HOMEWORK 4

Saikumar Yadugiri
9083079468, saikumar@cs.wisc.edu

Instructions: Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file to Canvas. Late submissions may not be accepted. Please wrap your code and upload to a public GitHub repo, then attach the link below the instructions so that we can access it. You can choose any programming language (i.e. python, R, or MATLAB). Please check Piazza for updates about the homework.

GitHub Link `https://github.com/saikumarysk/cs760_hw4`

1 Best Prediction Under 0-1 Loss (10 pts)

Suppose the world generates a single observation $x \sim$ multinomial(θ), where the parameter vector $\theta = (\theta_1, \ldots, \theta_k)$ with $\theta_i \geq 0$ and $\sum_{i=1}^{k} \theta_i = 1$. Note $x \in \{1, \ldots, k\}$. You know θ and want to predict x. Call your prediction $\hat{x}$. What is your expected 0-1 loss:

$$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}]$$

using the following two prediction strategies respectively? Prove your answer.
Strategy 1: $\hat{x} \in \arg\max_x \theta_x$, the outcome with the highest probability.
As this strategy is deterministic, there is no randomness in the choice of $\hat{x}$. So, the expectation is taken on the world's prediction. Note that the values of $\hat{x}$ can be several as $\arg\max$ gives a set of values. However, as we need a single prediction to match the world's observation, we take some fixed value in that set. Then,

$$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_x \Pr[\hat{x} \neq x] = 1 - \Pr[\hat{x} = x] = 1 - \theta_{\hat{x}}$$

Strategy 2: You mimic the world by generating a prediction $\hat{x} \sim$ multinomial(θ). (Hint: your randomness and the world's randomness are independent)
Now, we have some randomness in the choice of $\hat{x}$. Similar to above, we have

$$\begin{aligned}
\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] &= \Pr[\hat{x} \neq x] \\
&= \sum_{i=1}^{k} \Pr[x \neq \hat{x} \mid \hat{x} = i] \Pr[\hat{x} = i] \\
&= \sum_{i=1}^{k} \Pr[x \neq i] \Pr[\hat{x} = i] \\
&= \sum_{i=1}^{k} (1 - \theta_i)\theta_i \\
&= 1 - \sum_{i=1}^{k} \theta_i^2
\end{aligned}$$

2 Best Prediction Under Different Misclassification Losses (6 pts)

Like in the previous question, the world generates a single observation $x \sim \text{multinomial}(\theta)$. Let $c_{ij} \geq 0$ denote the loss you incur, if $x = i$ but you predict $\hat{x} = j$, for $i, j \in \{1, \ldots, k\}$. $c_{ii} = 0$ for all i. This is a way to generalize different costs on false positives vs false negatives from binary classification to multi-class classification. You want to minimize your expected loss:

$$\mathbb{E}[c_{x\hat{x}}]$$

Derive your optimal prediction $\hat{x}$.

If we do not know the values of c_{ij} a-priori, we can calculate them or stick to the strategies in the previous question. So, I am assuming that we know the matrix $\mathbf{C} = [c_{ij}]_{ij}$ a-priori but not the world's observation. As we do not know x, we can choose a randomized prediction a la strategy 2 in previous question, by choosing $\hat{x}$ from a multinomial distribution as well. Then,

$$\begin{aligned}\mathbb{E}[c_{x\hat{x}}] &= \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij} \Pr[x = i] \Pr[x = j] \\ &= \sum_{i \neq j} c_{ij}\theta_i\theta_j\end{aligned}$$

We need to minimize this loss. Hence, we need $\arg\min_j \mathbb{E}[c_{x\hat{x}} \mid \hat{x} = j] = \arg\min_j \sum_{i \neq j} c_{ij}\theta_i$. So, our prediction $\hat{x} = \arg\min_j \sum_{i \neq j} c_{ij}\theta_i$.

3 Language Identification with Naive Bayes (8 pts each)

Implement a character-based Naive Bayes classifier that classifies a document as English, Spanish, or Japanese - all written with the 26 lower case characters and space.
The dataset is languageID.tgz, unpack it. This dataset consists of 60 documents in English, Spanish and Japanese. The correct class label is the first character of the filename: $y \in \{e, j, s\}$. (Note: here each file is a document in corresponding language, and it is regarded as one data.)
We will be using a character-based multinomial Naïve Bayes model. You need to view each document as a bag of characters, including space. We have made sure that there are only 27 different types of printable characters (a to z, and space) – there may be additional control characters such as new-line, please ignore those. Your vocabulary will be these 27 character types. (Note: not word types!)

1. Use files 0.txt to 9.txt in each language as the training data. Estimate the prior probabilities $\hat{p}(y = e)$, $\hat{p}(y = j)$, $\hat{p}(y = s)$ using additive smoothing with parameter $\frac{1}{2}$. Give the formula for additive smoothing with parameter $\frac{1}{2}$ in this case. Print and include in final report the prior probabilities. (Hint: Store all probabilities here and below in $\log()$ internally to avoid underflow. This also means you need to do arithmetic in log-space. But answer questions with probability, not log probability.)

   The formula for the prior probabilities are given by

$$\hat{p}\left[y = e\right] = \frac{\left(\sum_{i=1}^{N} \mathbb{1}\left[y^{(i)} = e\right]\right) + 0.5}{N + 1.5}$$

$$\hat{p}\left[y = s\right] = \frac{\left(\sum_{i=1}^{N} \mathbb{1}\left[y^{(i)} = s\right]\right) + 0.5}{N + 1.5}$$

$$\hat{p}\left[y = j\right] = \frac{\left(\sum_{i=1}^{N} \mathbb{1}\left[y^{(i)} = j\right]\right) + 0.5}{N + 1.5}$$

   From my calculations, I got $\hat{p}\left[y = e\right] = 0.33333$, $\hat{p}\left[y = s\right] = 0.33333$, and $\hat{p}\left[y = j\right] = 0.33333$.

2. Using the same training data, estimate the class conditional probability (multinomial parameter) for English

$$\theta_{i,e} := \hat{p}(c_i \mid y = e)$$

where c_i is the i-th character. That is, $c_1 = a, \ldots, c_{26} = z, c_{27} = space$. Again use additive smoothing with parameter $\frac{1}{2}$. Give the formula for additive smoothing with parameter $\frac{1}{2}$ in this case. Print θ_e and include in final report which is a vector with 27 elements.

The formula is given by

$$\theta_{i,e} = \hat{p}\left[c_i \mid y = e\right] = \frac{\left(\sum_{j=1}^{N}\sum_{k=1}^{M_j} \mathbb{1}\left[x_k^{(j)} = c_i, y^{(j)} = e\right]\right) + 0.5}{\left(\sum_{c\in C}\sum_{j=1}^{N}\sum_{k=1}^{M_j} \mathbb{1}\left[x_k^{(j)} = c, y^{(j)} = e\right]\right) + 1.5}$$

where M_j is the number of characters in the j-th file for english language, $x_k^{(j)}$ is the k-th character in the j-th file for english, and C is the set of 27 characters, i.e, $a - z$ and space. The conditional probabilities are shown in table 1.

θ_e					
a	0.06017	j	0.00142	s	0.06618
b	0.01113	k	0.00373	t	0.08013
c	0.02151	l	0.02898	u	0.02666
d	0.02197	m	0.02052	v	0.00928
e	0.10537	n	0.05792	w	0.0155
f	0.01893	o	0.06446	x	0.00116
g	0.01748	p	0.01675	y	0.01384
h	0.04722	q	0.00056	z	0.00063
i	0.05541	r	0.05382	space	0.17925

Table 1: Likelihood for English files

3. Print θ_j, θ_s and include in final report the class conditional probabilities for Japanese and Spanish.
   The required conditional probabilities are shown in tables 2 and 3.

θ_s					
a	0.13177	j	0.00234	s	0.04217
b	0.01087	k	0.05741	t	0.05699
c	0.00549	l	0.00143	u	0.07062
d	0.01723	m	0.0398	v	0.00024
e	0.0602	n	0.05671	w	0.01974
f	0.00388	o	0.09116	x	3e-05
g	0.01401	p	0.00087	y	0.01415
h	0.03176	q	0.0001	z	0.00772
i	0.09703	r	0.0428	space	0.12345

Table 2: Likelihood for Japanese Files

θ_j					
a	0.10456	j	0.00663	s	0.06577
b	0.00823	k	0.00028	t	0.03561
c	0.03753	l	0.05294	u	0.0337
d	0.03975	m	0.02581	v	0.00589
e	0.11381	n	0.05418	w	9e-05
f	0.0086	o	0.07249	x	0.0025
g	0.00718	p	0.02427	y	0.00786
h	0.00453	q	0.00768	z	0.00268
i	0.04986	r	0.0593	space	0.16826

Table 3: Likelihood for Spanish files

4. Treat e10.txt as a test document x. Represent x as a bag-of-words count vector (Hint: the vocabulary has size 27). Print the bag-of-words vector x and include in final report.

   The vector can be found in table 4.

x_{e10}					
a	164	j	3	s	186
b	32	k	6	t	225
c	53	l	85	u	65
d	57	m	64	v	31
e	311	n	139	w	47
f	55	o	182	x	4
g	51	p	53	y	38
h	140	q	3	z	2
i	140	r	141	space	498

Table 4: Bag of Words Vector for e10.txt

5. Compute $\hat{p}(x \mid y)$ for $y = e, j, s$ under the multinomial model assumption, respectively. Use the formula

$$\hat{p}(x \mid y) = \prod_{i=1}^{d} \theta_{i,y}^{x_i}$$

where $x = (x_1, \ldots, x_d)$. Show the three values: $\hat{p}(x \mid y = e), \hat{p}(x \mid y = j), \hat{p}(x \mid y = s)$. Hint: you may notice that we omitted the multinomial coefficient. This is ok for classification because it is a constant w.r.t. y.

The conditional probabilities are really low. So, I am printing $\log_{10}(\cdot)$ of these probabilities.

$$\log_{10}(\hat{p}(x \mid y = e)) = -3405.70056$$
$$\log_{10}(\hat{p}(x \mid y = j)) = -3810.12905$$
$$\log_{10}(\hat{p}(x \mid y = s)) = -3677.90439$$

6. Use Bayes rule and your estimated prior and likelihood, compute the posterior $\hat{p}(y \mid x)$. Show the three values: $\hat{p}(y = e \mid x), \hat{p}(y = j \mid x), \hat{p}(y = s \mid x)$. Show the predicted class label of x.

   As we do not know $\hat{p}(x)$, we can use $\hat{p}(x|y) \times \hat{p}(y)$ as a proxy for the posterior probability $\hat{p}(y|x)$. This is because posterior probability is the product of likelihood and prior probabilities. As likelihood probabilities are really low, I am use $\log_{10}(\cdot)$ to show posterior probabilities as well.

$$\hat{p}(y = e|x) \propto \log_{10}(\hat{p}(x|y = e)) + \log_{10}(\hat{p}(y = e)) = -3406.17768$$
$$\hat{p}(y = j|x) \propto \log_{10}(\hat{p}(x|y = j)) + \log_{10}(\hat{p}(y = j)) = -3810.60617$$
$$\hat{p}(y = s|x) \propto \log_{10}(\hat{p}(x|y = s)) + \log_{10}(\hat{p}(y = s)) = -3678.38151$$

   The highest value in all the posterior probabilities will be our prediction from classifier. Hence, the prediction is english.

7. Evaluate the performance of your classifier on the test set (files 10.txt to 19.txt in three languages). Present the performance using a confusion matrix. A confusion matrix summarizes the types of errors your classifier makes, as shown in the table below. The columns are the true language a document is in, and the rows are the classified outcome of that document. The cells are the number of test documents in that situation. For example, the cell with row = English and column = Spanish contains the number of test documents that are really Spanish, but misclassified as English by your classifier.

   The confusion matrix can be found in table 5.

	English	Japanese	Spanish
English	10	0	0
Japanese	0	10	0
Spanish	0	0	10

Table 5: Confusion Matrix for Testing Data

8. If you take a test document and arbitrarily shuffle the order of its characters so that the words (and spaces) are scrambled beyond human recognition. How does this shuffling affect your Naive Bayes classifier's prediction on this document? Explain the key mathematical step in the Naive Bayes model that justifies your answer.

   The arbitrary shuffling does not affect classifier's prediction. This is because the formulae for calculating the prior and likelihood probabilities are just counting the number of files or number of characters in each file respectively. In other words, we are predicting using character frequency in each language. So, shuffling of characters will not change the character frequency. Even if you insert additional characters which are not in the set $\{a - z, \text{space}\}$, we will not see any changes in the classifier's prediction.

4 Simple Feed-Forward Network (20pts)

In this exercise, you will derive, implement back-propagation for a simple neural network and compare your output with some standard library's output. Consider the following 3-layer neural network.

$$\hat{y} = f(x) = g(W_2\sigma(W_1 x))$$

Suppose $x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{d_1 \times d}$, and $W_2 \in \mathbb{R}^{k \times d_1}$ i.e. $f : \mathbb{R}^d \to \mathbb{R}^k$, Let $\sigma(z) = [\sigma(z_1), ..., \sigma(z_n)]$ for any $z \in \mathbb{R}^n$ where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid (logistic) activation function and $g(z_i) = \frac{exp(z_i)}{\sum_{i=1}^{k} exp(z_i)}$ is the softmax function. Suppose the true pair is (x, y) where $y \in \{0, 1\}^k$ with exactly one of the entries equal to 1, and you are working with the cross-entropy loss function given below,

$$L(x, y) = -\sum_{l=1}^{k} y \log(\hat{y})$$

1. Derive backpropagation updates for the above neural network. (5 pts)
   Consider the derivative w.r.to W_2 first as it will be easier to derive. Let's denote $W_2 = \left[w_{ij}^{(2)}\right]_{ij}$. Now, by chain rule,

$$\frac{\partial L(x,y)}{\partial w_{ij}^{(2)}} = -\sum_{l=1}^{k} \frac{y_l}{\hat{y}_l} \left(\sum_{m=1}^{k} \frac{\partial \hat{y}_l}{\partial z_m} \frac{\partial z_m}{\partial w_{ij}^{(2)}} \right)$$

   where z_m is the linear input to the output neuron, i.e, $z_m = w_{m1}^{(2)} a_1 + \ldots + w_{md_1}^{(2)} a_{d_1}$ (where the bias term is implicit in the linear term) and a_i is the activation of the hidden layer's neurons. Now, $\frac{\partial z_m}{\partial w_{ij}^{(2)}} = a_j$ if $m = i$ and 0 otherwise. So,

$$\begin{aligned}
\frac{\partial L(x,y)}{\partial w_{ij}^{(2)}} &= -\sum_{l=1}^{k} \frac{y_l}{\hat{y}_l} a_j \frac{\partial \hat{y}_l}{\partial z_i}, \\
\frac{\partial \hat{y}_i}{\partial z_i} &= \frac{\left(\sum_{l=1}^{k} e^{z_l}\right) e^{z_i} - e^{2z_i}}{\left(\sum_{l=1}^{k} e^{z_l}\right)^2} \\
&= \hat{y}_i(1 - \hat{y}_i) \\
\frac{\partial \hat{y}_l}{\partial z_i} &= \frac{\left(\sum_{m=1}^{k} e^{z_m}\right) \cdot 0 - e^{z_i} e^{z_l}}{\left(\sum_{m=1}^{k} e^{z_m}\right)^2} \quad (l \neq i) \\
&= -\hat{y}_l \hat{y}_i \\
\implies \frac{\partial L(x,y)}{\partial w_{ij}^{(2)}} &= -y_i(1 - \hat{y}_i)a_j + \sum_{l \neq i} y_l \hat{y}_i a_j \\
&= -y_i a_j + \left(\sum_{l=1}^{k} y_l\right) \hat{y}_i a_j \\
\frac{\partial L(x,y)}{\partial w_{ij}^{(2)}} &= (\hat{y}_i - y_i)a_j
\end{aligned}$$

   The last step comes because this is a classification task, y can be thought of as a one-hot vector. Finally, we can write,

$$\frac{\partial L(x,y)}{\partial W_2} = (\hat{y} - y)a^T \quad \text{(outer product)}$$

Now let's derive backpropogation updates for $W_1 = \left[w_{ij}^{(1)}\right]_{ij}$. By chain rule,

$$\frac{\partial L(x,y)}{\partial w_{ij}^{(1)}} = -\sum_{l=1}^{k} \frac{y_l}{\hat{y}_l} \left(\sum_{m=1}^{k} \frac{\partial \hat{y}_l}{\partial z_m} \frac{\partial z_m}{\partial w_{ij}^{(1)}} \right)$$

where z_m is same as described above. In z_m, the only place which $w_{ij}^{(1)}$ appears is in the neuron for a_i. So, $\frac{\partial z_m}{\partial w_{ij}^{(1)}} = w_{mi}^{(2)} \frac{\partial a_i}{\partial w_{ij}^{(1)}}$.

$$\frac{\partial a_i}{\partial w_{ij}^{(1)}} = a_i(1-a_i) \frac{\partial \tilde{z}_i}{\partial w_{ij}^{(i)}}$$

where $\tilde{z}_i = w_{i1}^{(1)} x_1 + \ldots + w_{id}^{(1)} x_d$. This is because a_i is the activation of first hidden layer using the sigmoid function and $\sigma'(z) = \sigma(z)(1-\sigma(z))$. Hence, $\frac{\partial \tilde{z}_i}{\partial w_{ij}^{(i)}} = x_j$.

$$\begin{aligned}
\frac{\partial L(x,y)}{\partial w_{ij}^{(1)}} &= -\sum_{l=1}^{k} \frac{y_l}{\hat{y}_l} \left(\sum_{m=1}^{k} \frac{\partial \hat{y}_l}{\partial z_m} w_{mi}^{(2)} a_i(1-a_i) x_j \right) \\
&= -a_i(1-a_i)x_j \sum_{l=1}^{k} \frac{y_l}{\hat{y}_l} \left(\sum_{m=1}^{k} \frac{\partial \hat{y}_l}{\partial z_m} w_{mi}^{(2)} \right) \\
&= -a_i(1-a_i)x_j \sum_{l=1}^{k} \frac{y_l}{\hat{y}_l} \left(\hat{y}_l(1-\hat{y}_l) w_{li}^{(2)} - \sum_{m \neq l} \hat{y}_l \hat{y}_m w_{mi}^{(2)} \right) \\
&= -a_i(1-a_i)x_j \sum_{l=1}^{k} y_l \left(w_{li}^{(2)} - \sum_{m=1}^{k} \hat{y}_m w_{mi}^{(2)} \right) \\
&= -a_i(1-a_i)x_j \sum_{l=1}^{k} y_l w_{li}^{(2)} - \left(\sum_{l=1}^{k} y_l \right) \left(\sum_{m=1}^{k} \hat{y}_m w_{mi}^{(2)} \right) \\
&= -a_i(1-a_i)x_j \left(\sum_{l=1}^{k} y_l w_{li}^{(2)} - \sum_{m=1}^{k} \hat{y}_m w_{mi}^{(2)} \right) \\
\implies \frac{\partial L(x,y)}{\partial w_{ij}^{(1)}} &= \left(\sum_{l=1}^{k} (\hat{y}_l - y_l) w_{li}^{(2)} \right) a_i(1-a_i) x_j
\end{aligned}$$

This can be written using $*$ as element-wise multiplication,

$$\frac{\partial L(x,y)}{\partial W_1} = \left(\left(W_2^T (\hat{y} - y) \right) * a * (1-a) \right) x^T$$

2. Implement it in NumPy or PyTorch using basic linear algebra operations. (e.g. You are not allowed to use auto-grad, built-in optimizer, model, etc. in this step. You can use library functions for data loading, processing, etc.). Evaluate your implementation on MNIST dataset, report test errors and learning curve. (10 pts)

   My implementation was 94.48% accurate. This means the test error is 5.52%. I used random weights from a normal distribution $\mathcal{N}(0, 0.01)$ as they provided the better accuracy. I trained for 10 epochs and the learning curve can be found in figure 1.

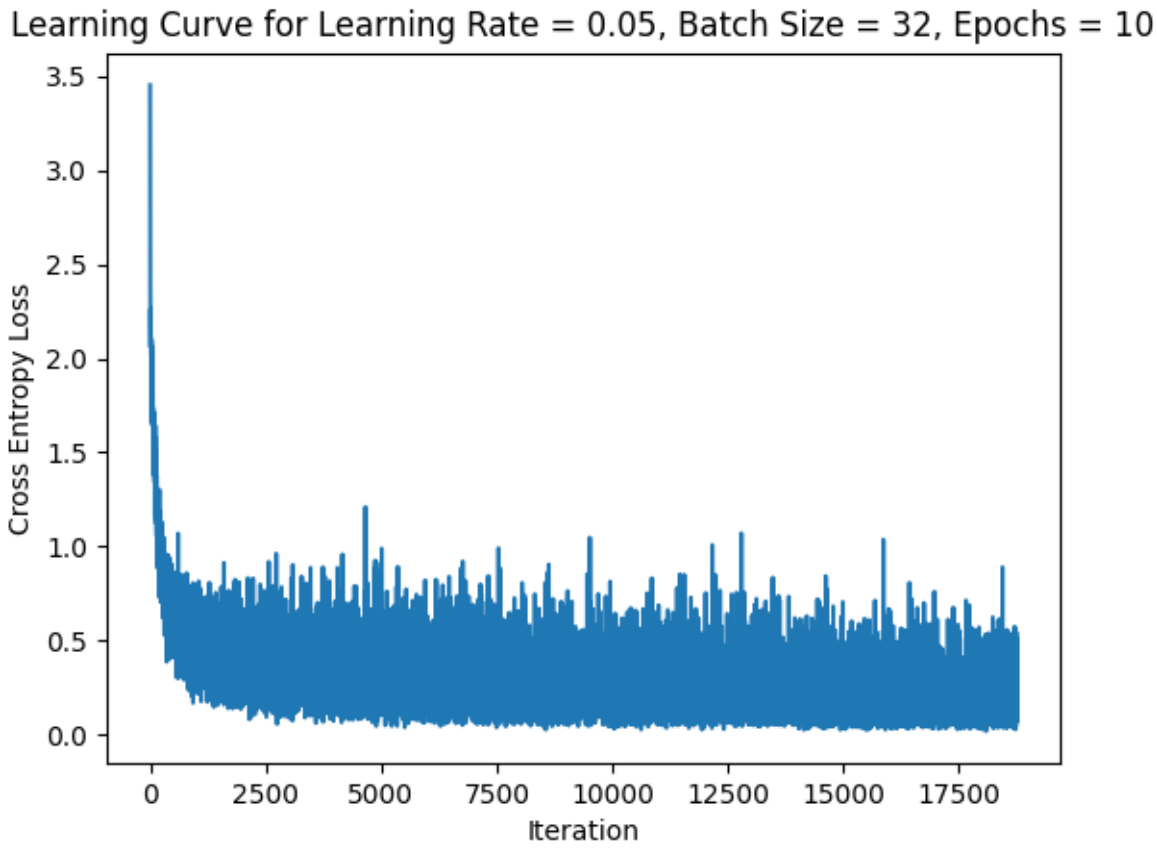


Figure 1: Learning Curve for MNIST Dataset on From Scratch Implementation of Neural Network

3. Implement the same network in PyTorch (or any other framework). You can use all the features of the framework e.g. auto-grad etc. Evaluate it on MNIST dataset, report test errors, and learning curve. (2 pts)

   The accuracy of the implementation was 91.71%. This means the test error is 8.29%. I used random weights from a normal distribution $\mathcal{N}(0, 0.01)$ just like the previous question. I trained for 10 epochs and the learning curve can be found in figure 2.

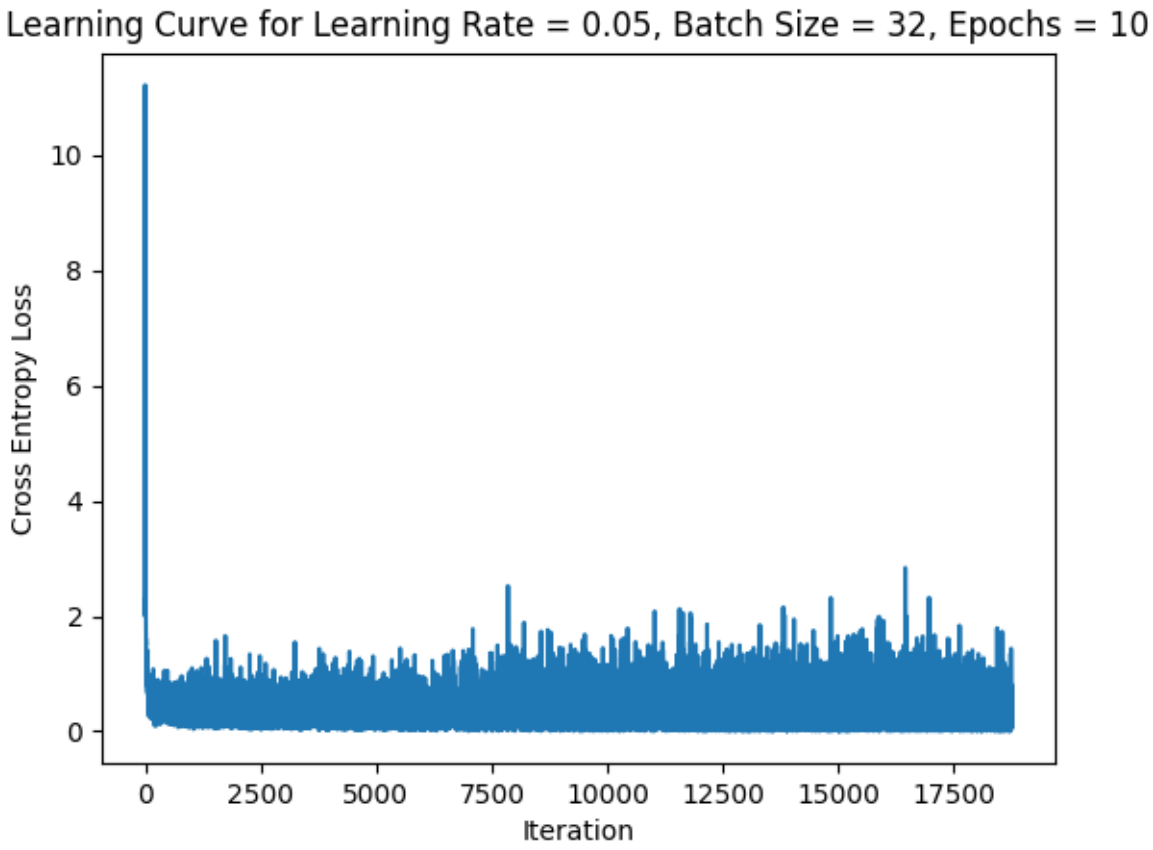


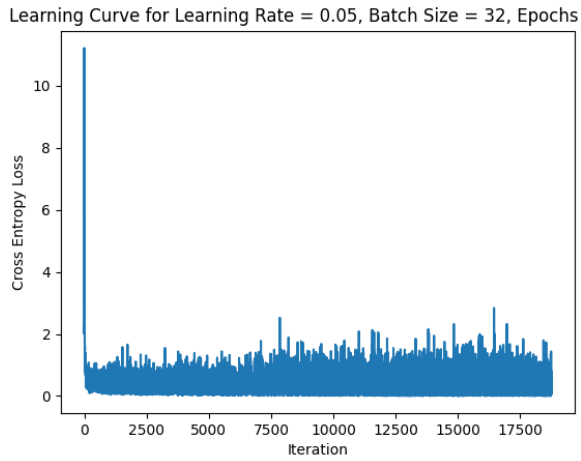
Figure 2: Learning Curve for MNIST Dataset on From Scratch Implementation of Neural Network

4. Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)

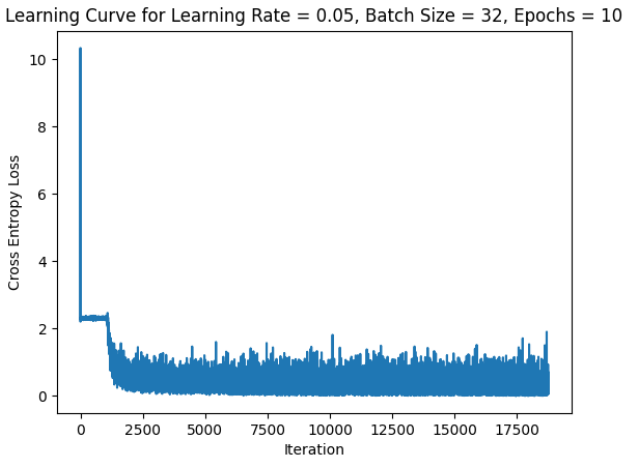
   The test error results can be found in table 6. The learning curves can be found in figure 3.

	Zero Weights	Random Weights
Scratch	21.22 %	5.52 %
PyTorch	7.74	8.29 %

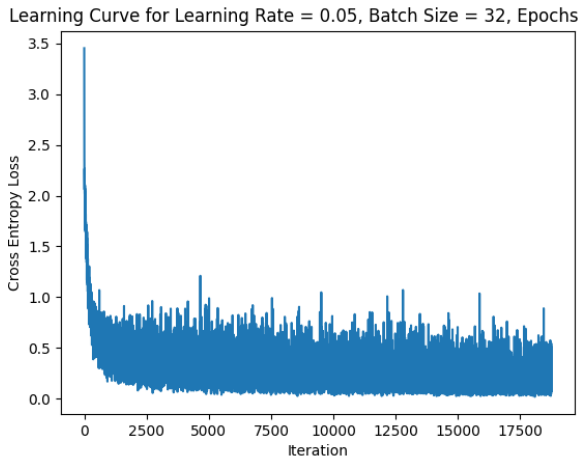
Table 6: Test Errors in Percentage (%) for My Implementations



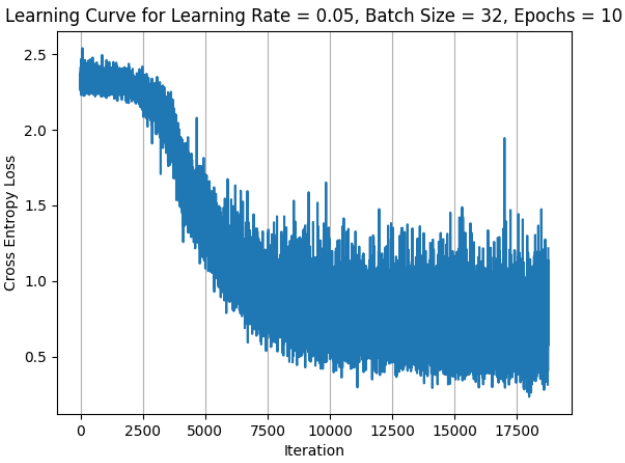
(a) Full PyTorch - Random Weights



(b) Full PyTorch - Zero Weights



(c) Scratch - Random Weights



(d) Scratch - Zero Weights

Figure 3: Zero Weights and Random Weights Learning Curves for Both My Implementations

You should play with different hyperparameters like learning rate, batch size, etc. for your own learning. You only need to report results for any particular setting of hyperparameters. You should mention the values of those along with the results. Use $d_1 = 300$ or $d_1 = 200$. For optimization use SGD (Stochastic gradient descent) without momentum, with some batch size say 32, 64, etc. MNIST can be obtained from here (https://pytorch.org/vision/stable/datasets.html)