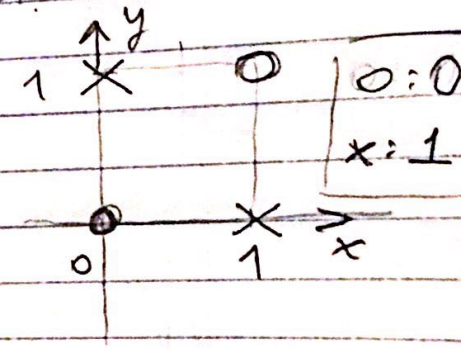
ECE 271B HW#2

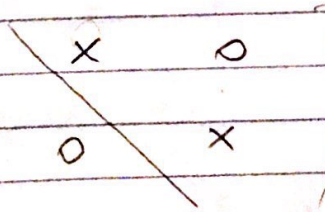
Problem 1)

a) The decision function $h(x)$ outlined in the problem will generate a linear decision boundary. Therefore it will struggle with an underlying function f that is non linear. For $f: \{0,1\} \times \{0,1\} \to \{0,1\}$ a very simple example of f is the XOR function.

XOR	x	y	Class
	0	0	0
	0	1	1
	1	0	1
	1	1	0



As it can be seen on the coordinate axis, there is no linear line that can seperate the two classes without error. The best we can do is:



A decision line such as this which will label one point incorrectly. As there are 4 points, this will correspond to 0.25 probability of error

b) M classes, $g_i(x) = w_i^T x + b_i$, $i = 1, \ldots, M$

assign x to class i if $x \in R_i$, where

$$R_i = \{x \mid g_i(x) \geq g_j(x), \forall j \neq i\}$$

Show that if $x_1 \in R_i$, $x_2 \in R_i$, $\lambda x_1 + (1-\lambda) x_2 \in R_i$ $\forall 0 \leq \lambda \leq 1$

We have to show that for any point x_{any} s.t. $x_{any} = \lambda x_1 + (1-\lambda) x_2 \in R_i$:

$g_i(x_{any}) \geq g_j(x_{any})$, $\forall j \neq i$. $x_{any} = \begin{bmatrix} \lambda x_1 \\ (1-\lambda) x_2 \end{bmatrix}$

$\forall j \neq i$ (1)

About x_1 and x_2 we know $g_i(x_1) \geq g_j(x_1)$ & $g_i(x_2) \geq g_j(x_2)$

$g_i(x_{any}) = g_i(\lambda x_1 + (1-\lambda)x_2) = \lambda w_i^T x_1 + (1-\lambda) w_i^T x_2 + b_i$

$= \lambda w_i^T x_1 + \lambda b_i + (1-\lambda) w_i^T x_2 - \lambda b_i + b_i$

$= \lambda (w_i^T x_1 + b_i) + (1-\lambda)(w_i^T x_2 - b_i)$

$= \lambda g_i(x_1) + (1-\lambda) g_i(x_2) \geq \lambda g_j(x_1) + (1-\lambda) g_j(x_2)$

Because of (1) and $0 \leq \lambda \leq 1$ conditions

$\lambda g_j(x_1) + (1-\lambda) g_j(x_2) = \lambda (w_j^T x_1 + b_j) + (1-\lambda)(w_j^T x_2 + b_j) =$

$= \lambda g_j(x_1) + (1-\lambda) g_j(x_2) = g_j(x_{any}), \forall j \neq i$

Therefore we proved that $g_i(x_{any}) \geq g_j(x_{any}), \forall j \neq i$, $\boxed{x_{any} \in R_i}$

Problem 2

$g(x) = w^T x + b$, $g(x) = 0$ hyperplane.

a) E. dist. from x_0 to the hyperplane $= \frac{|g(x_0)|}{\|w\|}$

by minimizing $\|x - x_0\|^2$ subject to $g(x) = 0$

As this is a case of constrained optimization, we can use lagrangian optimization with $g(x) = 0$ constraint.

$f(x) = \|x - x_0\|^2$, $g(x) = w^T x + b = 0$, $L(x) = f(x) - \lambda g(x)$

$L(x) = \|x - x_0\|^2 - \lambda (w^T x + b)$. We should check the 1st and 2nd order derivatives.

$$\frac{\partial L(x)}{\partial x} = 2(x-x_a) - \lambda w = 0, \quad \lambda w = 2(x-x_a)$$

$$\lambda w^T w = w^T 2(x-x_a) = \lambda \|w\|^2 = w^T 2(x-x_a)$$

$$\lambda = \frac{w^T 2(x-x_a)}{\|w\|^2} = \frac{2}{\|w\|^2} w^T(x-x_a)$$

Detour: $w^T(x-x_a) = w^T x - w^T x_a = w^T x + b - w^T x_a - b = g(x) - g(x_a)$

Continuing: $\lambda = \frac{2}{\|w\|^2}(g(x) - g(x_a))$, but we know $g(x) = 0$

$\lambda = \frac{2}{\|w\|^2}(-g(x_a))$, we have to isolate $\|x-x_a\|$ from this equation to find the minimum distance.

We know: $\lambda = \frac{2}{\|w\|^2}(-g(x_a))$, $\frac{\lambda}{2} = \frac{(-g(x_a))}{\|w\|^2}$, $\frac{\lambda}{2} = \frac{w^T(x-x_a)}{\|w\|^2}$ (From before)

$$\frac{\lambda}{2} w = \frac{w^T w (x-x_a)}{\|w\|^2} = \frac{\|w\|^2 (x-x_a)}{\|w\|^2} = \frac{(-g(x_a))}{\|w\|^2} w = \frac{\lambda}{2} w$$

$\|x-x_a\| = \frac{|g(x_a)|}{\|w\|}$, We also have to check the Hessian $\frac{\partial L}{\partial x^2}$ to be Positive Definite

$\frac{\partial L}{\partial x^2} = 2 \times I$ where I is the Identity matrix and P.D.

b) projection $x_p = x_a - \frac{g(x_a)}{\|w\|^2} w$

We have seen $x - x_a = \frac{(-g(x_a))}{\|w\|^2} w$ for x s.t. x is the closest point on $g(x)$ to x_a. Therefore x is x_p, the projection of x_a on x_p.

So, we have $x_p - x_a = -\frac{\rho(x_a)}{\|w\|^2} w$, $\quad x_p = x_a - \frac{\rho(x_a)}{\|w\|^2} w$

Problem 3

To set $D = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, $a = (w, b)$ of the function $f(x) = w^T x + b$, minimize $L(w,b) = \sum_{i=1}^{n} (y_i - (w^T x_i + b))^2$

a) The optimal solution of Linear Regression is:

$$a = (X^T X)^{-1} X^T y.$$

As always, we should check the 1st & 2nd derivatives to find the minimum point

$$\frac{\partial L(w,b)}{\partial w} = -2 \sum_{i=1}^{n} (y_i - (w^T x_i + b)) x_i = 0$$

$$\frac{\partial L(w,b)}{\partial b} = -2 \sum_{i=1}^{n} (y_i - (w^T x_i + b)) = 0$$

$$\sum_{i=1}^{n} y_i x_i = b \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} (w^T x_i) x_i$$

$$= \sum_{i=1}^{n} y_i x_i = b \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} (x_i x_i^T) w$$

$$\sum_{i=1}^{n} y_i = bn + \sum_{i=1}^{n} x_i^T w$$

We should satisfy these equations

- $w^T x_i$ is a scalar
- $w^T x_i = x_i^T w$
- $(x_i^T w) x_i = x_i (x_i^T w) = (x_i x_i^T) w$

if $\tilde{X} = \begin{bmatrix} — & x_1 & — \\ & \vdots & \\ — & x_n & — \end{bmatrix}$, $\sum_{i=1}^{n} y_i x_i = \tilde{X}^T \tilde{X} w + b \sum_{i=1}^{n} x_i$

we have $a = \begin{bmatrix} w \\ b \end{bmatrix}$, if we use homogeneous coordinates, we can integrate b into X.

$$X = \begin{bmatrix} \leftarrow x_1 \rightarrow & 1 \\ & 1 \\ \leftarrow x_n \rightarrow & 1 \end{bmatrix}_{n \times (d+1)}, \quad X^T = \begin{bmatrix} \uparrow & & \uparrow \\ x_1 & \cdots & x_n \\ \downarrow & & \downarrow \\ 1 & & 1 \end{bmatrix}_{(d+1) \times n}$$

$$a = \begin{bmatrix} w \\ b \end{bmatrix}_{(d+1) \times 1} = \begin{bmatrix} \leftarrow w \rightarrow b \end{bmatrix}^T_{(d+1) \times 1}$$

and for $y = [y_1 \cdots y_n]^T_{(n \times 1)}$

The previous equations can be represented as

For the left parts : $X^T X a$

For the right parts : $X^T y$

Finally $a = (X^T X)\, a = X^T y$

$a = (X^T X)^{-1} X^T y$, However we have to check the Hessian matrix is P.D. to be sure of the result.

b) In S.G.D. we compute the gradient for a single point, then take a small step in the opposite direction, The step size is η.

From part a) we have the following

$$\frac{\partial L}{\partial w} \text{ for } x_n = -2(y_i - x_n^T w - b)x_n$$

$$\frac{\partial L}{\partial b} \text{ for } x_n = -2(y_i - x_n^T w - b)$$

So, our updates for w and b should be the negative of the above values.

Modifying the Perceptron algorithm from the lecture slides, we can build the SGD algorithm.

$t=0$, $w_1=0$, $\eta=\eta_s$, $b_1=0$

while (Loss(w^t, b^t) is $\geq$ Loss-threshold)
{
  for $i = 1:n$ {
    $t = t+1$;
    $w_{t+1} = w_t + \eta(y_i - x_i^T w_t - b)x_i$
    $b_{t+1} = b_t + \eta(y_i - x_i^T w_t - b)$
  }
}

c) The first step is to set the derivatives to 0

$$\frac{\partial L(w,b)}{\partial w} = -2\sum_{i=1}^{n}(y_i - x_i^T w - b)x_i + 2\lambda w = 0$$

$$\frac{\partial L(w,b)}{\partial b} = -2\sum_{i=1}^{n}(y_i - x_i^T w - b) = 0$$

$$w = \frac{1}{\lambda}\sum_{i=1}^{n}(y_i - x_i^T w - b)x_i$$

$$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i^T w + n.b \;,\quad \sum_{i=1}^{n} x_i^T w = \sum_{i=1}^{n} y_i - n.b$$

If we set $a_i = (y_i - x_i^T w - b)/\lambda$

Then $w = \sum_{i=1}^{n} a_i x_i$, as required

Now we should integrate this result into the Loss function $L(w,b)$

$$L(w,b) = \sum_{i=1}^{n}(y_i - x_i^T w - b)^2 + \lambda w^T w$$

$$= \sum_{i=1}^{n}\left(y_i - x_i^T \sum_{j=1}^{n} a_j x_j - b\right)^2 + \lambda \sum_{i=1}^{n} a_i x_i^T \sum_{j=1}^{n} a_j x_j$$

$$= \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{n} a_j x_j^T x_i - b\right)^2 + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j x_i^T x_j$$

For the dot products $i,j \Rightarrow x_i^T x_j$, we get a scaler for each i,j pair. These can be represented as a matrix of dot products D where

$$D_{ij} = x_i^T x_j$$

To turn expressions such as $\sum_{j=1}^{n} a_j x_j^T x_i$ and $\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j x_i^T x_j$ into vector/matrix form including D, I can vectorize a, $a = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$

Then, $\sum_{j=1}^{n} a_j x_j^T x_i = \sum_{j=1}^{n} x_j^T x_i a_j$ = ith value of Da

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j x_i^T x_j = \begin{bmatrix} \rightarrow a \rightarrow \end{bmatrix} \begin{bmatrix} D \end{bmatrix} \begin{bmatrix} \uparrow \\ a \\ \downarrow \end{bmatrix}$$

$$= a^T D a$$

The following expression can be modified as $\sum_{i}^{n} (y_i - Da(i) - b)^2 = \sum_{i=1}^{n} (y_i - d_i)^2$,

if we consider $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$ and $d = \begin{bmatrix} d_1 - b \\ \vdots \\ d_n - b \end{bmatrix}$, this is the square of a distance function $\| y_i - d_i \|^2$

So we have $\sum_{i=1}^{n} (y_i - Da(i) - b)^2 = \| y - Da - b \|^2$

The overall expression is:

$$L = \| y - Da - b \|^2 + \lambda a^T D a$$

The values to be optimized in this equation are the a_i's, therefore $a = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$ should be optimized

We should take the derivative w.r.t $\underline{a}$ and set it to zero.

$$\frac{\partial L}{\partial a} = -2D(y - Da - b) + 2\lambda Da = 0$$

$$\lambda Da = D(y - Da - b)$$

$$\lambda Da + DDa = Dy - bD$$

$$D(\underbrace{\lambda + D}_{\lambda I})a = D(y - b)$$

$$(\lambda I + D)a = (y - b)$$

$$a = (\lambda I + D)^{-1}(y - b)$$

as $w^* = \sum_i a_i x_i = X^T a$ where $X = \begin{bmatrix} \leftarrow x_1 \rightarrow \\ \vdots \\ \leftarrow x_n \rightarrow \end{bmatrix}$

$y = f(x) = w^{*T} x$ in our solution

therefore $y = a^T X x = (y - b)^T (\lambda I + D)^{-1} X x$

here D is made up of $x_i^T x_j$ dot products.

a)

Problem 4) For the Perceptron algorithm the learning rate η is irrelevant. This is because we are interested in the sign of of the $w^T x + b$.

if we start with $w=0$ and $b=0$, for $x \in J$,

$J \Rightarrow$ the points that are misclassified

$$w_{t+1} = \eta \sum_{i \in J} y_i x_i$$

$$b_{t+1} = \eta \sum_{i \in J} y_i R^2$$

} η scales these values. however as $\eta > 0$ it doesn't change their sign

During the loop we have if $y_i (w_t^T x_i + b_t)$, this does not depend on the size of the expression, but on the sign. The decision function also doesn't care about the sizes, but just the signs.

b) $y_i \left(\sum_{j=1}^{\hat{}} a_j y_j x_j^T x_i + b \right) \leq 0$? (wikipedia kernel perceptron page)

This is the dual form of the Perceptron algorithm. It is a weighted sum where the weights are the #mistakes for each point. The a_j values can be thought as 'mistake counters'

Previously we had $w_t^T x_i + b_t$ & $w_{t+1} = \eta \sum_{i \in J} y_i x_i$

Now we have $w_{new}^T x_i + b^T$ & $w_{new} = \eta \sum_{i=1}^{\hat{}} a_i y_i x_i^T$

then, the expression in the if statement becomes:

$$y_i (w_{new}^T x_j + b) \Rightarrow y_i \left(\eta \sum_{i=1}^{\hat{}} a_i y_i x_i^T x_j + b \right)$$

c) We update a_i's each time a point causes a mistake. The points that are frequently misclassified have high a_i counts. Our algorithm has trouble classifying these points. They may be stated as points that are 'harder to classify.'

d) Using the dual form of the Perceptron is the first step for Kernelizing the algorithm. (From wikipedia, Kernel perceptron page)

For b's, $b = b + y_i R^2$, if $b_1 = 0$

$b_i = a_i y_i R^2$ for each x_i, n total

$$b = \sum_{i=1}^{n} a_i y_i R^2, \text{ we already have } y_i \left(\sum_{i=1}^{n} a_j y_j x_j^T x_i + b \right)$$

$$\text{So, } h(x) = \text{sgn} \left[\left(\sum_{i=1}^{n} a_j y_j (x_j^T x_i) + \sum_{i=1}^{n} a_i y_i R^2 \right) \right]$$

In kernel perceptron, the dot product can be replaced by a kernel function of the form $K(x_i, x_j)$, creating a new mapping. The final result will be:

$$h(x) = \text{sgn} \left[\left(\sum_{i=1}^{n} a_j y_j K(x_i, x_j) + \sum_{i=1}^{n} a_i y_i R^2 \right) \right]$$

Problem 5)

a) i) $$-\frac{\partial E^{(n)}(w)}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n$$

$$E(w)^{(n)} = -\sum_{k=1}^{c} t_k^n \ln y_k^n \quad \text{where } y_k^n = \frac{e^{a_k^n}}{\sum_j e^{a_j^n}} = \text{softmax}$$

From lecture slides 8, page 20:

$$\frac{\partial E}{\partial w_{jk}} = -\sum_{k=1}^{c} \frac{\partial}{\partial w_{jk}} t_k^n \ln y_k^n \,, \quad \frac{\partial E}{\partial w_{jk}} = \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{jk}}$$

$$= -\sum_{l=1}^{c} \frac{t_k^n}{y_l^n} \cdot \frac{\partial y_l^n}{\partial a_k} \cdot x_j^n \,, \quad \left(a_k = \sum_i w_{ji} x_i\right)$$

$\frac{\partial y_l^n}{\partial a_k}$ is the derivative of the softmax function

$$= \frac{\partial \frac{e^{a_k^n}}{\sum_{j=1}^{N} e^{a_j^n}}}{\partial a_k} \Longrightarrow f'(x) = \frac{g'h - h'g}{h^2}$$

where $f(x) = \frac{g(x)}{h(x)}$

This derivative is different for $l = k$ and $l \neq k$ cases

$l = k \rightarrow y_k^n (1 - y_k^n)$

$l \neq k \rightarrow -y_k^n y_l^n$

Therefore, $\frac{\partial E}{\partial w_{jk}} = \left(\frac{t_k^n}{y_k^n} \cdot y_k^n (1 - y_k^n) + \frac{1}{y_l^n} \sum_{l \neq k} t_l^n (-y_l^n y_k^n) \right) x_j^n$

$$= \left(t_k^n (1 - y_k^n) - \sum_{l \neq k} t_l^n y_k^n \right) x_j^n$$

$$= \left(t_k^n - t_k^n y_k^n - y_k^n \sum_{l \neq k} t_l^n \right) x_j^n$$

$$= \left(t_k^n - y_k^n \sum_{l} t_l^n \right) x_j^n$$

$\sum_l t_l^n = 1$ because we are using one hot encoding for $t = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

Therefore finally we get $\frac{-\partial E^n(w)}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n$

b) i) For derivation of S.G.D, I will use the notation of the lecture slides 8, pages 20–24.

There are two derivatives to be calculated, for the ① hidden and ② output layers. The below formulas will be for a single point x_n, however I will not write the n's explicitly as they make expressions cumbersome.

We have to calculate: $\frac{\partial E}{\partial w_{ji}}$ for hidden ①, $\frac{\partial E}{\partial w_{jk}}$ for output ②

$$\frac{\partial E}{\partial w_{jk}} = \sum_k \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial u_k} \frac{\partial u_k}{\partial w_{jk}}$$

$$u_k = \sum_j w_{jk} y_j, \quad \frac{\partial u_k}{\partial w_{kj}} = y_j, \quad \boxed{z_k = \text{softmax}(u_k)}$$

$$E = \sum_i t_i \ln(\text{softmax}(u_i)) = \sum_i t_i \ln\left(\frac{\exp(u_i)}{\sum_j \exp(u_j)}\right)$$

$\frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial u_k} \left[\sum_i t_i \ln\left(\frac{\exp(u_i)}{\sum_j \exp(u_j)}\right)\right]$ has two outcomes, as before calling softmax as z_i:

$\rightarrow$ $i = j$, $\frac{\partial}{\partial u_k} \rightarrow \frac{t_i}{z_i} z_i(1 - z_i)$

$j \neq i$, $\frac{\partial}{\partial u_k} \rightarrow -\sum_{j \neq i} t_j (-z_i z_j) \cdot \frac{1}{z_j}$

summing the two parts: $t_i - t_i z_j - \sum_{j \neq i} t_j z_j$

$= t_i - \sum_j t_j z_j = (t_i - z_i) \Rightarrow (t_k - z_k)$

Therefore $\frac{\partial E}{\partial w_{kj}} = -\sum_k (t_k^{(n)} - z_k^{(n)})\, y_j$

For $\frac{\partial E}{\partial w_{ji}}$, the hidden layer we have:

$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}}$, $s_j = \sum_i w_{ji} x_i$

$y_j = \sigma[s_j]$

$\frac{\partial y_j}{\partial s_j} = \sigma'[s_j] = \sigma[s_j](1 - \sigma[s_j])$

$\frac{\partial s_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} x_i = x_i$

$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial u_k} \frac{\partial u_k}{\partial y_j} \frac{\partial y_j}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}}$

$= -\sum_k (t_k - z_k)\, w_{kj}\, \sigma'[s_j]\, x_i$

$\frac{-\partial E}{\partial w_{ji}} = -\sigma[s_j^{(n)}](1 - \sigma[s_j^{(n)}]) \sum_k (t_k^{(n)} - z_k^{(n)})\, w_{kj}^{(n)}\, x_i^{(n)}$

Forward step is:

$s_j^{(n)} = \sum_i w_{ji} x_i^{(n)}$, $y_j^{(n)} = \sigma[s_j^{(n)}]$, u_k

$u_k^{(n)} = \sum_j w_{kj} y_j^{(n)}$, $z_k = \frac{\exp(u_k)}{\sum_j \exp(u_j)}$

Arda Cankat Bati

ECE 271B Homework #2, Computer Assignment

Std Id = A53284500

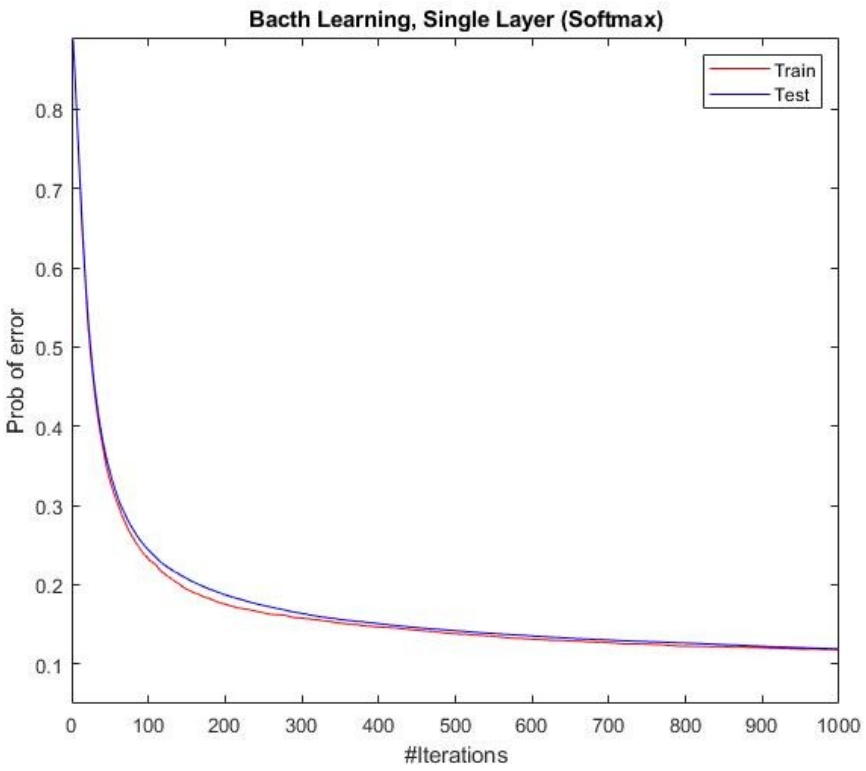
Question 5 Part A)

i) Derivation of the gradient is done in the handwritten part of the HW report.

ii) I implemented batch learning using the gradient found in part i. I calculated the prob of error in each iteration, and I ran 1000 iterations. After 1000 iterations the results seem to start converge. The results of train and test set are very close. The final error rates after 1000 iterations were:

Batch Learning, Single Layer(Softmax): Train err: 0.117700, Test err: 0.119050.

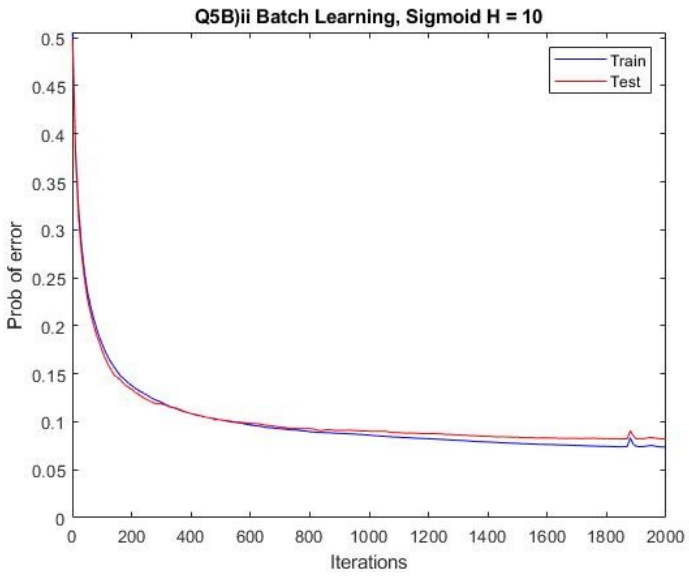
For further iterations, my expectation is that the gap between the train & test error will increase. After too many iterations, there will be overfitting, which I observed in the following sections.

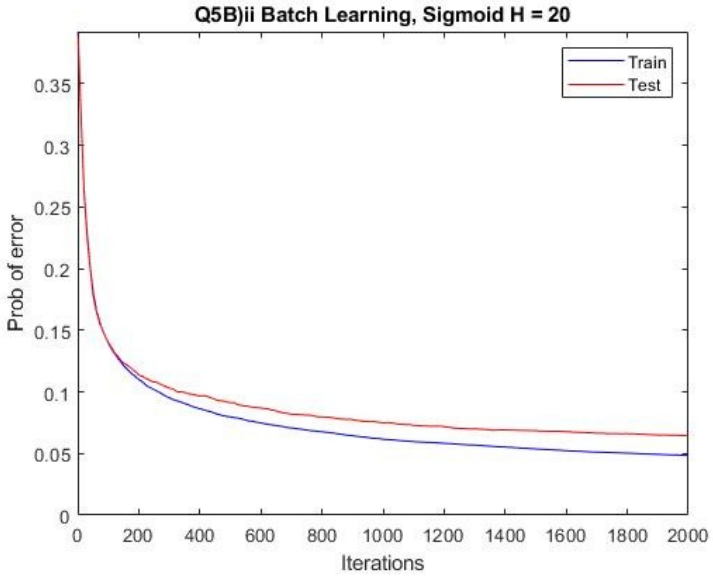


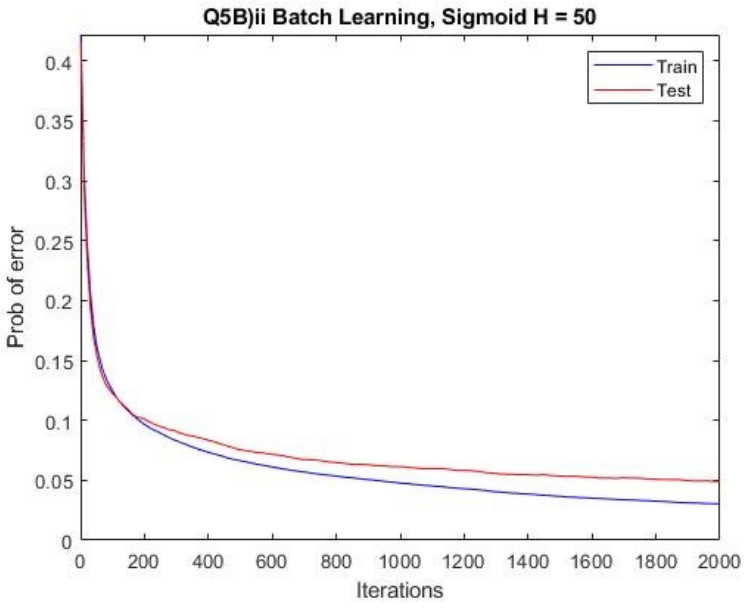
Question 5 Part B)

i) Derivation of the S.G.D. for two layers is done in the handwritten part of the HW report.

ii) The derivation of Batch Learning backpropagation formulas mostly followed directly from the lecture slides. The differences were in the Loss function and the non linearities. For this section, I ran 2000 iterations for each of the 3 networks. However I calculated the prob of error after every 10 iterations. Therefore in total 2000/10 = 200 separate points were used to build the below graphs.







The final error rates were as follows:

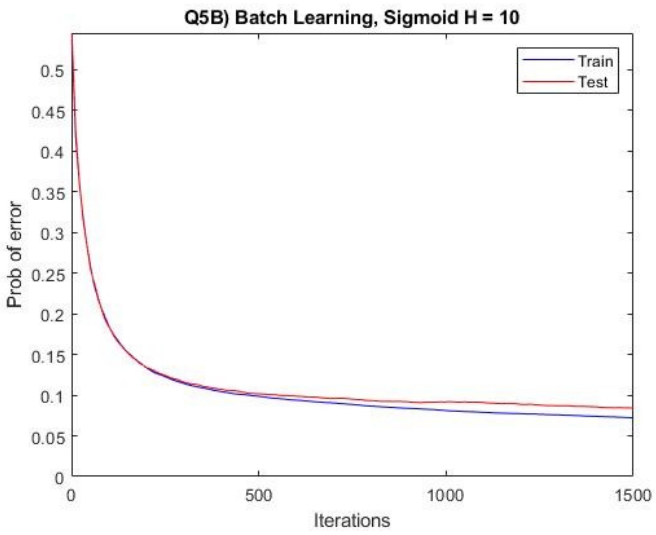
- H = 10: Train err: 0.073817, Test err: 0.082500
- H = 20: Train err: 0.048367, Test err: 0.064500
- H = 50: Train err: 0.030317, Test err: 0.049000'

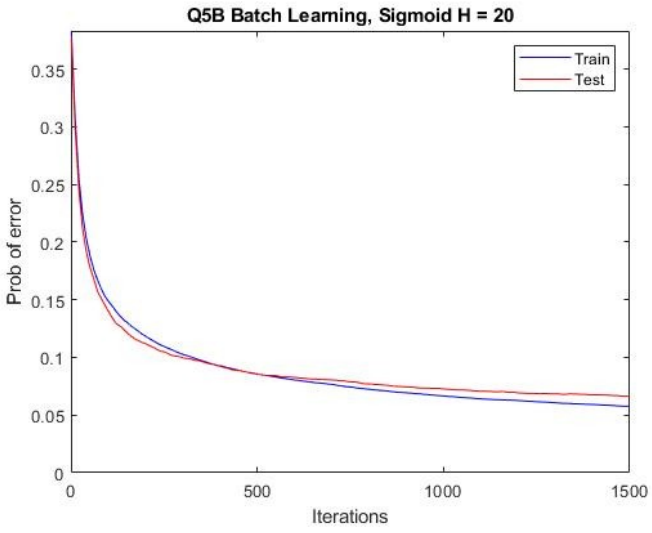
From the graphs and error rates, it is clear that H = 50 case has the best performance both in final accuracy and convergence rate. After a certain number of iterations, we start to see the beginnings of overfitting, especially for H = 20& H = 50. After the difference between the Train and Test error rates reaches a certain size, it is wise to stop further iterations.

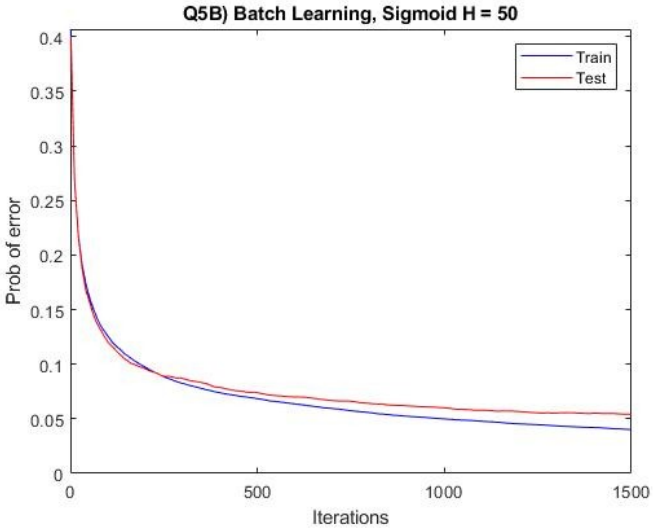
iii)

For this part my RELU implementation was probably not correct and it didn't yield comparable values to the sigmoid case. Therefore, I will only include the graphs for the sigmoid cases with weight decay. From my general knowledge of RELU, it should be computationally faster than the sigmoids. I would expect the final error rates to be somewhat similar for both cases (for same # epochs). I ran 1500 iterations and sampled the error in for every 10 iterations. My results from the sigmoid case are as follows:

- H = 10: Train err: 0.072317, Test err: 0.084100
- H = 20: Train err: 0.057617, Test err: 0.066200
- H = 50: Train err: 0.040100, Test err: 0.054300





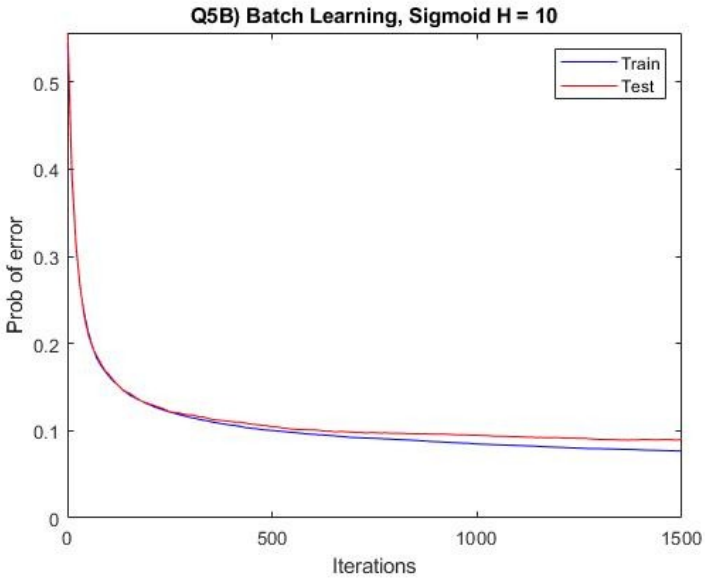


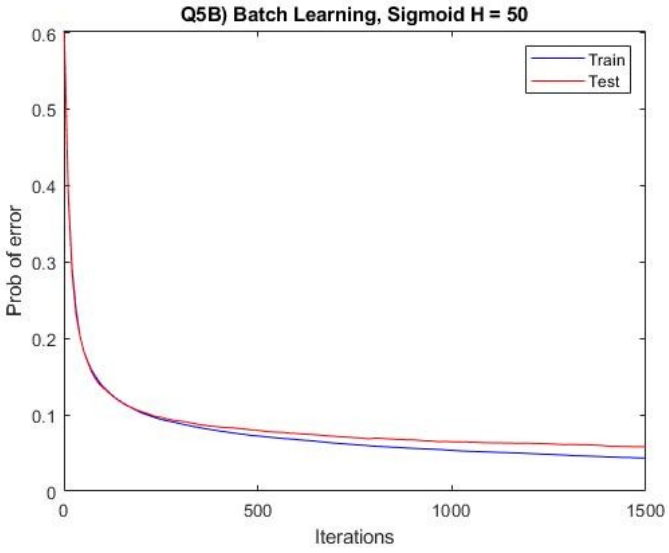
I can’t make the comparisons with the RELU networks, However from among the Sigmoid ones, H = 50 network had the best performance, as expected.

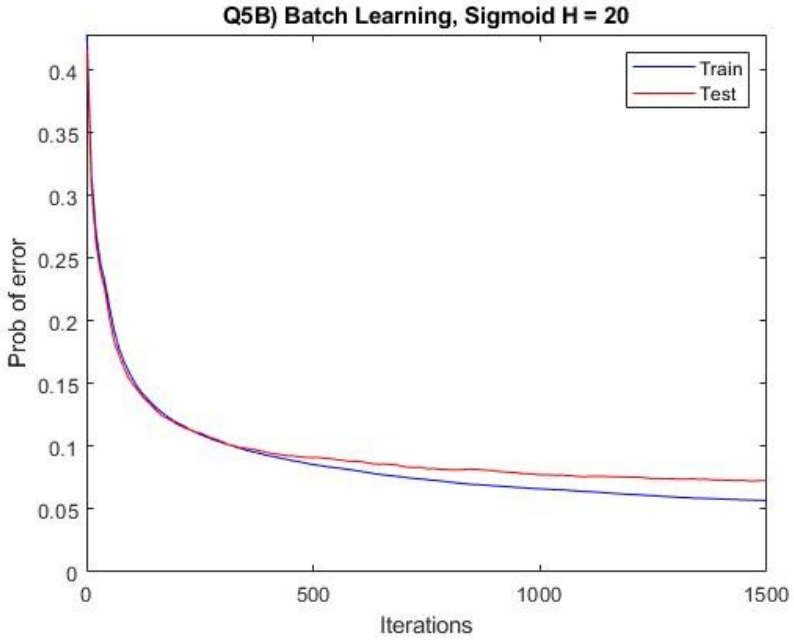
iv)

I repeated part iii again without RELU networks only changing the gamma value. The results were somewhat worse, compared to the previous part. This may show that for wrong choices of weight decay constants, the network may become worse.

- H = 10: Train err: 0.076750, Test err: 0.089800
- H = 20: Train err: 0.056933, Test err: 0.072700
- H = 50: Train err: 0.043383, Test err: 0.058300



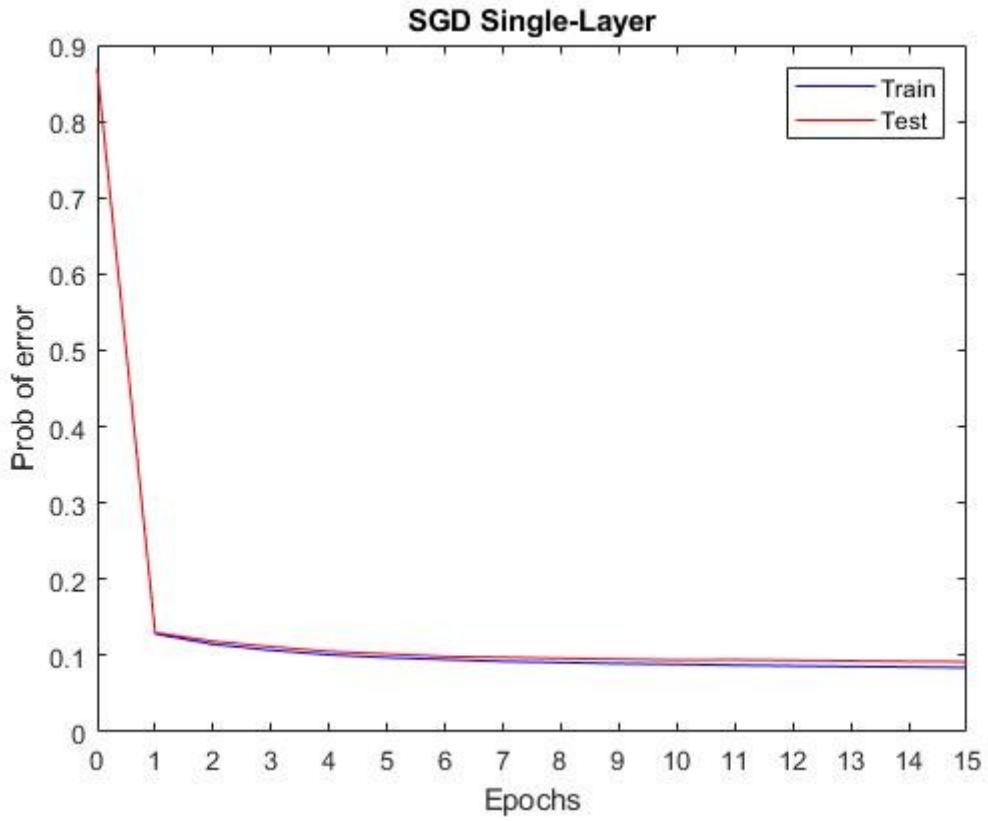




Q5C)

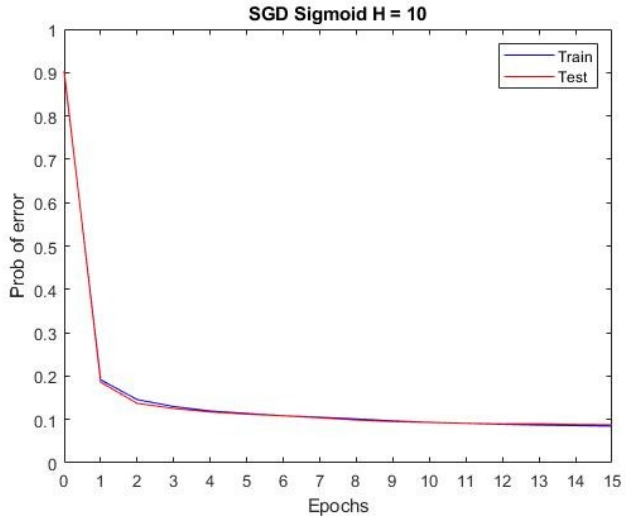
a) For the single layer SGD network I ran the SGD for 15 Epochs. I calculated the prob. of error for each epoch. The computation speed and convergence rate in the SGD case was very fast compared to the batch learning. It was faster than running 1500 iterations on the batch learning case, and it produced better results. Even 1 Epoch, which is computed quite fast, it reached a prob of error of about 0.13.

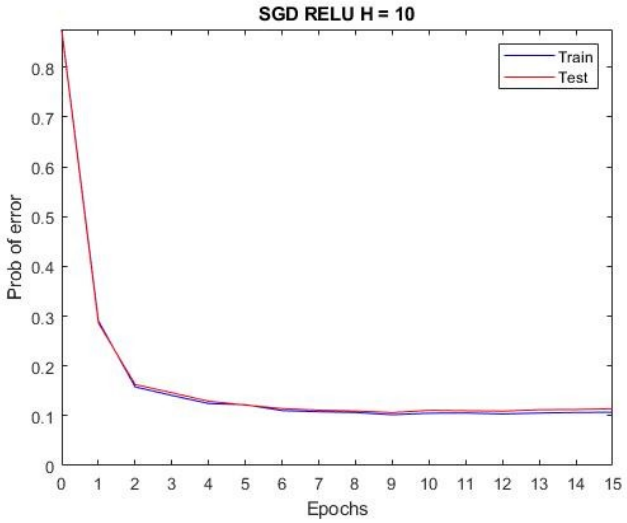
The error rate was after 15 Epochs: Train err: 0.084167, Test err: 0.091900.

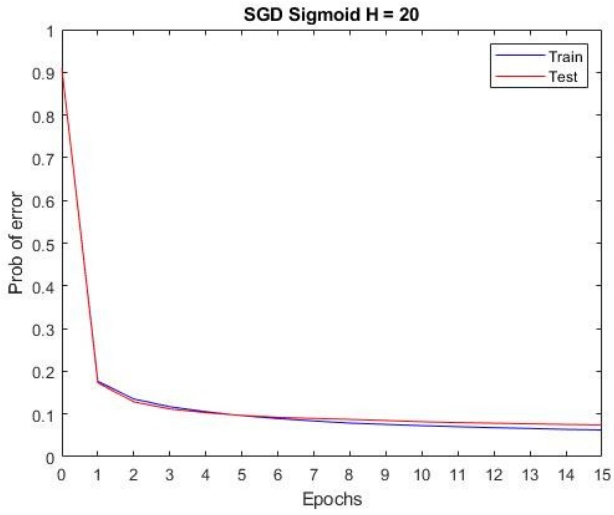


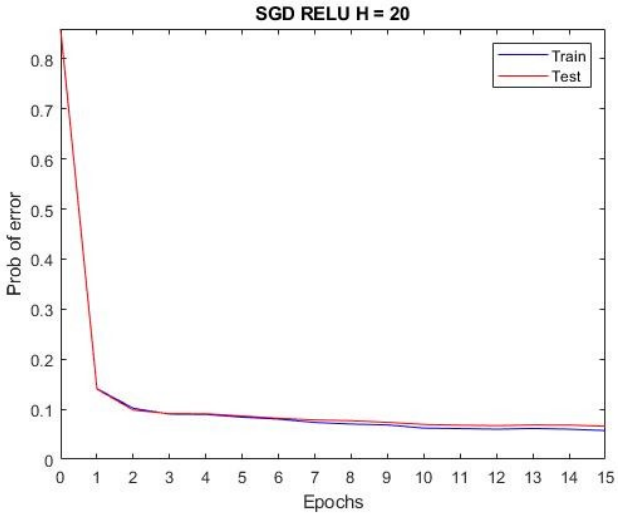
b) For the S.G.D. case my RELU implementation seemed to work correctly. Therefore I will be providing all the 6 plots for H=10,20,50 for the Sigmoid and RELU cases. Similar to the previous part, I ran the SGD for 15 Epochs and recorded the error in each epoch. The final error rates were as follows (after 15 Epochs):

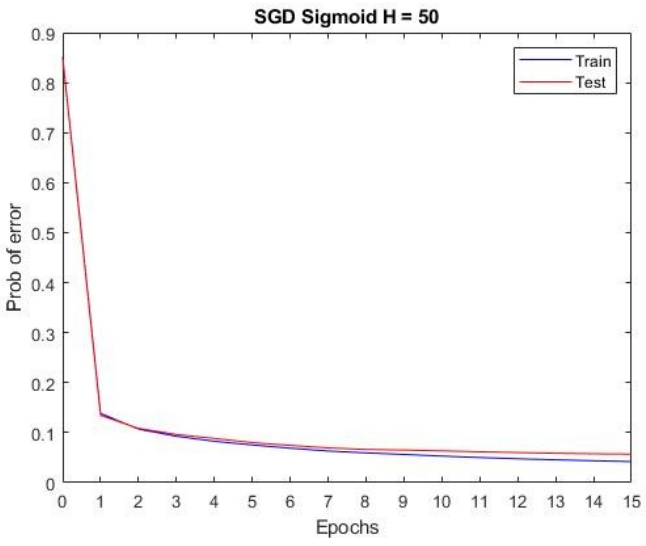
- Sigmoid, H = 10: Train err: 0.076983, Test err: 0.080900
- Sigmoid, H = 20: Train err: 0.057117, Test err: 0.066000
- Sigmoid, H = 50: Train err: 0.042983, Test err: 0.058000
- RELU, H = 10: Train err: 0.107033, Test err: 0.114100
- RELU, H = 20: Train err: 0.057350, Test err: 0.067900
- RELU, H = 50: Train err: 0.044350, Test err: 0.059900

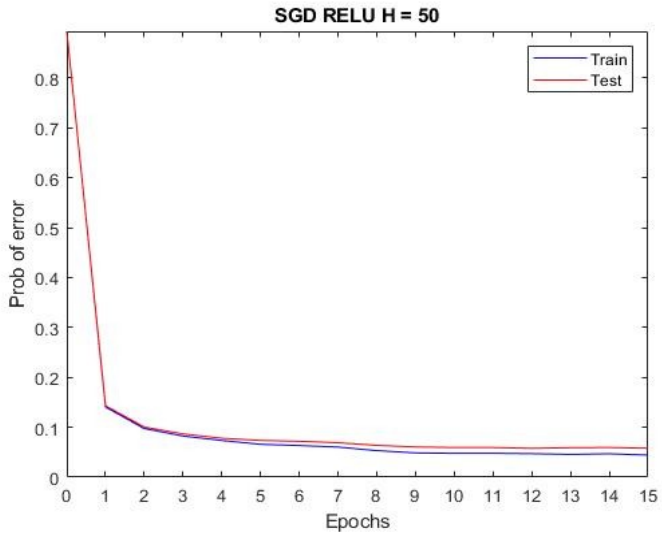












For S.G.D. the sigmoid network error rates were slightly better than the RELU networks. The difference seems to become less noticeable as the number of hidden units increases. I noticed the fast computation advantage of RELU, however as SGD is already quite fast, it did not matter too much.

```
clc;
clear all;
trainImg = 'train-images.idx3-ubyte';
trainLabel = 'train-labels.idx1-ubyte';

testImg = 't10k-images.idx3-ubyte';
testLabel = 't10k-labels.idx1-ubyte';

[train_imgs, train_labels] = readMNIST(trainImg, trainLabel, 60000,
 0);
[test_imgs, test_labels] = readMNIST(testImg, testLabel, 10000, 0);

save HW2_Data
load('HW2_Data.mat')

mu = 0; sigma = 1; size_letters = 784;
k_count = 10; %class count / output units count

train_images_squared = train_imgs.*train_imgs;

train_bias = ones(60000,1);
train_imgs = [train_imgs train_bias];

test_bias = ones(10000,1);
test_imgs = [test_imgs test_bias];

train_imgs_T = train_imgs';
test_imgs_T = test_imgs';

nu = 10^(-5);

train_labels_one_hot = zeros(10,60000);
for i = 1:60000
    train_labels_one_hot(train_labels(i) + 1,i) = 1;
end

% Homogenous coordinates, b should be added
weights = normrnd(mu, sigma, [k_count size_letters]);
bias = ones(10, 1);
w_all = [weights bias]; % Homogenous weights, input bias term added
loop_count = 1000;
loop_w_all = zeros(10,785,loop_count);
loop_w_all(:,:,1) = w_all;

for loop = 1:loop_count
    %Forward step
    cur_w = loop_w_all(:,:,loop);
    actvs = cur_w * train_imgs_T;
    % Softmax calculation
    y_values = exp(actvs) ./ sum(exp(actvs),1);
    % Descent step calculation
    Grad_E = (train_labels_one_hot - y_values)*(train_imgs);
```

```
    % Updating weights
    loop_w_all(:,:,loop + 1) = cur_w + nu * Grad_E;
end

errors_train = zeros(1,loop_count);
errors_test = zeros(1,loop_count);

for loop = 1:loop_count
    %Forward step
    cur_w = loop_w_all(:,:,loop);
    err1 = calc_err(cur_w,train_imgs,train_labels);
    err2 = calc_err(cur_w,test_imgs,test_labels);
    errors_train(loop) = err1;
    errors_test(loop) = err2;
end

figure()
loop = [1:loop_count];
plot(loop, errors_test, 'r');
hold on
plot(loop, errors_train, 'b');
ylim([0.05, inf]);
title('Bacth Learning, Single Layer (Softmax)');
xlabel('#Iterations');
ylabel('Prob of error')
legend('Train','Test')
sprintf('Batch Learning, Single Layer(Softmax), Train err: %f, Test
 err: %f', errors_train(1,loop_count),errors_test(1,loop_count))

function err = calc_err(W1,data,labels)
    actvs = W1 * data';
    % Softmax results
    y_values = exp(actvs) ./ sum(exp(actvs),1);
    [values, indices] = max(y_values);
    indices = indices - 1;
    % Error calculation
    err = sum(sum((labels ~= indices'))) / size(data,1);
end
```

Published with MATLAB® R2018b

```
clc;
clear all;

load('HW2_Data.mat')

mu = 0; sigma = 1; size_letters = 784;
k_count = 10; %class count / output units count

train_images_squared = train_imgs.*train_imgs;
train_bias = ones(60000,1);
train_imgs = [train_imgs train_bias];

test_bias = ones(10000,1);
test_imgs = [test_imgs test_bias];

train_imgs_T = train_imgs';
test_imgs_T = test_imgs';

nu = 10^(-4);

train_labels_one_hot = zeros(10,60000);
for i = 1:60000
    train_labels_one_hot(train_labels(i) + 1,i) = 1;
end


H_Values = [10, 20, 50];
for H_Layer = H_Values

    % Homogenous coordinates, b should be added
    bias_input = ones(H_Layer, 1);

    hidden_weights = normrnd(mu, sigma, [H_Layer size_letters]);
    W1 = [hidden_weights bias_input]; % Homogenous weights, input bias
 term added
    W2 = normrnd(mu, sigma, [k_count H_Layer]);

    % Backpropogation loops initialization
    checkpoints = 200;
    loop_count = 2000;
    loop_w_hidden = zeros(size(W1,1), size(W1,2), checkpoints);
    loop_w_output = zeros(size(W2,1), size(W2,2), checkpoints);
    loop_w_hidden(:,:,1) = W1;
    loop_w_output(:,:,1) = W2;

    count = 0;
    % Backpropogation loop
    for loop = 1:loop_count
        % **** FORWARD STEP ****
        g = W1 * train_imgs';
        y = 1 ./ (1 + exp(-g));
        u = W2 * y;
```

```
        z = exp(u) ./ sum(exp(u),1); %Softmax

        Grad2 = (train_labels_one_hot - z)*y';

    %   % The longer, loop solution
    %     Grad1 = zeros(15,785);
    %     for n = 1:60000
    %         %Grad1_1 = (train_labels_one_hot(:,n) - z(:,n))'*W2;
    %         %Grad1_2 = (y(:,n) .* (1 - y(:,n)))  * train_imgs(n,:);
    %         %Grad1 = Grad1 + diag((train_labels_one_hot(:,n) -
 z(:,n))'*W2) * (y(:,n) .* (1 - y(:,n)))  * train_imgs(n,:);
    %     end

        % Faster solution, same operation as above
        Grad1 = ((((train_labels_one_hot - z)'*W2)').*(y .* (1 -
 y)))*train_imgs;
        W1 = W1 + nu * Grad1;
        W2 = W2 + nu * Grad2;

        if rem(loop,loop_count/checkpoints) == 0
            count = count + 1;
            loop_w_hidden(:,:,count) = W1;
            loop_w_output(:,:,count) = W2;
        end
    end

    errors_train = zeros(1,checkpoints);
    errors_test = zeros(1,checkpoints);
    for loop = 1:checkpoints
        % **** FORWARD STEP ****
        W1 = loop_w_hidden(:,:,loop);
        W2 = loop_w_output(:,:,loop);
        err1 = calc_err(W1,W2,train_imgs,train_labels);
        err2 = calc_err(W1,W2,test_imgs,test_labels);
        errors_train(1,loop) = err1;
        errors_test(1,loop) = err2;
    end

    figure()
    plot(linspace(1,loop_count,checkpoints), errors_train, 'b')
    hold on
    plot(linspace(1,loop_count,checkpoints), errors_test, 'r')
    hold off
    xlabel('Iterations');
    ylim([0 inf]);
    %xticks(0:Epochs)
    ylabel('Prob of error')
    legend('Train','Test')
    title(sprintf('Q5B)ii Batch Learning, Sigmoid H = %d', H_Layer))
    sprintf('Q5B)ii Batch Learning, Sigmoid, H
 = %d: Train err: %f, Test err: %f', H_Layer,
 errors_train(1,checkpoints),errors_test(1,checkpoints))

end
```

```
function err = calc_err(W1,W2,data,labels)
    g = W1 * data';
    y = 1 ./ (1 + exp(-g));
    u = W2 * y;
    z = exp(u) ./ sum(exp(u),1); %Softmax

    [values, indices] = max(z);
    indices = indices - 1;
    err = sum(sum((labels ~= indices'))) / size(data,1);
end
```

Published with MATLAB® R2018b

```
clc;
clear all;

load('HW2_Data.mat')

mu = 0; sigma = 1; size_letters = 784;
k_count = 10; %class count / output units count

train_images_squared = train_imgs.*train_imgs;
train_bias = ones(60000,1);
train_imgs = [train_imgs train_bias];

test_bias = ones(10000,1);
test_imgs = [test_imgs test_bias];

train_imgs_T = train_imgs';
test_imgs_T = test_imgs';

nu = 10^(-4);
gam = 3*10^(-2);

train_labels_one_hot = zeros(10,60000);
for i = 1:60000
    train_labels_one_hot(train_labels(i) + 1,i) = 1;
end


H_Values = [10, 20, 50];
for H_Layer = H_Values

    % Homogenous coordinates, b should be added
    bias_input = ones(H_Layer, 1);

    hidden_weights = normrnd(mu, sigma, [H_Layer size_letters]);
    W1 = [hidden_weights bias_input]; % Homogenous weights, input bias
 term added
    W2 = normrnd(mu, sigma, [k_count H_Layer]);

    % Backpropogation loops initialization
    checkpoints = 150;
    loop_count = 1500;
    loop_w_hidden = zeros(size(W1,1), size(W1,2), checkpoints);
    loop_w_output = zeros(size(W2,1), size(W2,2), checkpoints);
    loop_w_hidden(:,:,1) = W1;
    loop_w_output(:,:,1) = W2;

    count = 0;
    % Backpropogation loop
    for loop = 1:loop_count
        % **** FORWARD STEP ****
        g = W1 * train_imgs';
        y = 1 ./ (1 + exp(-g));
```

```
        u = W2 * y;
        z = exp(u) ./ sum(exp(u),1); %Softmax

        Grad2 = (train_labels_one_hot - z)*y';

    %   % The longer, loop solution
    %     Grad1 = zeros(15,785);
    %     for n = 1:60000
    %         %Grad1_1 = (train_labels_one_hot(:,n) - z(:,n))'*W2;
    %         %Grad1_2 = (y(:,n) .* (1 - y(:,n)))  * train_imgs(n,:);
    %         %Grad1 = Grad1 + diag((train_labels_one_hot(:,n) -
z(:,n))'*W2) * (y(:,n) .* (1 - y(:,n)))  * train_imgs(n,:);
    %     end

        % Faster solution, same operation as above
        Grad1 = ((((train_labels_one_hot - z)'*W2)').*(y .* (1 -
y)))*train_imgs;
        W1 = W1 + nu * Grad1 - nu*gam*W1;
        W2 = W2 + nu * Grad2 - nu*gam*W2;

        if rem(loop,loop_count/checkpoints) == 0
            count = count + 1;
            loop_w_hidden(:,:,count) = W1;
            loop_w_output(:,:,count) = W2;
        end
    end

    errors_train = zeros(1,checkpoints);
    errors_test = zeros(1,checkpoints);
    for loop = 1:checkpoints
        % **** FORWARD STEP ****
        W1 = loop_w_hidden(:,:,loop);
        W2 = loop_w_output(:,:,loop);
        err1 = calc_err(W1,W2,train_imgs,train_labels);
        err2 = calc_err(W1,W2,test_imgs,test_labels);
        errors_train(1,loop) = err1;
        errors_test(1,loop) = err2;
    end

    figure()
    plot(linspace(1,loop_count,checkpoints), errors_train, 'b')
    hold on
    plot(linspace(1,loop_count,checkpoints), errors_test, 'r')
    hold off
    xlabel('Iterations');
    ylim([0 inf]);
    %xticks(0:Epochs)
    ylabel('Prob of error')
    legend('Train','Test')
    title(sprintf('Q5B Batch Learning, Sigmoid H = %d', H_Layer))
    sprintf('Q5B Batch Learning, Sigmoid, H
= %d: Train err: %f, Test err: %f', H_Layer,
errors_train(1,checkpoints),errors_test(1,checkpoints))
```

```
end

function err = calc_err(W1,W2,data,labels)
    g = W1 * data';
    y = 1 ./ (1 + exp(-g));
    u = W2 * y;
    z = exp(u) ./ sum(exp(u),1); %Softmax

    [values, indices] = max(z);
    indices = indices - 1;
    err = sum(sum((labels ~= indices'))) / size(data,1);
end
```

Published with MATLAB® R2018b

```
clc;
clear all;

load('HW2_Data.mat')

mu = 0; sigma = 1; size_letters = 784;
k_count = 10; %class count / output units count

train_images_squared = train_imgs.*train_imgs;
train_bias = ones(60000,1);
train_imgs = [train_imgs train_bias];

test_bias = ones(10000,1);
test_imgs = [test_imgs test_bias];

train_imgs_T = train_imgs';
test_imgs_T = test_imgs';

nu = 10^(-2);

train_labels_one_hot = zeros(10,60000);
for i = 1:60000
    train_labels_one_hot(train_labels(i) + 1,i) = 1;
end

% Homogenous coordinates, b should be added
bias_input = ones(10, 1);
output_weights = normrnd(mu, sigma, [10 size_letters]);
W1 = [output_weights bias_input];

Epochs = 15;
errors_train = zeros(1,Epochs+1);
errors_test = zeros(1,Epochs+1);

err1 = calc_err(W1, train_imgs, train_labels);
err2 = calc_err(W1, test_imgs, test_labels);
errors_train(1,1) = err1;
errors_test(1,1) = err2;

for Epoch = 1:Epochs
    % Backpropogation loop
    for loop = 1:60000
        x = train_imgs(loop,:);
        % **** FORWARD STEP ****
        y = W1 * x';
        z = exp(y) / sum(exp(y),1); %Softmax

        sens1 = train_labels_one_hot(:,loop) - z;
        Grad1 = sens1*x;
        W1 = W1 + nu * Grad1;
    end
```

```
        err1 = calc_err(W1,train_imgs, train_labels);
        err2 = calc_err(W1,test_imgs, test_labels);
        errors_train(1,Epoch+1) = err1;
        errors_test(1,Epoch+1) = err2;
end

figure()
plot(0:Epochs, errors_train, 'b')
hold on
plot(0:Epochs, errors_test, 'r')
hold off
xlabel('Epochs');
xticks(0:Epochs)
ylabel('Prob of error')
legend('Train','Test')
title('SGD, Single-Layer (Softmax)')
sprintf('SGD, Single Layer(Softmax), Train err: %f, Test err: %f',
 errors_train(1,16),errors_test(1,16))

function err = calc_err(W1,x,labels)
    y = W1 * x';
    z = exp(y) ./ sum(exp(y),1); %Softmax

    [values, indices] = max(z);
    indices = indices - 1;
    err = sum(sum((labels ~= indices'))) / size(x,1);
end
```

Published with MATLAB® R2018b

```
clc;
clear all;

load('HW2_Data.mat')

mu = 0; sigma = 1; size_letters = 784;
k_count = 10; %class count / output units count

train_images_squared = train_imgs.*train_imgs;
train_bias = ones(60000,1);
train_imgs = [train_imgs train_bias];

test_bias = ones(10000,1);
test_imgs = [test_imgs test_bias];

train_imgs_T = train_imgs';
test_imgs_T = test_imgs';

nu = 10^(-2);

train_labels_one_hot = zeros(10,60000);
for i = 1:60000
    train_labels_one_hot(train_labels(i) + 1,i) = 1;
end

Epochs = 15;
H_Values = [10, 20, 50];
for H_Layer = H_Values

    % Homogenous coordinates, b should be added
    bias_input = ones(H_Layer, 1);
    hidden_weights = normrnd(mu, sigma, [H_Layer size_letters]);

    W1 = [hidden_weights bias_input]; % Homogenous weights, input bias
 term added
    output_weights = normrnd(mu, sigma, [k_count H_Layer]);
    W2 = output_weights;

    errors_train = zeros(1,Epochs+1);
    errors_test = zeros(1,Epochs+1);

    err1 = calc_err(W1,W2,train_imgs, train_labels);
    err2 = calc_err(W1,W2,test_imgs, test_labels);
    errors_train(1,1) = err1;
    errors_test(1,1) = err2;

    for Epoch = 1:Epochs
        % Backpropogation loop
        for loop = 1:60000
            x = train_imgs(loop,:);
            % **** FORWARD STEP ****
            g = W1 * x';
```

```
            y = 1 ./ (1 + exp(-g));
            u = W2 * y;
            z = exp(u) / sum(exp(u),1); %Softmax

            sens2 = train_labels_one_hot(:,loop) - z;
            %sigmoid_derivative_g = (1 ./ (1 + exp(-g))) .* (1 - (1 ./
 (1 + exp(-g))));
            sigmoid_derivative_g = y.*(1-y);
            sens1 = sigmoid_derivative_g .* ((W2')*sens2);

            Grad2 = sens2*y';
            Grad1 = sens1*x;

            W1 = W1 + nu * Grad1;
            W2 = W2 + nu * Grad2;

        end

          err1 = calc_err(W1,W2,train_imgs, train_labels);
          err2 = calc_err(W1,W2,test_imgs, test_labels);
          errors_train(1,Epoch+1) = err1;
          errors_test(1,Epoch+1) = err2;
    end

    figure()
    plot(0:Epochs, errors_train, 'b')
    hold on
    plot(0:Epochs, errors_test, 'r')
    hold off
    xlabel('Epochs');
    xticks(0:Epochs)
    ylabel('Prob of error')
    legend('Train','Test')
    title(sprintf('SGD Sigmoid H = %d', H_Layer))
    sprintf('SGD, Sigmoid, H = %d: Train err: %f, Test err: %f',
 H_Layer, errors_train(1,16),errors_test(1,16))

end

function err = calc_err(W1,W2,x, labels)
    g = W1 * x';
    y = 1 ./ (1 + exp(-g));
    u = W2 * y;
    z = exp(u) ./ sum(exp(u),1); %Softmax

    [values, indices] = max(z);
    indices = indices - 1;
    err = sum(sum((labels ~= indices'))) / size(x,1);
end
```

Published with MATLAB® R2018b

```
clc;
clear all;

load('HW2_Data.mat')

mu = 0; sigma = 1; size_letters = 784;
k_count = 10; %class count / output units count

train_images_squared = train_imgs.*train_imgs;
train_bias = ones(60000,1);
train_imgs = [train_imgs train_bias];

test_bias = ones(10000,1);
test_imgs = [test_imgs test_bias];

train_imgs_T = train_imgs';
test_imgs_T = test_imgs';

nu = 2*10^(-2);

train_labels_one_hot = zeros(10,60000);
for i = 1:60000
    train_labels_one_hot(train_labels(i) + 1,i) = 1;
end

Epochs = 15;
H_Values = [10,20,50];
for H_Layer = H_Values
    % Homogenous coordinates, b should be added
    bias_input = ones(H_Layer, 1);
    hidden_weights = normrnd(mu, sigma, [H_Layer size_letters]);

    W1 = [hidden_weights bias_input]; % Homogenous weights, input bias
 term added
    output_weights = normrnd(mu, sigma, [k_count H_Layer]);
    W2 = output_weights;

    errors_train = zeros(1,Epochs+1);
    errors_test = zeros(1,Epochs+1);

    err1 = calc_err(W1,W2,train_imgs, train_labels);
    err2 = calc_err(W1,W2,test_imgs, test_labels);
    errors_train(1,1) = err1;
    errors_test(1,1) = err2;

    for Epoch = 1:Epochs
        % Backpropogation loop
        for loop = 1:60000
            x = train_imgs(loop,:);
            % **** FORWARD STEP ****
            g = W1 * x';
            y = (g > 0) .* g;
```

```
            u = W2 * y;
            z = exp(u) / sum(exp(u),1); %Softmax

            sens2 = train_labels_one_hot(:,loop) - z;
            relu_derivative_g = g > 0;
            sens1 = relu_derivative_g .* ((W2')*sens2);

            Grad2 = sens2*y';
            Grad1 = sens1*x;

            W1 = W1 + nu * Grad1;
            W2 = W2 + nu * Grad2;

        end
        err1 = calc_err(W1,W2,train_imgs, train_labels);
        err2 = calc_err(W1,W2,test_imgs, test_labels);
        errors_train(1,Epoch+1) = err1;
        errors_test(1,Epoch+1) = err2;
    end

    figure()
    plot(0:Epochs, errors_train, 'b')
    hold on
    plot(0:Epochs, errors_test, 'r')
    hold off
    xlabel('Epochs');
    xticks(0:Epochs)
    ylabel('Prob of error')
    legend('Train','Test')
    ylim([0 inf])
    title(sprintf('SGD RELU H = %d', H_Layer))
    sprintf('SGD, RELU, H = %d: Train err: %f, Test err: %f', H_Layer,
 errors_train(1,16),errors_test(1,16))
end

function err = calc_err(W1,W2,x, labels)
    g = W1 * x';
    y = (g > 0) .* g;
    u = W2 * y;
    z = exp(u) ./ sum(exp(u),1); %Softmax

    [values, indices] = max(z);
    indices = indices - 1;
    err = sum(sum((labels ~= indices'))) / size(x,1);
end
```

Published with MATLAB® R2018b