10707 Homework 1

Barun Patra
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
bpatra@andrew.cmu.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

Answer 1 :
Figure 1 shows the general variation of error with increasing model complexity. From left to right, the model moves from a region of under-fitting to a region of over-fitting (ultimately leading to 0 training error, assuming model class is complex enough, else would saturate to a non zero value). The black vertical line shows where the training error saturates (or the most complex model that the training data can support), while the vertical red line shows the place where over-fitting potentially starts (or the most complex model the data distribution can support, i.e the best model for the task).

Answer 2:
Figure 2 shows the variation of error with increasing data size. As we move from left to right, we see moving from a region of over-fitting to saturation. In the left region, the validation error is high, but training error is low (since its easy to over-fit on less data), but as we move to the right, the test error decreases, but training error increases (as we tend to over-fit less).

Answer 3. Since the objective (loss) is to minimize training loss, initially, the model learns

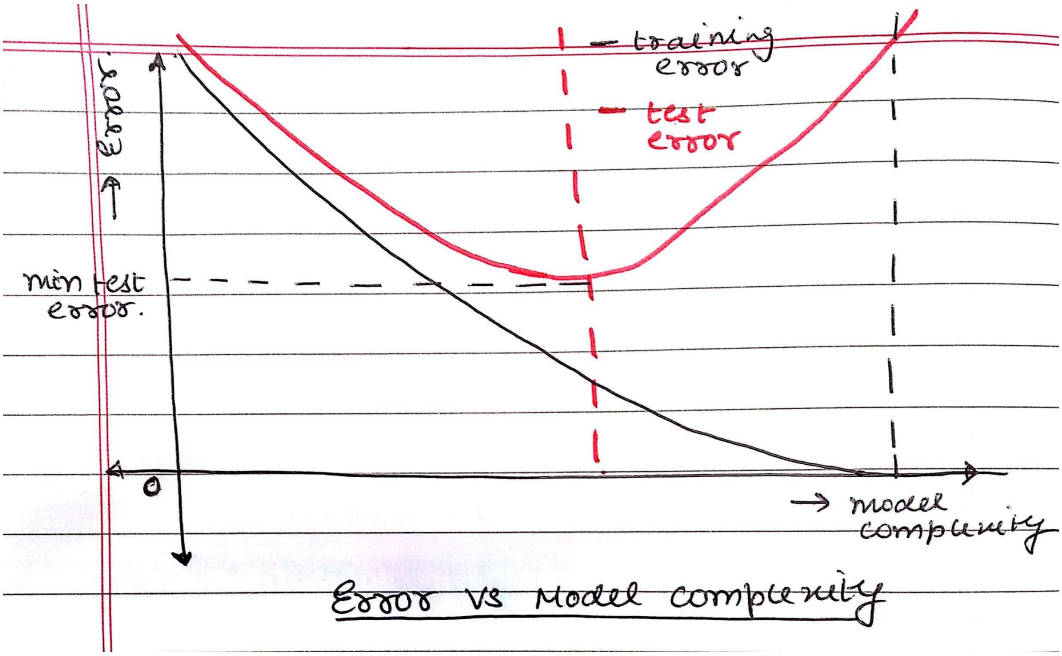


Figure 1: Error vs Model Complexity.

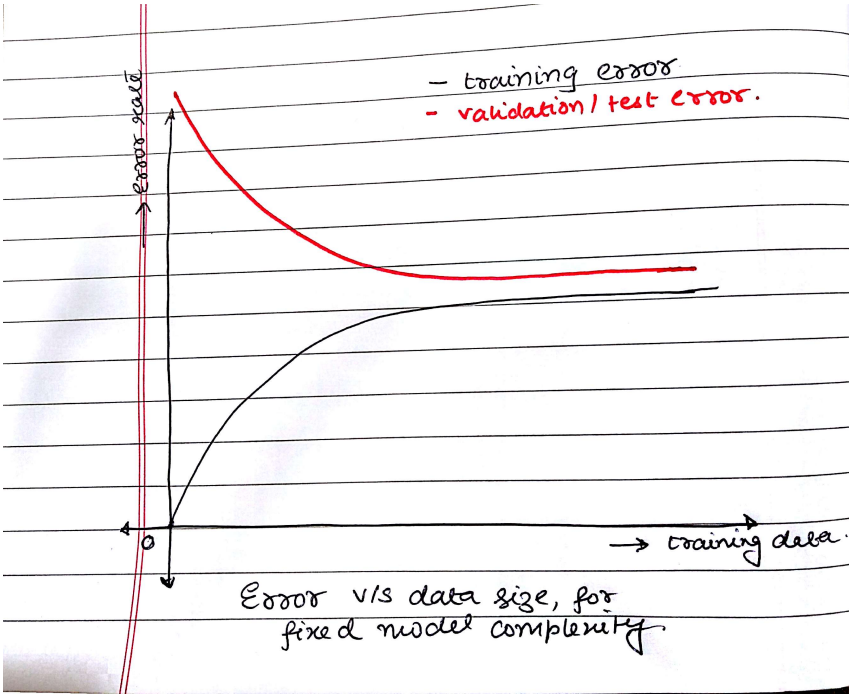


Figure 2: Error vs Increasing Data size

general features, useful for the task. But as the epochs increase, the model starts trying to fit the noise in the training data, instead of learning general features, which in turn is harmful during test time. Thus, early stopping (stopping when we don't see an improvement on the validation set over some epochs) allows the model to stop when it starts fitting noise, thereby allowing for better generalization.

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 Linear Regression :

$$\begin{aligned}\hat{f}(x^*) &= x^*(\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^T y \\ &= \sum_{i=1}^{n} y_i l_i(x^*; \mathcal{X})\end{aligned}$$

Where $l_i(x^*; \mathcal{X}) = [x^*(\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^T]_i$

For K-Nearest Neighbors:

$$\begin{aligned}\hat{f}(x^*) &= \sum_{i=1}^{N} \frac{\mathbb{1}(x_i \in \text{nbr}_k(x^*))}{k} y_i \\ &= \sum_{i=1}^{n} y_i l_i(x^*; \mathcal{X})\end{aligned}$$

Where $\mathbb{1}(x_i \in \text{nbr}_k(x^*))$ is the indicator, denoting if x_i is in the k neighborhood of x^* (i.e among the k closest points to x^* in $\mathcal{X}$, and $l_i(x^*; \mathcal{X}) = \frac{\mathbb{1}(x_i \in \text{nbr}_k(x^*))}{k}$.

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

Answer 1. To show the distribution is normalized:

$$\begin{aligned}\sum_{x\in\{-1,1\}} \mathbb{P}(x|\mu) &= \mathbb{P}(x=-1|\mu) + \mathbb{P}(x=1|\mu) \\ &= \frac{1-\mu}{2} + \frac{1+\mu}{2} \\ &= 1\end{aligned}$$

Answer 2. Mean :

$$\begin{aligned}\mathbb{E}(X) &= \sum_{x\in\{-1,1\}} x\mathbb{P}(X=x|\mu) \\ &= -1\cdot\mathbb{P}(X=-1|\mu) + 1\cdot\mathbb{P}(X=1|\mu) \\ &= \frac{-(1-\mu)}{2} + \frac{(1+\mu)}{2} \\ &= 0\end{aligned}$$

Answer 3: Variance :

$$\begin{aligned}\mathbb{V}(X) &= \mathbb{E}((X-\mu)^2) \\ &= \mathbb{E}(X^2) \\ &= \mathbb{P}(X=-1 \vee X=1) \\ &= 1\end{aligned}$$

Answer 4: Entropy :

$$\begin{aligned}\mathcal{H}(x) &= \sum_{x\in\{-1,1\}} -\mathbb{P}(X=x)\log\mathbb{P}(X=x) \\ &= -[\mathbb{P}(X=-1)\log\mathbb{P}(X=-1) + \mathbb{P}(X=1)\log\mathbb{P}(X=1)] \\ &= -[\frac{1-\mu}{2}\log\frac{1-\mu}{2} + \frac{1+\mu}{2}\log\frac{1+\mu}{2}] \\ &= -\frac{1}{2}[\log\frac{(1-\mu)^{(1-\mu)}\cdot(1+\mu)^{(1+\mu)}}{2}]\end{aligned}$$

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

Say we are trying to minimize $-\log \mathbb{P}(Y|X)$. Then

$$\begin{aligned}
-\log \mathbb{P}(Y|X) &= -\log \prod_{i=1}^{n} \mathbb{P}(y^i|x_i) \\
&= -\sum_{i=1}^{n} \log \mathbb{P}(y^i|x_i) \\
&= -\sum_{i=1}^{n} \log[\mathbb{P}(y \text{ is correct}) \cdot \mathbb{P}(y^i|x_i, y \text{ is correct}) \\
&+ \mathbb{P}(y \text{ is incorrect}) \cdot \mathbb{P}(1-y^i|x_i, y \text{ is incorrect})] \\
&= -\sum_{i=1}^{n} \log[((1-\epsilon) \cdot y(x;w)^{y_i}(1-y(x;w))^{1-y_i}) \\
&+ (\epsilon \cdot y(x;w)^{1-y_i}(1-y(x;w))^{y_i})]
\end{aligned}$$

When ϵ is 0, then the cost function reduces to the usual negative log likelihood loss that we generally use to train models.

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 note that

$$\begin{aligned}
\tanh(a) &= \frac{e^a - e^{-a}}{e^a + e^{-a}} \\
&= \frac{1 - e^{-2a}}{1 + e^{-2a}} \\
&= \frac{1}{1+e^{-2a}} - [1 - \frac{1}{1+e^{-2a}}] \\
&= \frac{2}{1+e^{-2a}} - 1 \\
&= 2\sigma(2a) - 1
\end{aligned}$$

Now, consider the following notation. Let x be the input, W_i and $b_i$the weights and biases connecting hidden layers i to i+1 for the network with σ activation, $W_i^{'}$ and $b_i^{'}$ the weights and biases connecting hidden layers i to i+1 for the network with $\tanh$, z_i and $z_i^{'}$ the weighted inputs to layer i and a_i and $a_i^{'}$ the outputs after activation, o and o' the outputs respectively. Then we have for the network with activation σ,

$$\begin{aligned}
z_1 &= x \cdot W_1 + b_1 \\
a_1 &= \sigma(x \cdot W_1 + b_1) \\
z_2 &= \sigma(x \cdot W_1 + b_1) \cdot W_2 + b_2 \\
a_2 &= \sigma(\sigma(x \cdot W_1 + b_1) \cdot W_2 + b_2) \\
o &= \sigma(\sigma(x \cdot W_1 + b_1) \cdot W_2 + b_2) \cdot W_3 + b_3
\end{aligned}$$

For network with activation $\tanh$, we have

$$
\begin{aligned}
z_1^{'} &= x \cdot W_1^{'} + b_1^{'} \\
a_1^{'} &= 2 \cdot \sigma(x \cdot 2W_1^{'} + 2b_1^{'}) - \mathbb{1} \\
z_2^{'} &= \sigma(x \cdot 2W_1^{'} + 2b_1^{'})2W_2^{'} - \mathbb{1}W_2^{'} + b_2^{'} \\
a_2^{'} &= 2\sigma(\sigma(x \cdot 2W_1^{'} + 2b_1^{'})4W_2^{'} - 2 \cdot \mathbb{1}W_2^{'} + 2b_2^{'}) - \mathbb{1} \\
o^{'} &= \sigma(\sigma(x \cdot 2W_1^{'} + 2b_1^{'})4W_2^{'} - 2 \cdot \mathbb{1}W_2^{'} + 2b_2^{'})2W_3^{'} - \mathbb{1}W_3^{'} + b_3^{'}
\end{aligned}
$$

Thus, for

$$
\begin{aligned}
b_3 &= b_3^{'} - \mathbb{1}W_3^{'} \\
W_3 &= 2W_3^{'} \\
b_2 &= 2b_2^{'} - 2 \cdot \mathbb{1}W_2^{'} \\
W_2 &= 4W_2^{'} \\
b_1 &= 2b_1^{'} \\
W_1 &= 2W_1^{'}
\end{aligned}
$$

We have, $o = o^{'}$, i.e both networks have the same output.

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]

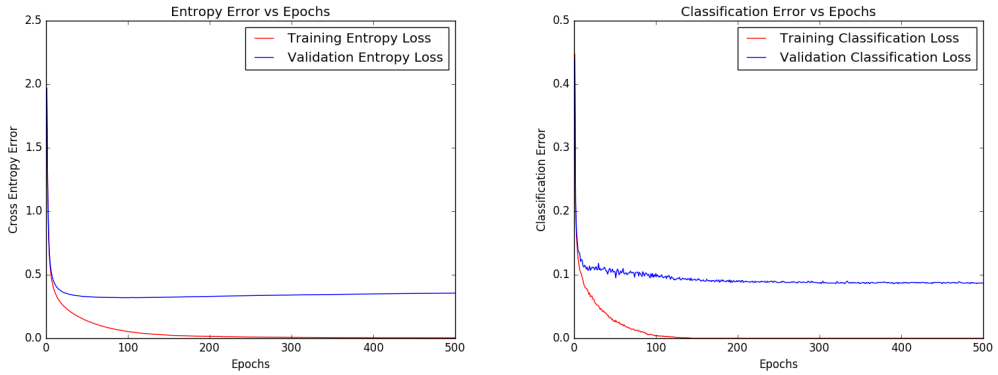
Train a single layer neural network with 100 hidden units (with architecture: 784 → 100 → 10). You should use initialization scheme discussed in class as well as choose a reasonable learning rate (e.g. 0.1). Train the network repeatedly (more than 5 times) using different random seeds, so that each time, 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

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



(a) Cross Entropy Loss for 500 epochs

(b) Classification Loss for 500 epochs

Figure 3: Losses for a single hidden layer (100 units) network for 500 epochs

Figure 3a shows the variation of Cross Entropy Loss of a single layer network across epochs for the training and validation set. We observe that initially, training and validation error go down as the model learns. But then, as the training error approaches 0, the validation error increases. This is to be expected, as with increasing epochs, the model starts fitting noise (starts over-fitting), in which case the validation error increases. Note that the amount of over-fitting is not very high (the validation error does not go up by a large margin).

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?

Your answer here

Figure 3b shows the variation of Classification Error of the same network across epochs for training and validation set. The classification error, by and large, follows the same trend as the Cross Entropy Loss. However, while the validation cross entropy error loss starts increasing around 60 epochs, the classification loss actually goes down till around the 200th epoch, and then mostly stabilizes. This can be attributed to the fact that minor changes in cross entropy may not directly affect the classification (which just checks the argmax). Moreover, since the magnitude of over-fitting is less, consequently, the error does not increase significantly with increasing epochs.

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

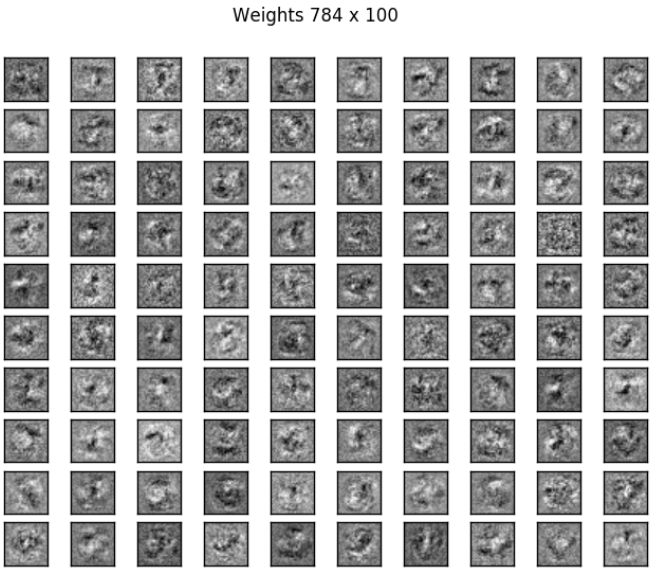


Figure 4: Weights of the first layer

As seen in the figure 4, the weights do learn certain structures after training.

d) Learning rate [5 points]

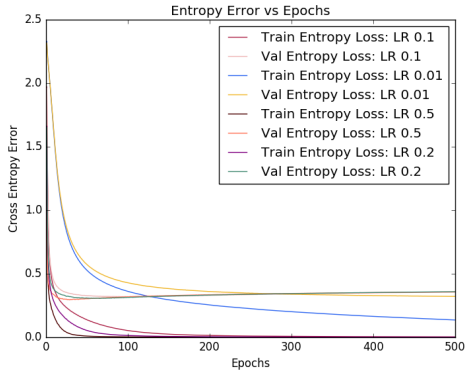
Try different values of the learning rate ϵ. You should start with a learning rate of 0.1. You should then reduce it to .01, and increase it to 0.2 and 0.5. What happens to the convergence properties of the algorithm (looking at both average cross entropy and % Incorrect)? Try momentum of $\{0.0, 0.5,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

Your answer here

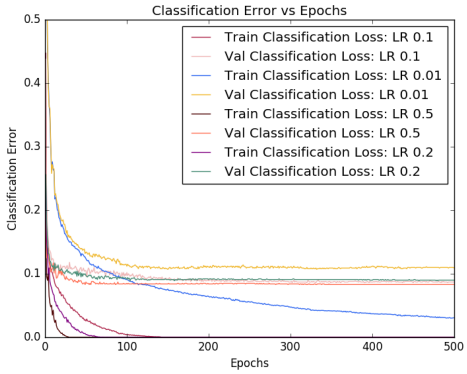
Figure 5a and 5b show the effect of varying the learning rate on the Cross Entropy Loss and the Classification Loss. As can be seen from the figures, the learning rates of 0.5, 0.2 and 0.1 all lead to the convergence of the training loss (to almost 0), while for the learning rate of 0.01, the loss can be seen going down even at 500 epochs. Moreover, for 0.01, the validation loss achieved is lower that that for the others, which may mean that it could, given more epochs, potentially reach a better minimum. In terms of classification accuracies, we see the lr of 0.5 performs the best, while for lr of 0.01, the accuracy is worse. A potential reason could be that the lr 0.01 model needs to train further.

Figure 6a and 6b show the effects of different momentum for the same LR of 0.5. As can be seen from the figure, the classification error is minimum for the case of 0.9 momentum. That said, all the networks converge to approximately the same accuracies (92.30 - 93.50). Thus, momentum helps, but not drastically so, at least for the mnist dataset.

A way to pick the best parameters would be to do a grid search on discrete values, and ran-

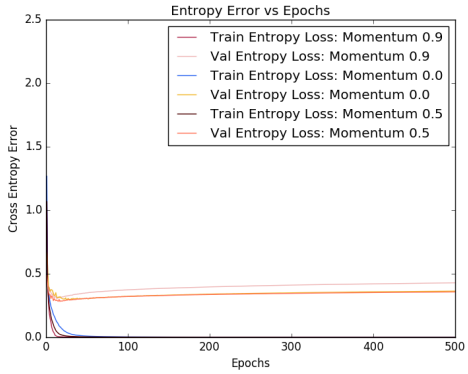


(a) Cross Entropy Loss for different LR

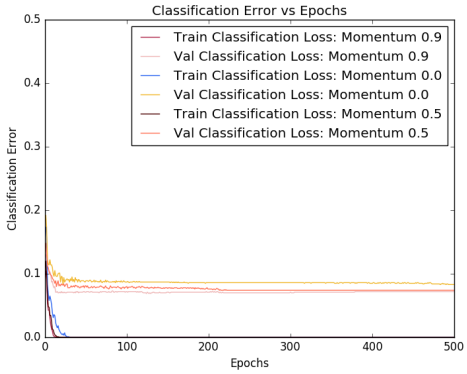


(b) Classification Loss for different LR

Figure 5: Losses for a single hidden layer (100 units) network for 500 epochs with varying LR



(a) Cross Entropy Loss for different Momentum



(b) Classification Loss for different Momentum

Figure 6: Losses for a single hidden layer (100 units) network for 500 epochs with varying Momentum

dom sampling multiple continuous valued parameters for each configuration. That is the approach that was chosen for the validation parts of the question.

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.

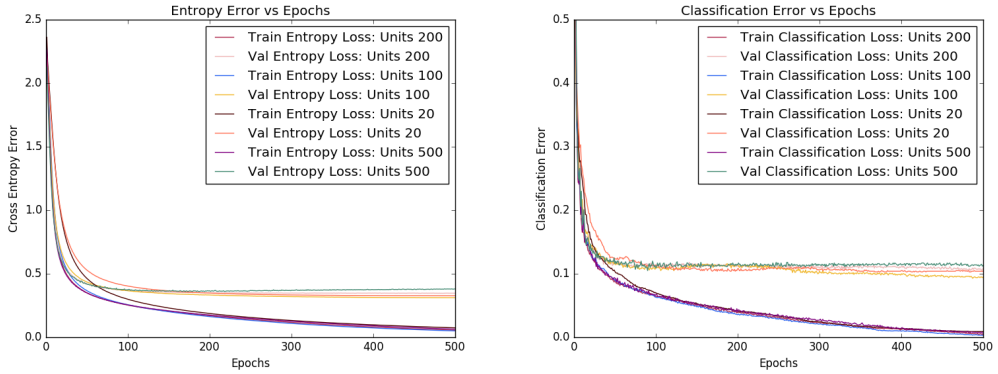
Your answer here

We observe that as we increase the number of hidden units, the loss first decreases, then increases (100 units is optimal). In particular, the validation loss is maximum for 500 units (for lr 0.01 and momentum 0.5). We could say that the tendency to over-fit increases with increasing the hidden layer size, but a concrete conclusion cannot be made (the errors are too close to each other. Particularly, a better random initialization/ better lr could maybe do better).

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

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

- learning rates



(a) Cross Entropy Loss for different hidden layer size (b) Classification Loss for different hidden layer size

Figure 7: Losses for a single hidden layer (100 units) network for 500 epochs with varying hidden layer units

- 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

Units	Epoch	L2	Gamma	Train Loss	Val Loss	Val Acc	Test Loss	Test Acc
200	225	0.0052	0.9160	0.0130	0.1974	0.9470	0.2688	0.9200
50	190	0.0095	0.8750	0.0435	0.2158	0.9380	0.2733	0.9187
50	114	0.0074	0.5970	0.0256	0.2423	0.9340	0.3003	0.9137
100	308	0.0077	0.6970	0.0239	0.2411	0.9320	0.2952	0.9170
100	380	0.0093	0.4640	0.0269	0.2358	0.9310	0.2742	0.9220

Table 1: Different Parameters Grid search for single layer network : Top 5 Models

For finding the best parameters, a grid search was carried out. For layer units of sizes $[50, 100, 200, 500]$, and learning rate of $[0.1, 0.5]$, 4 models per configuration were executed, by sampling an l2 value in the range $0.0001 \leq \text{lr} \leq 0.01$, and sampling a momentum value in the range $0.4 \leq \text{momentum} \leq 0.95$. The learning rate was chosen to be high, as they were shown to be better in the previous parts.

Table 1 shows the Top 5 models resulting from the grid search. The best models all had learning rate of 0.5 and Training Accuracy of nearly 1 and hence those columns have been omitted for brevity. Note that **Gamma** denotes the momentum parameter.

We find that increasing number of units helps, but only to a certain extent, beyond which the models have a high tendency to over-fit (No model with 500 units made it to the top 5). A high LR also helps (No model with 0.1 lr made it to the top 5). The models with low L2 values and high momentum values perform better. Finally, early stopping always helps (the best accuracies were obtained well before the maximum iterations of 500 was achieved for all models), and hence is an effective way of regularizing.

Figure 8 and 9 show the weights for the best single layered model on the validation and

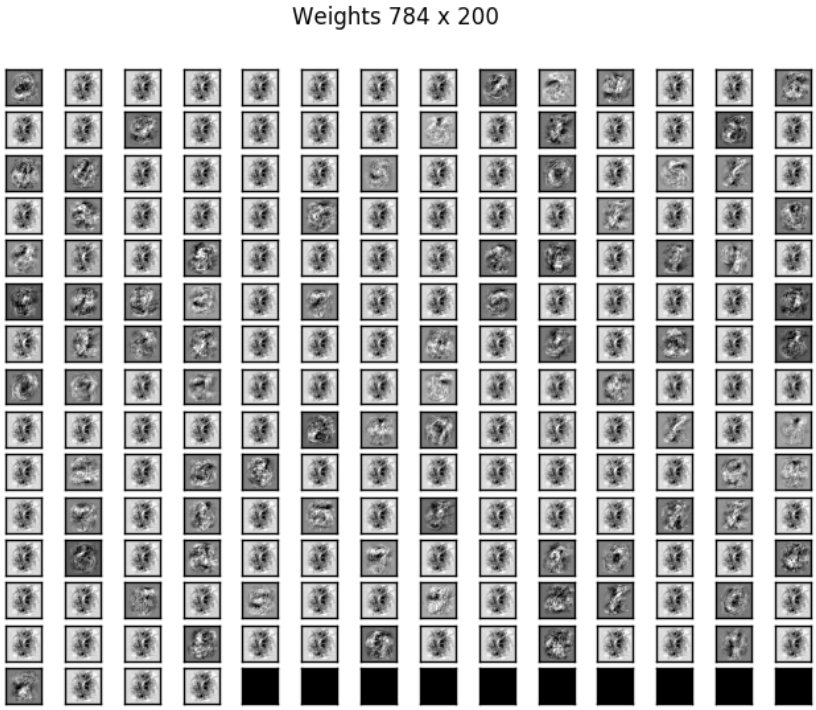


Figure 8: Weights of the first layer for best single layer model (Validation)

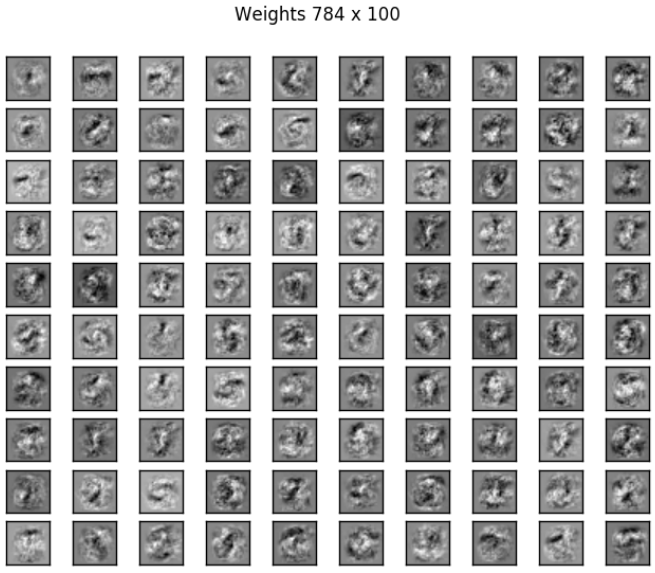


Figure 9: Weights of the first layer for best single layer model (Test)

test set respectively. As can be seen from the image, the weights have some definite structure to them,

and are not random noise. The weight of 9 appear less similar (correlated) than those of 8, which could be an indicator of why it performs better on the test data.

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

H1	H2	Epoch	L2	Gamma	Train Loss	Val Loss	Val Acc	Test Loss	Test Acc
500	200	375	0.0089	0.8320	0.0293	0.2074	0.9530	0.2712	0.9240
200	200	316	0.0070	0.9120	0.0125	0.1721	0.9490	0.2316	0.9310
500	200	219	0.0032	0.8130	0.0035	0.2123	0.9470	0.2870	0.9213
100	50	171	0.0034	0.7340	0.0051	0.2221	0.9450	0.2846	0.9210
200	50	134	0.0015	0.7080	0.0040	0.2351	0.9420	0.3264	0.9150

Table 2: Different Parameters Grid search for a two layer network : Top 5 Models

Table 2 shows the Top 5 models for a grid search for a two layer neural network, using a similar framework as done in the previous question. As can be seen, increasing the depth definitely helps. In particular, both the max accuracy and mean accuracy of the Top 5 models go up for both the validation and test set. Early stopping definitely helps, so does a high value of momentum. Increasing the layer sizes help, but only to a certain extent (note that no 500 x 500 model made it to the Top 5). Judging by these experiments, I would say that a 1 layer network does not outperform a two layer network in terms of generalization capabilities (performance on the test data).

Figures 10 and 11 show the weights of the best performing two layer models for both the validation and test data. As can be seen 11 has fewer similar weights that 10. Perhaps that is why the model generalizes slightly better on the test data. However, compared to 8 (the weights of a single layer network), the weights appear more structured, which makes sense, since a multi layer network allows for structured hierarchical representation of features.

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

Figures 12a and 12b show the effect of batch normalization on the best two layer network model. As is evident from the figures, batch normalization results in much more stable loss curves, with the best performance being very similar (around 94.9 for batch norm model vs 95.3 for the model without).

The effect of batch norm is more pronounced for models that didn't do so well. In particular consider the two worst models encountered during the cross validation stage. Figures 13a and

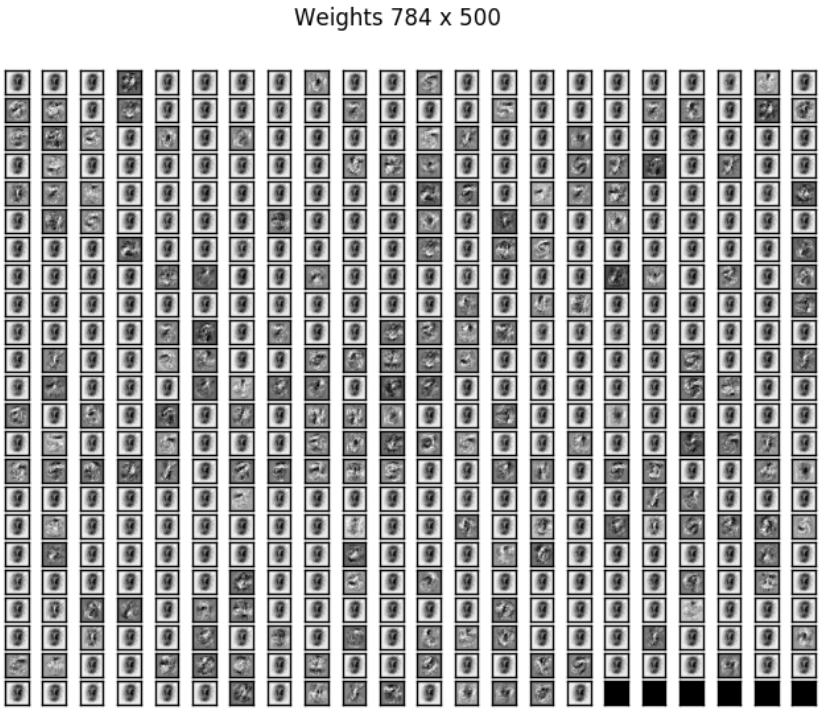


Figure 10: Weights of the first layer for best two layer model (validation)

13b show the losses as a function of the number of epochs. Notice that not only does batch norm lead to a more stable loss, it also leads to a faster and better convergence as well. Consequently tackling the internal covariance shift by batch normalizing seems to lead to a more stable learning system. That said, batch norm doesn't improve accuracy by a huge margin, at least on this dataset.

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

The results of using different activations is given in table 3. Since no explicit grid search was carried out, it is possible that the accuracies reported are not optimal. However, on the test and validation set, we do see that relu out-performs tanh, which in turn outperforms sigmoid for the one layer networks. Moreover, relu is by far the most computationally efficient (around 182s for 500 iterations as opposed to around 200s for sigmoid and 240s for Tanh). Another thing of note was that the lr of 0.5 led to non convergence for both relu and tanh (hence all values have been reported at lr of 0.1). This makes sense for the relu case, since a large gradient could potentially cause the unit to die, and never be able to recover from it.

Model		Validation				Test			
		Loss	New Loss	Acc	New Acc	Loss	New Loss	Acc	New Acc
One Layer*	Relu	0.1974	0.2073	0.9470	0.9420	0.2688	0.2654	0.9200	0.9277
One Layer*	Tanh		0.3169		0.9330		0.3856		0.9223
Two Layer*	Relu	0.1721	0.2175	0.9490	0.9510	0.2316	0.3073	0.9310	0.9233
Two Layer*	Tanh		0.2198		0.9460		0.2962		0.9287

Table 3: Validation and Test Accuracy with Different Activations for the best model

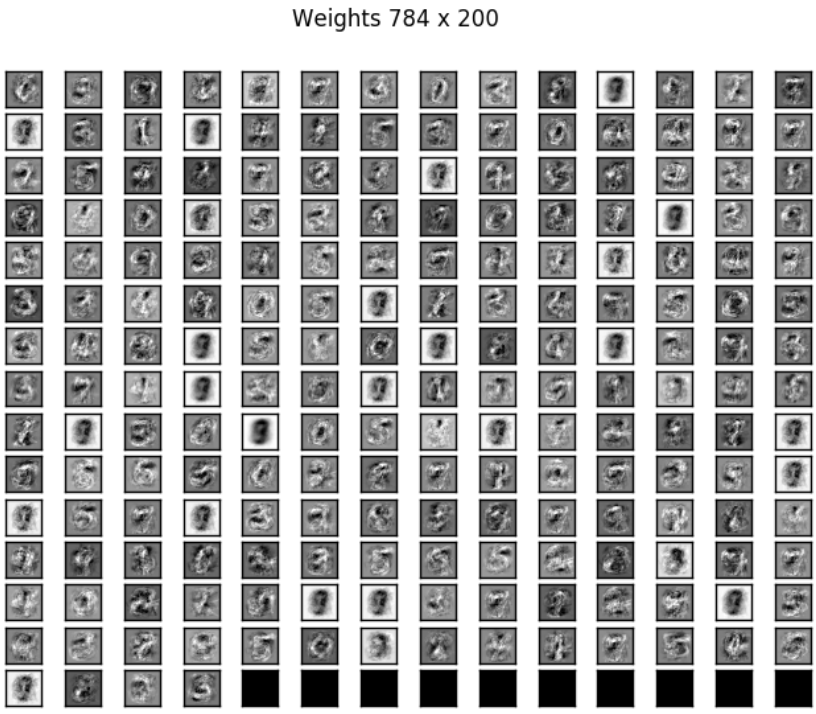
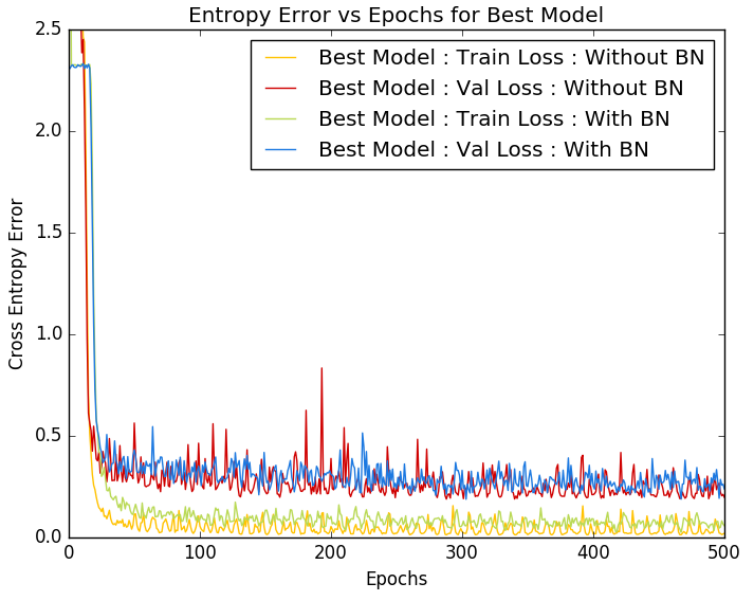
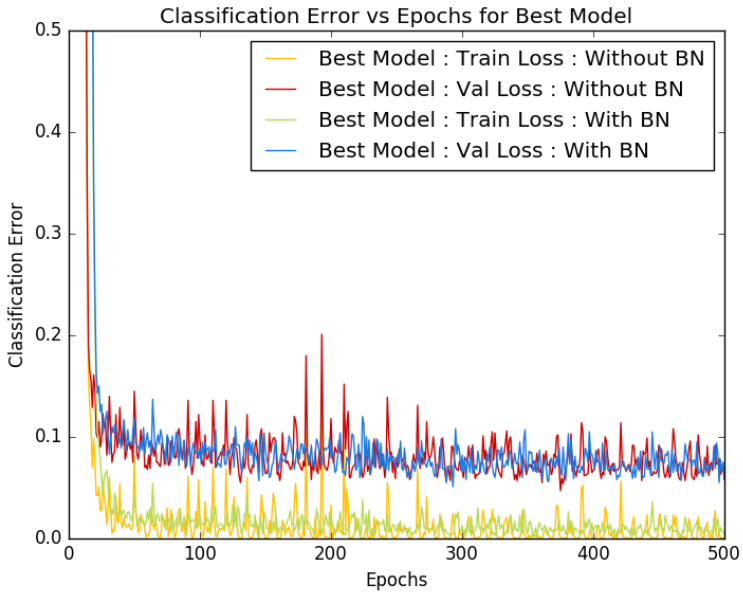


Figure 11: Weights of the first layer for best two layer model (Test)

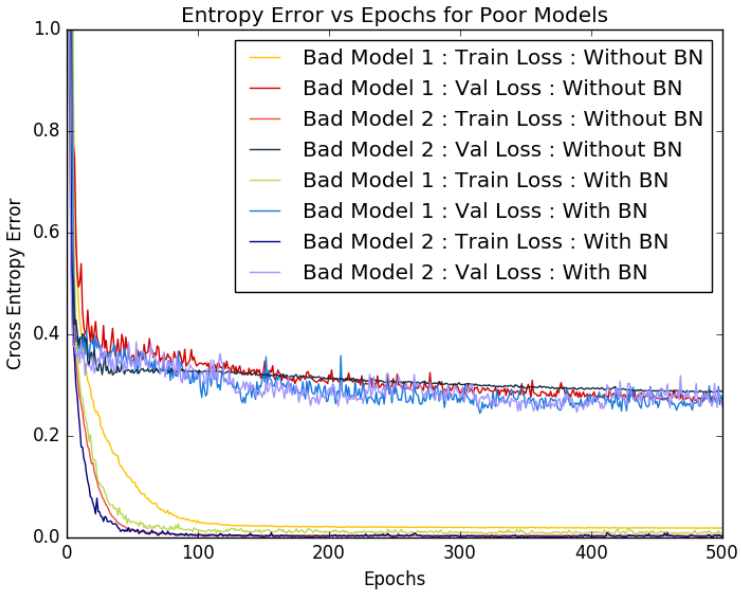


(a) Cross Entropy Loss

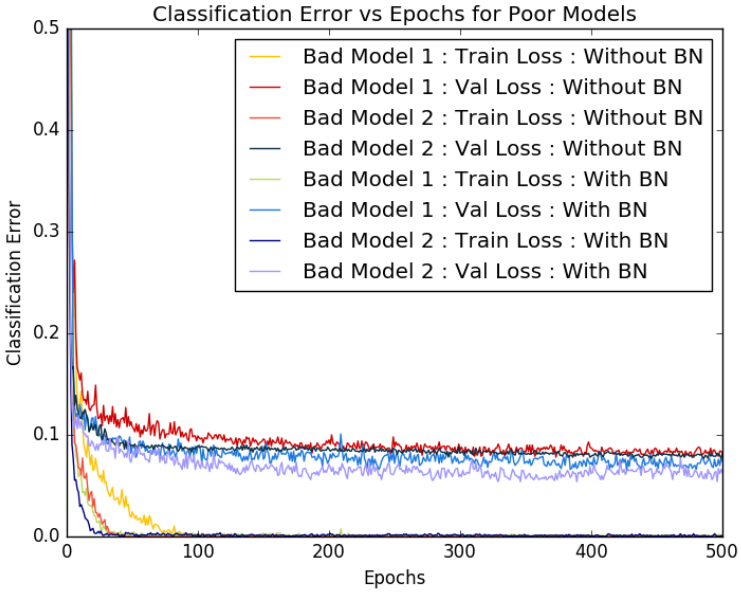


(b) Classification Loss

Figure 12: Losses for the best two layer model with and without Batch Normalization



(a) Cross Entropy Loss



(b) Classification Loss

Figure 13: Losses for the two worst two layer model with and without Batch Normalization