6.867 PSET 3

Author: made you look

November 2015

1 Neural Networks

Gradient Calculation First we review notation. We say that $x_t^{(i)}$ corresponds to the value of the t^{th} input node of example $x^{(i)}$ for $1 \leq t \leq D$ (the training vectors are D dimensional). Say that $a_m^{(1,i)}$ corresponds to the input to the m^{th} hidden node when the neural network is fed $x^{(i)}$ as input; here $1 \leq m \leq M$, i.e., M is the number of nodes of the hidden layer. Say $z_m^{(1,i)} = \sigma(a_m^{(1,i)})$ is the output of the m^{th} hidden node. Say that K is the number of output nodes (i.e. $y^{(i)}$, the vector corresponding to the training label, is K-dimensional) and that $a_k^{(2,i)}$ refers to the input to output node k and that $z_k^{(2,i)} = \sigma(a_k^{(2,i)})$ is the output of the output node (for $1 \leq k \leq K$). Additionally, the weights between layers have form $w_{ab}^{(1)}$, where $1 \leq a \leq D$ and $1 \leq b \leq M$ (where $w_{ab}^{(1)}$ corresponds to a weight connecting input node a to hidden node b) and $w_{bc}^{(2)}$, where $1 \leq b \leq M$ and $1 \leq c \leq K$ (where $w_{bc}^{(2)}$ corresponds to a weight connecting hidden node b to output node c).

Note that our objective function was $J(w) = l(w) + \lambda(|w^{(1)}|_F^2 + |w^{(2)}|_F^2)$ where

$$l(w) = \sum_{i=1}^{N} [\sum_{k=1}^{K} -y_k^{(i)} \log(h_k(x^{(i)}, w)) - (1 - y_k^{(i)}) \log(1 - h_k(x^{(i)}, w))]$$

Problem 2 Part 1: Calculating for 2nd layer weights First, we find what $\frac{dh_k}{dw_{bc}^{(2}}$ is. Note that $h_k(x^{(i)}, w) = \sigma(a_k^{(2,i)}$ and so $h_k(x^{(i)}, w) = \sigma(\sum_{b=1}^{M} w_{bk}^{(2))} z_b^{(1,i)})$

We know that $\frac{d\sigma(x)}{dx} = \sigma(-x)\sigma(x)$. By chain rule we get that $\frac{dh_k}{dw_{bc}^{(2)}} = \frac{dh_k}{da_k^{(2,i)}} \frac{da_k^{(2,i)}}{dw_{bc}^{(2)}}$

We know that $\frac{dh_k}{da_k^{(2,i)}} = \sigma(a_k^{(2,i)})\sigma(-a_k^{(2,i)})$. Then our final expression is $\frac{dh_k}{dw_{bc}^{(2)}} = \sigma(a_k^{(2,i)})\sigma(-a_k^{(2,i)}) z_b^{(1,i)}$

Additionally, we know by chain rule that $\frac{d(\log(h_k))}{dw_{bc}^{(2)}} = \frac{d(\log(h_k))}{dh_k} \times \frac{dh_k}{dw_{bc}^{(2)}} = \frac{1}{h_k} \times \frac{dh_k}{dw_{bc}^{(2)}} = \sigma(-a_k^{(2,i)}) z_b^{(1,i)}$ and that

$$\frac{d(\log(1 - h_k))}{dw_{bc}^{(2)}} = \frac{d(\log(1 - h_k))}{dh_k} \times \frac{dh_k}{dw_{bc}^{(2)}} = -\frac{1}{1 - h_k} \times \frac{dh_k}{dw_{bc}^{(2)}} = \sigma(a_k^{(2,i)}) z_b^{(1,i)}$$

(in the last simplification we use the fact that $\sigma(-a_k^{(2,i)}) = 1 - \sigma(a_k^{(2,i)})$ - this is a well-known fact). Then, taking into account that $\frac{d(\lambda|w^{(2)}|_F^2}{dw_{bc}^{(2)}} = 2\lambda w_{bc}^{(2)}$ the derivative of J with respect to $w_{bc}^{(2)}$ is

$$\sum_{i=1}^{N} [(-y_k^{(i)} \sigma(-a_k^{(2,i)}) + (1 - y_k^{(i)}) \sigma(a_k^{(2,i)})) z_b^{(1,i)}] + 2\lambda w_{bc}^{(2)} = \sum_{i=1}^{N} [(z_k^{(2,i)} - y_k^{(i)}) z_b^{(1,i)}] + 2\lambda w_{bc}^{(2)}$$

where in the last equation we use the fact that $z_k^{(2,i)} = \sigma(a_k^{(2,i)})$ and used that $1 - \sigma(a_k^{(2,i)}) = \sigma(-a_k^{(2,i)})$.

Calculating for 1st layer weights First we calculate $\frac{dh_k}{dw_{ab}^{(1)}}$. We have $h_k(x^{(i)}, w) = \sigma(a_k^{(2,i)})$ and we know that

$$a_k^{(2,i)} = \sum_{m=1}^{M} w_{mk}^{(2)} \sigma(a_m^{(1,i)})$$

so $\frac{da_k^{(2,i)}}{dw_{ab}^{(1}} = w_{bk}^{(2)} \frac{d\sigma(a_b^{(1,i)})}{dw_{ab}^{(1)}}$. Then $\frac{d\sigma(a_b^{(1,i)})}{dw_{ab}^{(1)}} = \frac{d\sigma(a_b^{(1,i)})}{da_b^{(1,i)}} \times \frac{da_b^{(1,i)}}{dw_{ab}^{(1)}}$.

We know that $\frac{d\sigma(a_b^{(1,i)})}{da_b^{(1,i)}} = \sigma(a_b^{(1,i)})\sigma(-a_b^{(1,i)}) = z_b^{(1,i)}(1 - z_b^{(1,i)})$

and that $a_b^{(1,i)} = \sum_{d=1}^{D} w_{db}^{(1)} x_d^{(i)}$ so $\frac{da_b^{(1,i)}}{dw_{ab}^{(1)}} = x_a^{(i)}$.

Then we get that $\frac{da_k^{(2,i)}}{dw_{ab}^{(1}} = w_{bk}^{(2)} z_b^{(1,i)}(1 - z_b^{(1,i)})x_a^{(i)}$. Finally, we get that

$$\frac{dh_k}{dw_{ab}^{(1)}} = \frac{d\sigma(a_k^{(2,i)})}{da_k^{(2,i)}} \times \frac{da_k^{(2,i)}}{dw_{ab}^{(1)}} = \sigma(a_k^{(2,i)})\sigma(-a_k^{(2,i)})w_{bk}^{(2)} z_b^{(1,i)}(1 - z_b^{(1,i)})x_a^{(i)} = z_k^{(2,i)}(1 - z_k^{(2,i)})w_{bk}^{(2)} z_b^{(1,i)}(1 - z_b^{(1,i)})x_a^{(i)}$$

We can plug in this value and easily use analogous steps that we had shown for calculating the derivatives with respect to second layer weights to get the final expression for the derivative of J with respect to $w_{ab}^{(1)}$

$$\sum_{i=1}^{N}[\sum_{k=1}^{K}(z_k^{(2,i)} - y_k^{(i)})w_{bk}^{(2)} z_b^{(1,i)}(1 - z_b^{(1,i)})x_a^{(i)}] + 2\lambda w_{ab}^{(1)}$$

Vectorizing the gradient Using loops in code to iterate over each training example individually can be very slow. It's of interest to eliminate having to loop through examples and simply compute the gradient using matrix operations. We will discuss how to do this using some matlab operations since that is what we used. We define the following matrices: Z is the MxN matrix such that $Z_{b,i} = z_b^{(1,i)}$. P is the NxK matrix such that $P_{i,k} = z_k^{(2,i)} - y_k^{(i)}$. Additionally $W1$ is the DxM matrix such that $W1_{d,m} = w_{d,m}^{(1)}$ and $W2$ is the MxK matrix such that $W2_{m,k} = w_{m,k}^{(2)}$. We'll also denote $D_{W1}J$ to be the DxM gradient matrix of $W1$ with respect to J and $D_{W1}J$ to be the MxK gradient matrix of $W2$ with respect to J. Also note that X and Y are the NxD and NxK dimensional matrices corresponding to the training data.

First, note that $D_{W2}J = ZP + 2\lambda W2$. To show this we must show that the $(b,c)^{th}$ entry of $D_{W2}J$ has value $\sum_{i=1}^{N}[(z_k^{(2,i)} - y_k^{(i)})z_b^{(1,i)}] + 2\lambda w_{bc}^{(2)}$. It's clear that the $(b,c)^{th}$ entry of $2\lambda W2$ is $2\lambda w_{bc}^{(2)}$; furthermore by matrix multiplication we must have that the $(b,c)^{th}$ entry of ZP is $\sum_{i=1}^{N}[(z_k^{(2,i)} - y_k^{(i)})z_b^{(1,i)}]$ and thus the result is true.

Before we move on to calculating $D_{W1}J$, we first review the $.*$ operator. The operation $A.*B$ in matlab for two matrices A and B of the same dimensions returns a matrix of the same dimensions for which each entry is the product of the two entries in A and B which are at the same positions. Then we say $D_{W1}J = X^T((P(W2^T)).*(Z.*(1-Z))^T) + 2\lambda W1$. Doing the according matrix multiplications and performing the analogous procedure as we did for checking for $D_{W2}J$ to check that this matrix is in fact the gradient matrix (i.e. you have to check that the $(a,b)^{th}$ entry of $X^T((P(W2^T)).*(Z.*(1-Z))^T) + 2\lambda W1$ is in fact the derivative of J with respect to $w_{a,b}^{(1)}$, which we computed above. This formula can be verified by performing the analogous matrix computations as we had done above; we only include this formula to show how this computation is done in the first place.

Finally, the matrices Z and P, upon which all of our matrix computations above hinge, can be computed using purely matrix operations (it is far more trivial to figure out how this is done so we won't discuss it). The $.*$ operator is supported in the language we worked in, so all of the operations above can be done with matrix operations.

Problem 2 Part 2: Implementation of Two-Layer Neural Network The two layer neural network was implemented using final cost function stated below. $J(w) = l(w) + \lambda(|w^{(1)}|_F^2 + |w^{(2)}|_F^2)$ where

$$l(w) = \sum_{i=1}^{N}[\sum_{k=1}^{K} -y_k^{(i)} \log(h_k(x^{(i)}, w)) - (1 - y_k^{(i)}) \log(1 - h_k(x^{(i)}, w))]$$

In order to find the optimal value of the weight matrix that minimized the value of the final cost function, the gradient descent procedure was used and where the analytical expressions of the gradient of the final cost function found in the previous part were necessary. This was because some of the datasets that our implementation would be tested on would be very large and therefore take too much time to numerically calculate the gradient of the final cost function at each point.

Problem 2 Part 3: Stochastic Gradient Descent Unlike the batch gradient descent method where you run through all the samples in the training set to do a single update for a parameter in a particular iteration, the stochastic gradient descent uses only one training sample from the training set to do the update for a parameter in a particular iteration. In the case of a large number of training samples, running the batch gradient descent may take a really long time because in every iteration, when the values of the parameters are being updated, you have to run through the complete training set. In this case, the stochastic gradient descent will be much faster because only one training sample is used and the guess is improved right away from the first sample. The graphs for Figures A and B show a scatterplot of the classification error rate on the

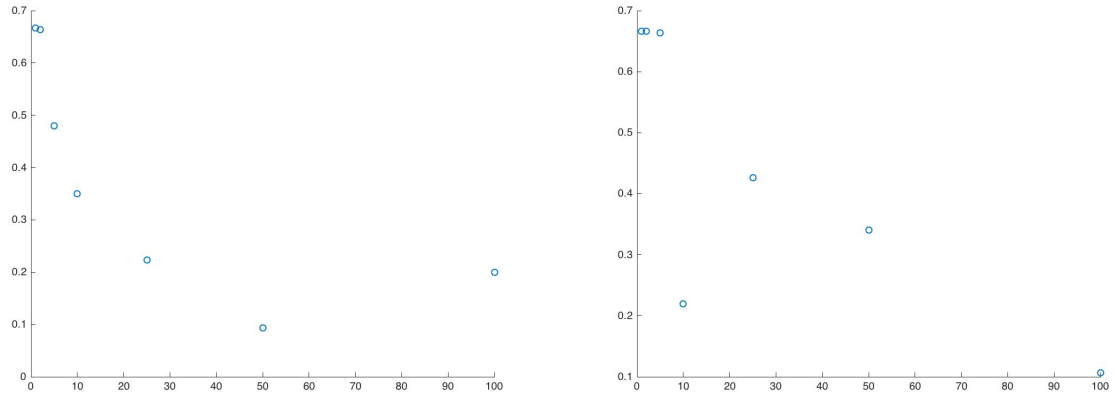


Figure 1: Figure A: Classification Error Rate for Toy 1 Validation Set Using Stochastic Gradient Descent. Figure B: Classification Error Rate for Toy 2 Validation Set Using Stochastic Gradient Descent

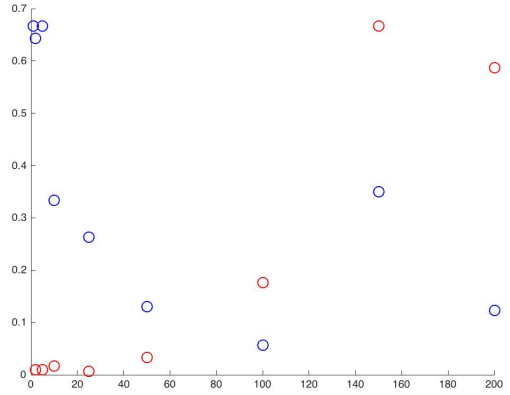


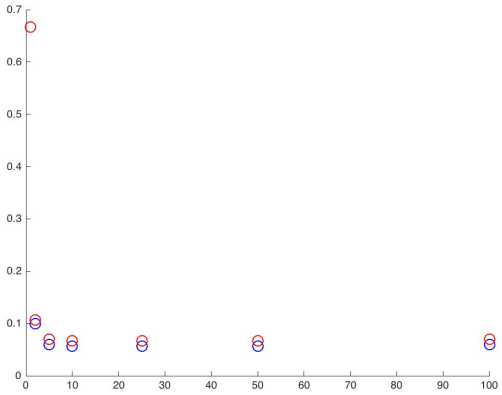
Figure 2: Figure C: Comparison Classification Error Rate for Stochastic and Batch Gradient Descent Approaches on Toy 1 Validation Set where the Batch Gradient Descent is represented in red and the Stochastic Gradient Descent is represented in blue

toymulticlass1dataset in which the hidden nodes that were tested range in value from 1 to 100 for a lambda value of 0. As depicted in the graphs, the classification error rate is initially high for small values of the number of hidden nodes, becomes smaller when the number of hidden nodes is around 50, and rises back up as the number of hidden nodes become 100. On the contrary, the classification error rate fluctuates immensely for the toymulticlass2 dataset as seen in Figure B in which the hidden nodes that were tested ranged in value from 1 to 100 and lambda value of 0. The classification error starts out pretty high for small values of the number of hidden nodes, drops drastically when the number of hidden nodes equals 10, shoots up again when the number of hidden nodes is between 20 and 50, and finally drops back down when the number of hidden nodes becomes 100. It turns out however, that testing the stochastic gradient descent procedure multiple times on both the toy data sets led to some very different trends. This is likely due to the fact that different values of the number of hidden nodes require a different convergence criterion. After decreasing the convergence criterion for several values of the number of hidden nodes, the classification rate for those values became smaller as well. The classification error rate obtained using the stochastic gradient descent approach was compared with the error rate obtained using the batch gradient descent method on the toymulticlass1 dataset. The resulting scatterplot is shown in Figure A where the points in red represent the classification error rates obtained using the batch gradient descent method and the points in blue represent the classification error rates obtained using the stochastic gradient descent method. In general, it is apparent from the graph above that the batch gradient descent approach produces much smaller error rates for small values of the number of hidden nodes. This is likely due to the fact that the batch gradient descent approach is able to see the entire training set and thus produce a more accurate update at each step, leading to a better prediction model and a lower classification error rate.

Problem 2 Part 4: Testing of Neural Network Code The neural network code implemented in the previous part was tested on two datasets: toymulticlass1 and toymulticlass2. For each of the datasets, the training set was used to find the optimal w1 and w2 matrices for various values of M, the number of hidden nodes. Each of these pairs of optimal matrices, w1 and w2, were tested along with the corresponding value of M on the validation set for each of toy data set. For each of these of the M's that were tested on the validation set, the classification error rate was measured. The w1 and w2 matrices along with their corresponding M value that resulted in the lowest classification error rate on the validation set were tested on the test set. Figure C illustrates the classification error rate for various values of M ranging from 1 to 200 for the toymulticlass1 validation dataset. As can be seen, the classification error rate is initially pretty high when M = 1, but drops down quickly as the value of M is between around 2 to 25, and then steadily increases once again for larger values of M. This makes sense intuitively because a small number of hidden nodes would likely not be enough to produce an accurate enough weight matrix to train the neural network while too many hidden nodes would introduce the issue of overfitting and thus result in high classification error rates on the validation set. The final error on the toymulticlass1 testing dataset turned out to be 0.0133 for a value of M at 2. As explained above, the weight matricies and their corresponding M producing the lowest classification error rate on the validation set were the ones used on the testing set. Like what was done on the toymulticlass1 dataset, various values of M ranging from 1 to 200 were tested on the toymulticlass2 dataset to find the corresponding classification error rates. As can be seen in Figure D, a similar parabolic pattern was seen for the classification error rate as the toymulticlass1 dataset, only this time, slightly higher values of M resulted in the lowest classification error rates. This graph also displays the classification error rates of the toymulticlass2 dataset for both the training and validation sets of various values of M ranging from 1 to 200. The classification error rates from the training set are in blue and those from the validation set are in red. As evident in the graph, the classification error rates for most values of M on the training set are smaller than those on the validation set. This is to be expected since the weight matrix tested on the validation set was itself computed to minimize the loss function on the training set data. The final error on the toymulticlass2 testing dataset turned out to be 0.0867 for a value of M at 10. This classification error rate was quite higher than that observed on the toymulticlass1 testing dataset.

Problem 2 Part 5: MNIST Data After testing our existing implementation of the 2 layer neural network on the MNIST data set, it was evident that the current approach was way too slow to produce any usable results. Specifically, the existing approach was slowing down the whole process by using a considerable number of for loops. This was a problem because the MNIST dataset was huge with each X having 785 dimensions in direct contrast to the 2 that the toy dataset had. In order to speed up the process, all for loops were eliminated and the implementation was completely vectorized.

Problem 2 Part 6: Application of Neural Network on MNIST Dataset Similar to the toy data set, various values of M in the range of 1 to 200 were tested on the MNIST data set with the lambda value of the regularization term equal to 0 using the modified implementation of the 2 layer neural network as described in the previous part. Figure F displays a parablic pattern for the classification error rate with too low and too high values of M possessing high classification rates while those in the range of 10-50 producing the lowest classification rates. The classification error rates for most values of M on the validation set were higher than those on the training set which was expected. The difference between the classification error on the training set and validation set is higher for smaller values of M and decreased as the value of M increases. However, this is still fine due to the fact that the error rate is lowest for these smaller values of M and thus as M increases, the vector matrix is pretty inaccurate for both the training set and the validation set, so the trends on higher values of M are less meaningful to us. The final error on the MNIST testing dataset turned out to be 0.0920 for a value of M at 50. Figure G. displays the effect of testing out different values of lambda in the regularization term of the final cost function and the resulting classification error rate that is achieved on the MNIST testing set using the batch gradient descent. The optimal lambda value out of those that were tested was 0.1 which resulted in a classification error of 0.0720 on the MNIST testing set. As is evident based on the graph, too large values of lambda result in high classification errors because the values of the elements in the w matrix tends to move towards 0 since the square of the norm of the w matrix is heavily penalized. Therefore, the resulting w matrix is not accurate enough to represent the neural network model well and thus there is a high classification error when tested on the MNIST testing set. The stochastic gradient descent approach was also tested out on the MNIST Data Set with the lambda value of the regularization term equal to 1. A constant step factor was used. Figure H shows the classification errors obtained from the MNIST Training dataset and the MNIST Validation dataset for a lambda value of 0, a step-size of 0.01, and a convergence threshold of 0.001. As is evident from the graph, the classification error is initially very high for small values of the number of hidden nodes but drops significantly as the number of hidden nodes are increased. The performance of the stochastic gradient descent on the MNIST dataset is also comparable to that of the batch gradient descent. Finally, various values of the stepsize were tested out on the MNIST dataset using the batch gradient descent method with the lambda value of the regularization term equal to 1. Figure I displays the various stepsize values tested along with the corresponding classification error rate on the MNIST test dataset. Large step-sizes lead to larger classification error rates on the MNIST test dataset while these classification error rates decrease as the step size is lowered.



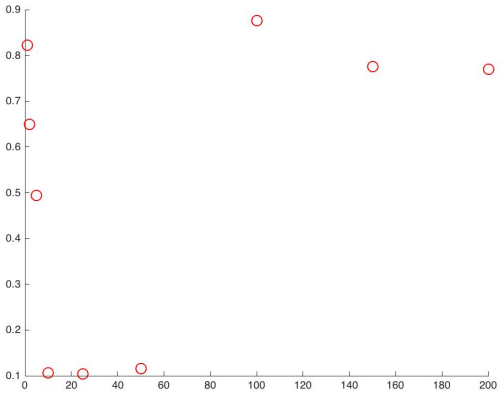
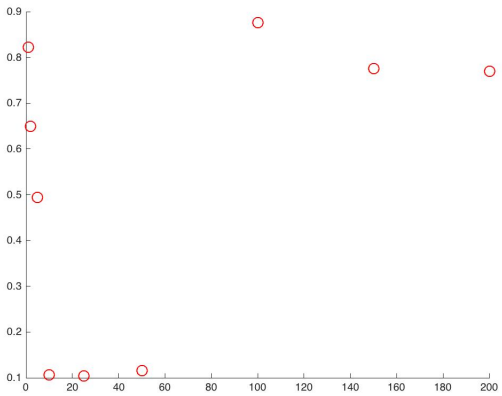


Figure 3: Figure D: Classification Error Rate Comparison using Batch Gradient Descent Between Toy 2 Training and Validation Sets where Blue Represents the Training Error and Red Represents the Testing Error where the number of hidden nodes are varied. Figure E: Classification Error Rate using Batch Gradient Descent on MNIST Validation Dataset where the number of hidden nodes are varied



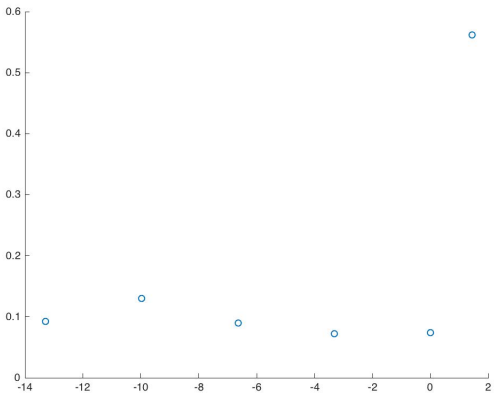


Figure 4: Figure F. Number of Hidden Nodes vs Classification Error Rate using Batch Gradient Descent on MNIST Training and Validation Datasets where the number of hidden nodes are being varied. Figure G. Log2(λ) vs. Classification Error Rate on MNIST Testing Dataset Using the Batch Gradient Descent Approach

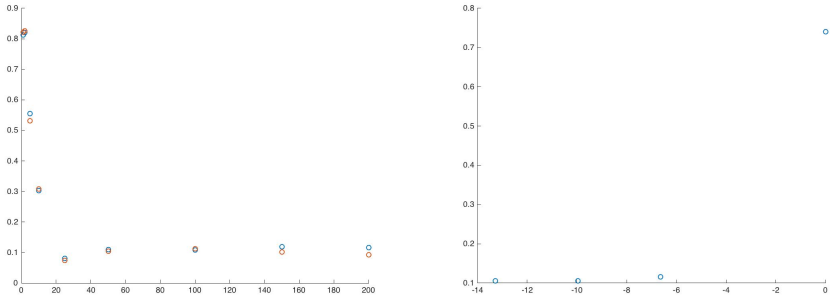


Figure 5: Figure H. Number of Hidden Nodes vs. Classification Error Rate on MNIST Training and Validation Datasets Using the Stochastic Gradient Descent Approach where the Training Error is Represented in Red and the Validation Error is Represented in Blue Figure I. Log2(Stepsize) vs. Classification Error Rate on MNIST Testing Datasets Using the Batch Gradient Descent Approach

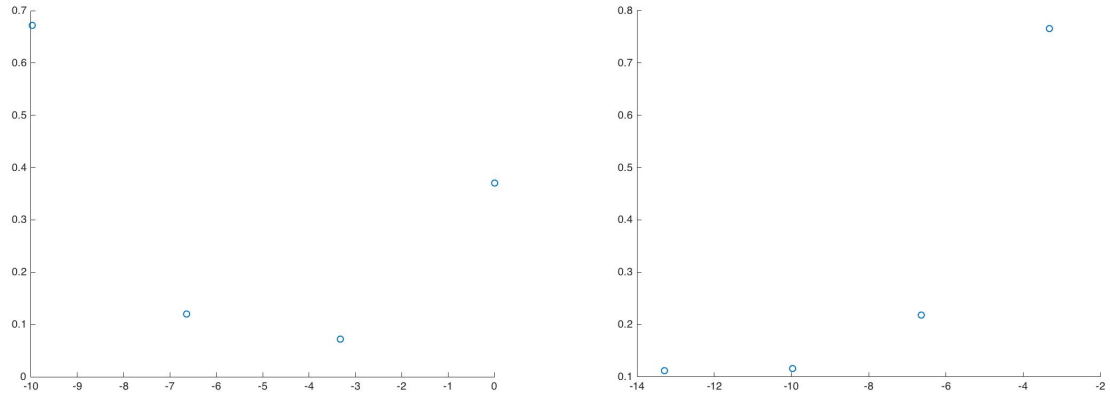


Figure 6: Figure J: log2(stepSizes) vs Classification Error Rate on MNIST Testing Dataset Using the Stochastic Gradient Descent Approach with Lambda = 0. Figure K: log2(λ) vs Classification Error Rate on MNIST Testing Dataset Using the Stochastic Gradient Descent Approach with StepSize = 0.01

Finally, we note that throughout this problem set, we used a fixed step size for stochastic gradient descent. We decided to use this step size instead of a decaying step size as for regular stochastic gradient descent because we tried several different decaying step size functions and they led to very poor classification. We provide the classification errors for choosing a step size function $\frac{1}{2n}$, where n is the current iteration number. The resulting error rates fluctuated greatly and were at best around 0.6 which was clearly not good. When using a fixed, constant step size, as done in all the graphs above (for both MNIST and toy data), we get much better performance.

We note that in general the statistics for both stochastic gradient descent and batch gradient descent are quite similar as can be seen from the graphs. This is likely due to the fact that we're using constant step size for both; differences can probably be seen if decaying step size is used for stochastic gradient descent; however, as noted above, we determined that using a fixed step size was the optimal choice over all of the different step size functions we tested. Finally the convergence threshold was empirically determined by looking at the classification error rates on the different datasets.

The lambda values were chosen between 0 and 1. Having a lambda value of 0 would prevent the model from generalizing and having too high of a lambda value would increase the classification error rate since it is very likely that it would miss the minimum. The stepsize was chosen for most plots to be around 0.01. Too small of a stepsize led to easy convergence and too large of a stepsize led to a high classification error rate since the minimum could be easily missed. The hidden nodes tested were between 1 to 200 where a small value for the hidden nodes would generally not be enough to train the model while a large value for the hidden nodes would lead to overfitting.