer neural network (both average cross entropy and % Incorrect) on the training, validation, and test sets. Visualize your best results of the learned W as 28x28 images (plot all filters as one image, as we have seen in class).

Your answer here

Figure 9 and figure 10 shows the best performance and the visualization of a one-layer network. To achieve the performance, I found that it's difficult to trade off among all the parameters. And I found that regularization is really good at preventing generalization. The parameter values are:

- learning rates: 0.01
- momentum:0.1
- number of hidden units in each layer:100
- number of epochs (early stopping):200
- L_2 regularization (weight decay):3000

The Error:

- training data: 0 (for best performance)
- validation data: 0.08(for best performance)
- testing data: 0.0857(for average performance)

The Loss:

- training data: 0.0014 (for best performance)
- validation data: 0.3(for best performance)
- testing data: 0.4261(for average performance)

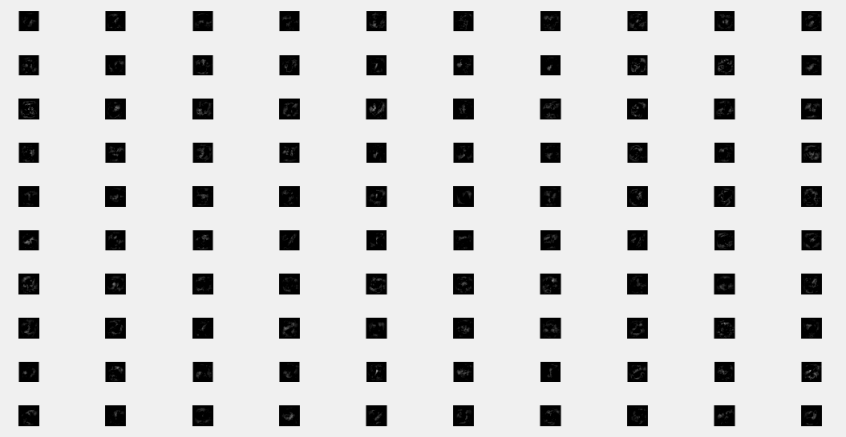


Figure 10: visualization of best performance

g) Extension to multiple layers [10 points]

Implement a 2-layer neural network, starting with a simple architecture containing 100 hidden units in each layer (with architecture: $784 \rightarrow 100 \rightarrow 100 \rightarrow 10$).

Cross-validation: Explore various model hyper-parameters, including learning rates, momentum, number of hidden units in each layer, number of epochs, and weight decay to achieve the best validation accuracy. Briefly describe your findings.

Given the best found values, report the final performance of your 2-layer neural network (both average cross entropy and % Incorrect) on the training, validation, and test sets. Visualize your best results of the learned 1st-layer W as 28x28 images (plot all filters as one image, as we have seen in class). How do these filters compare to the ones you obtained when training a single-layer network?

Does 1-layer network outperform a 2-layer model in term of generalization capabilities?

Your answer here

Figure 11 and 12 shows the best performance and visualization of a 2-layer network. From the picture and the performance of the testing data, we can see that the 2-layer network's performance is not as good as a one-layer network. And the 1-layer network outperforms a 2-layer model in terms of generalization capabilities. The parameter values are:

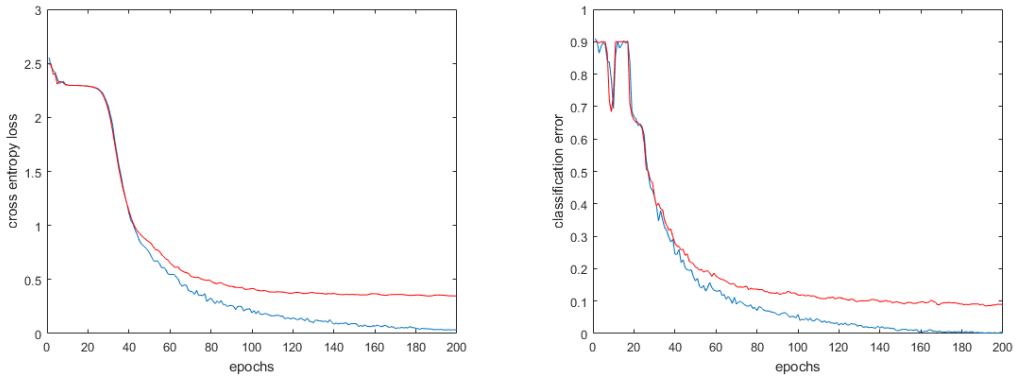
- learning rates: 0.01
- momentum:0.01
- number of hidden units in each layer:200, 100
- number of epochs (early stopping):200
- L_2 regularization (weight decay):3000

The loss:

- training data: 0.0216 (for best performance)
- validation data: 0.35(for best performance)
- testing data: 0.04295(for average performance)

The error:

- training data: 0.002 (for best performance)



(a) Best performance for loss of 2-layer network (b) Best performance for error of 2-layer network

Figure 11: Loss and classification error for best performance of 2-layer performance

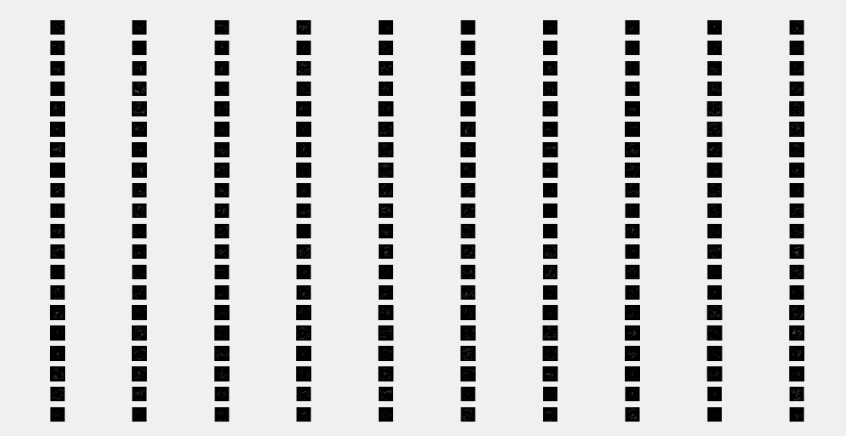


Figure 12: visualization for best performance of 2-layer performance

- validation data: 0.1(for best performance)
- testing data: 0.104(for average performance)

h) Batch Normalization [10 points]

For your two-layer network, implement batch normalization for a batch size of 32. Describe how batch norm helps (or doesn't help) in terms of speed and accuracy (train and validation).

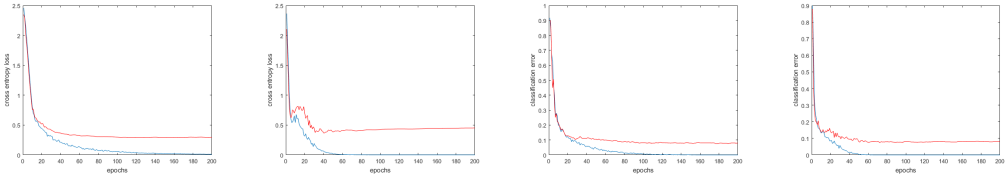
Your answer here

i) Different Activation Functions [5 points]

Now, change the activation functions to ReLU and tanh instead of the original sigmoid. Do you see any difference in terms of performance or accuracy ? Report your findings.

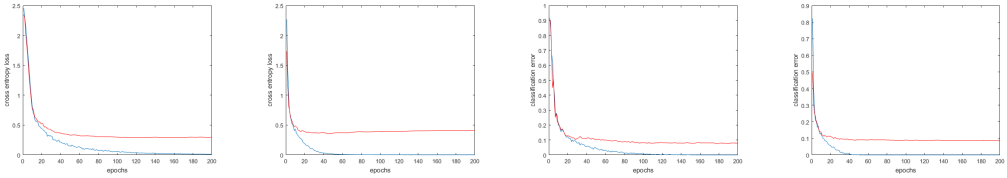
Your answer here

I used the same parameters from the problem 6-f, for all three different activation functions.
From figure 13, we can see that using the same parameters, the sigmoid outperforms than relu both in loss and classification error. And we can also find that the line of sigmoid is more smooth than relu. Figure 14 shows the comparison between the sigmoid and tanh. We can see that, using the same



(a) Performance for loss of sigmoid (b) Performance for loss of Relu (c) Performance for error of sigmoid (d) Performance for error of Relu

Figure 13: Comparison between sigmoid and relu



(a) Performance for loss of sigmoid (b) Performance for loss of tanh (c) Performance for error of sigmoid (d) Performance for error of tanh

Figure 14: Comparison between sigmoid and tanh

parameters, sigmoid still outperforms than tanh. But the lines of tanh activation function is more smooth than sigmoid.