HOMEWORK 4

Matt Myers
908-464-4252
GitHub

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 it 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.

1 Best Prediction

1.1 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.

1. Strategy 1: $\hat{x} \in \arg\max_x \theta_x$, the outcome with the highest probability.

   The expected 0-1 loss is $1 - \theta$, where θ is the highest probability of any outcome. This is because the probability of $\hat{x}$ being different from x is $1 - \theta$, and the probability of them being the same is θ. Since the 0-1 loss function is 1 when $\hat{x} \neq x$ and 0 otherwise, the expected 0-1 loss is simply the probability of $\hat{x}$ being different from x.

   $$\begin{aligned} E[1(\hat{x} \neq x)] &= P(\hat{x} \neq x) \cdot 1 + P(\hat{x} = x) \cdot 0 \\ &= (1 - \theta^*) \cdot 1 \\ &= 1 - \theta^* \end{aligned}$$

2. 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)

   We generate a prediction $\hat{x}$ following the same multinomial distribution as the world (θ). Since our randomness and the world's randomness are independent, $\hat{x}$ is generated with probability θ_i, and the world generates x with probability θ_i as well. The probability of $\hat{x}$ being different from x is $(1 - \theta_i)$. The expected 0-1 loss is the weighted average of these probabilities, so $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum \theta_i(1 - \theta_i)$ for $i = 1$ to k.

1.2 Under Different Misclassification Losses (6 pts)

Like in the previous question, the world generates a single observation $x \sim$ multinomial(θ). 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 of 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 $\hat{x}$.

We can find the optimal prediction $\hat{x}$ by minimizing the expected loss $\mathbb{E}[c_{x\hat{x}}]$ by choosing the prediction that minimizes the expected loss for each value of x. The expected loss for predicting $\hat{x}$ over all possible values of x is:

$$\mathbb{E}[c_{x\hat{x}}] = \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}\theta_i \mathbb{I}(\hat{x}=j) = \sum_{i=1}^{k} c_{i\hat{x}}\theta_i.$$

We want to find $\hat{x}$ that minimizes $\sum_{i=1}^{k} c_{i\hat{x}}\theta_i$. This is the same as finding $\hat{x}$ that maximizes $\sum_{i=1}^{k}(1-c_{i\hat{x}})\theta_i$. This means the optimal prediction $\hat{x}$ is given by:

$$\hat{x} = \arg\max_{j\in 1,\ldots,k} \sum_{i=1}^{k}(1-c_{ij})\theta_i.$$

This means we choose $\hat{x}$ to be the outcome that maximizes the expected probability of being correct, this is then weighted by the cost of being incorrect for each possible value of x.

2 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 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 the 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!)
In the following questions, you may use the additive smoothing technique to smooth categorical data, in case the estimated probability is zero. Given N data samples $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$, where $\boldsymbol{x}^{(i)} = [x_1^{(i)}, \ldots, x_j^{(i)}, \ldots, x_{M_i}^{(i)}]$ is a bag of characters, M_i is the total number of characters in $\boldsymbol{x}^{(i)}$, $x_j^{(i)} \in S, y^{(i)} \in L$ and we have $|S| = K_S, |L| = K_L$. Here S is the set of all character types, and L is the set of all classes of data labels. Then by the additive smoothing with parameter α, we can estimate the conditional probability as

$$P_\alpha(a_s \mid y = c_k) = \frac{(\sum_{i=1}^{N}\sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = c_k]) + \alpha}{(\sum_{b_s\in S}\sum_{i=1}^{N}\sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = c_k]) + K_S\alpha},$$

where $a_s \in S, c_k \in L$. Similarly, we can estimate the prior probability

$$P_\alpha(Y = c_k) = \frac{(\sum_{i=1}^{N} \mathbb{1}[y^{(i)} = c_k]) + \alpha}{N + K_L\alpha},$$

where $c_k \in L$ and N is the number of training samples.

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 the prior probabilities.

   To estimate the prior probabilities with additive smoothing, we use the formula:

   $$P_\alpha(Y = c_k) = \frac{(\sum_{i=1}^{N} \mathbb{1}[y^{(i)} = c_k]) + \alpha}{N + K_L\alpha}$$

   where $\alpha = \frac{1}{2}$ is the smoothing parameter, N is the total number of training samples, and K_L is the number of classes (in this case, $K_L = 3$).

   $$P_{\frac{1}{2}}(Y = c_k) = \frac{(\sum_{i=1}^{N} \mathbb{1}[y^{(i)} = c_k]) + \frac{1}{2}}{N + 3 \cdot \frac{1}{2}}$$

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 which is a vector with 27 elements.

To estimate the class conditional probability for English with additive smoothing, we use the formula:

$$P_\alpha(a_s \mid y = c_k) = \frac{(\sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = c_k]) + \alpha}{(\sum_{b_s \in S} \sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = c_k]) + K_S \alpha}$$

where $\alpha = \frac{1}{2}$ is the smoothing parameter, N is the total number of training samples, K_S is the size of the character vocabulary (in this case, $K_S = 27$), a_s is a character in the vocabulary, and e is the English language label.

$$P_{\frac{1}{2}}(a_s \mid y = c_k) = \frac{(\sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = c_k]) + \frac{1}{2}}{(\sum_{b_s \in S} \sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = c_k]) + 27 \cdot \frac{1}{2}}$$

$\theta_e = [0.06163156,\ 0.01194826,\ 0.02179464,\ 0.02213319,\ 0.10595438,\ 0.019989,\ 0.01601095,$
$0.04656576,\ 0.05471935,\ 0.0008605,\ 0.00399216,\ 0.03014572,\ 0.02216141,\ 0.05742781,$
$0.06507357,\ 0.01640593,\ 0.00069122,\ 0.05054381,\ 0.06377576,\ 0.08329924,\ 0.02565983,$
$0.00918337,\ 0.0154749,\ 0.00119906,\ 0.01338713,\ 0.00052194,\ 0.17944956]$

Letter	Percentage
a	0.06163
b	0.01195
c	0.02179
d	0.02213
e	0.106
f	0.01999
g	0.01601
h	0.04657
i	0.05472
j	0.0008605
k	0.003992
l	0.03015
m	0.02216
n	0.05743
o	0.06507
p	0.01641
q	0.0006912
r	0.05054
s	0.06378
t	0.0833
u	0.02566
v	0.009183
w	0.01547
x	0.001199
y	0.01339
z	0.0005219
	0.1794

3. Print θ_j, θ_s, the class conditional probabilities for Japanese and Spanish.

$$\theta_s = [0.10653656,\ 0.00959803,\ 0.03674721,\ 0.04048421,\ 0.11234968,\ 0.00736222,\ 0.00720252,\ 0.00477506,\ 0.0500024,\ 0.00672341,\ 0.00023955,\ 0.05313254,\ 0.02470575,\ 0.05434626,\ 0.07076353,\ 0.02416277,\ 0.0077455,\ 0.05853043,\ 0.06635578,\ 0.03534184,\ 0.0349905,\ 0.00592491,\ 0.00027149,\ 0.00257119,\ 0.00739416,\ 0.00330581,\ 0.16843669]$$

$$\theta_j = [1.32126462e-01,\ 9.92637505e-03,\ 5.21437852e-03,\ 1.63707233e-02,\ 5.99913382e-02,\ 3.48202685e-03,\ 1.48809008e-02,\ 3.14768298e-02,\ 9.89692508e-02,\ 2.09614552e-03,\ 5.71849285e-02,\ 1.19532265e-03,\ 4.08315288e-02,\ 5.69423993e-02,\ 9.03767865e-02,\ 7.10264184e-04,\ 5.19705500e-05,\ 4.25985275e-02,\ 4.25985275e-02,\ 5.79818103e-02,\ 7.02815071e-02,\ 1.90558683e-04,\ 2.02858380e-02,\ 1.73235167e-05,\ 1.38414898e-02,\ 7.74361195e-03,\ 1.22633175e-01]$$

Letter (θ_j)	Percentage
a	0.1321
b	0.009926
c	0.005214
d	0.01637
e	0.05999
f	0.003482
g	0.01488
h	0.03148
i	0.09897
j	0.002096
k	0.05718
l	0.001195
m	0.04083
n	0.05694
o	0.09038
p	0.0007103
q	5.197e-05
r	0.0426
s	0.0426
t	0.05798
u	0.07028
v	0.0001906
w	0.02029
x	1.732e-05
y	0.01384
z	0.007744
	0.1226

Letter (θ_s)	Percentage
a	0.1065
b	0.009598
c	0.03675
d	0.04048
e	0.1123
f	0.007362
g	0.007203
h	0.004775
i	0.05
j	0.006723
k	0.0002396
l	0.05313
m	0.02471
n	0.05435
o	0.07076
p	0.02416
q	0.007746
r	0.05853
s	0.06636
t	0.03534
u	0.03499
v	0.005925
w	0.0002715
x	0.002571
y	0.007394
z	0.003306
	0.1684

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.

$$x = [164.,\ 32.,\ 53.,\ 57.,\ 311.,\ 55.,\ 51.,\ 140.,\ 140.,\ 3.,\ 6.,\ 85.,\ 64.,\ 139.,\ 182.,\ 53.,\ 3.,\ 141.,\ 186.,\ 225.,\ 65.,\ 31.,\ 47.,\ 4.,\ 38.,\ 2.,\ 498.]$$

Letter	Percentage
a	164.0
b	32.0
c	53.0
d	57.0
e	311.0
f	55.0
g	51.0
h	140.0
i	140.0
j	3.0
k	6.0
l	85.0
m	64.0
n	139.0
o	182.0
p	53.0
q	3.0
r	141.0
s	186.0
t	225.0
u	65.0
v	31.0
w	47.0
x	4.0
y	38.0
z	2.0
	498.0

5. For the x of e10.txt, 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. Also, Store all probabilities here and below in $\log()$ internally to avoid underflow. This also means you need to do arithmetic in log space.

$$\hat{p}(x|y = e) = e^{-7841.662478537944}$$
$$\hat{p}(x|y = s) = e^{-8421.593490399442}$$
$$\hat{p}(x|y = j) = e^{-8818.782947292133}$$

6. For the x of e10.txt, use the 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.

The predicted class label of x is e. English has the highest value among the languages for its logarithm of the posterior probability:

$$\hat{p}(y = e|x) = e^{-7842.7610908266115}$$
$$\hat{p}(y = s|x) = e^{-8422.69210268811}$$
$$\hat{p}(y = j|x) = e^{-8819.881559580801}$$

Based on this information the test document is most likely in English. This probability is followed by Spanish and finally Japanese in order of likelihood.

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 output from my codes confusion matrix follows:

   Confusion Matrix:

   $$\begin{bmatrix}[10, & 0, & 0], \\ [0, & 10, & 0], \\ [0, & 0, & 10]]\end{bmatrix}$$

   This leads to the result:

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

8. Take a test document. 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.

   Example of shuffled document:

   gcastp naecoe nsiabard dtmcgitlaen eob osm peau tad rr

   aommalvrnneneukcnempfilletneli epoptlcce mipse eaimsd nejpo naxontoz jic doce ttsddtaesa di ssere s l

   The output and prediction for this were as follows:

   $$\hat{p}(y = e|x) = e^{-4937.7991901192745}$$
   $$\hat{p}(y = s|x) = e^{-5344.818784811843}$$
   $$\hat{p}(y = j|x) = e^{-4378.6596573223715}$$

   Predicted language: j, Actual language: j

   This was still able to correctly classify the language even while shuffled beyond human recognition. The way that humans view language is much different than Naive Bayes. Naive Bayes utilizes the bag-of-words model, this ignores the order of words in the document and only considers the frequency of each word in the document. This means that it affects this process much less than scrambling the documents affects our ability to read it.

   The key mathematical step in the Naive Bayes model that justifies this is the assumption of conditional independence between the words in the document, given the class label. This lets us treat words as a set and compute the likelihood of the document as the product of the likelihood of each word.

3 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.

$$\hat{y} = f(x) = g(W_3\sigma(W_2\sigma(W_1x)))$$

Suppose $x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{d_1 \times d}$, $W_2 \in \mathbb{R}^{d_2 \times d_1}$, $W_3 \in \mathbb{R}^{k \times d_2}$ 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,

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

1. Derive backpropagation updates for the above neural network. (5 pts)

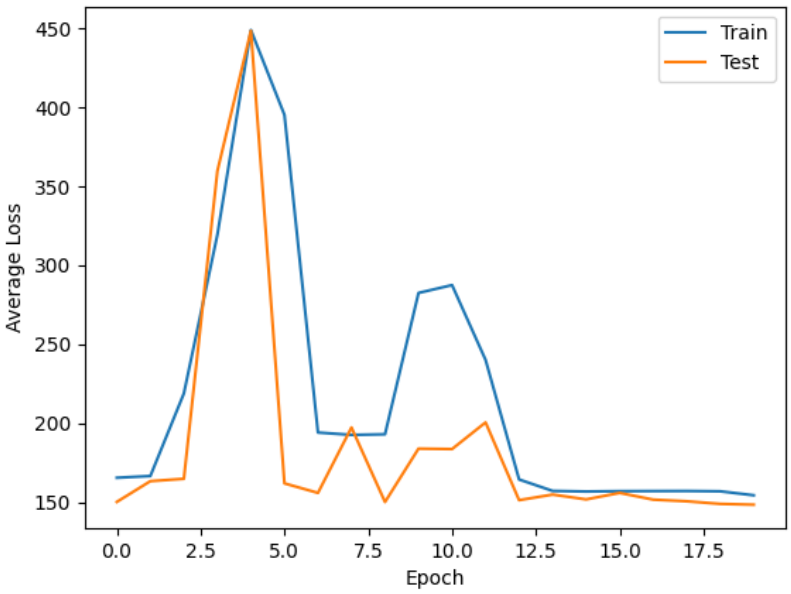
   Let $z_1 = W_1x$, $h_1 = \sigma(z_1)$, $z_2 = W_2h_1$, $h_2 = \sigma(z_2)$, $z_3 = W_3h_2$ and $\hat{y} = g(z_3)$. Then the loss function is $L(x, y) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$:

$$\begin{aligned}
\frac{\partial L}{\partial z_3} &= \frac{\partial}{\partial z_3}\left(-\sum_{i=1}^{k} y_i \log(\hat{y}_i)\right) \\
&= \hat{y} - y \\
\frac{\partial L}{\partial W_3} &= \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial W_3} \\
&= (\hat{y} - y)h_2^T \\
\frac{\partial L}{\partial h_2} &= \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial h_2} \\
&= W_3^T(\hat{y} - y) \\
\frac{\partial L}{\partial z_2} &= \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial z_2} \\
&= \frac{\partial L}{\partial h_2}\text{diag}(\sigma'(z_2)) \\
&= \frac{\partial L}{\partial h_2}\text{diag}(\sigma(z_2) \circ (1 - \sigma(z_2))) \\
\frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial W_2} \\
&= (\frac{\partial L}{\partial z_2}h_1^T) \\
\frac{\partial L}{\partial h_1} &= \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial h_1} \\
&= W_2^T\frac{\partial L}{\partial z_2} \\
\frac{\partial L}{\partial z_1} &= \frac{\partial L}{\partial h_1}\frac{\partial h_1}{\partial z_1} \\
&= \frac{\partial L}{\partial h_1}\text{diag}(\sigma'(z_1)) \\
&= \frac{\partial L}{\partial h_1}\text{diag}(\sigma(z_1) \circ (1 - \sigma(z_1))) \\
\frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial z_1}x^T
\end{aligned}$$

Let $\circ$ be the element-wise product, diag(v) is a diagonal matrix with the elements of vector v on the diagonal.

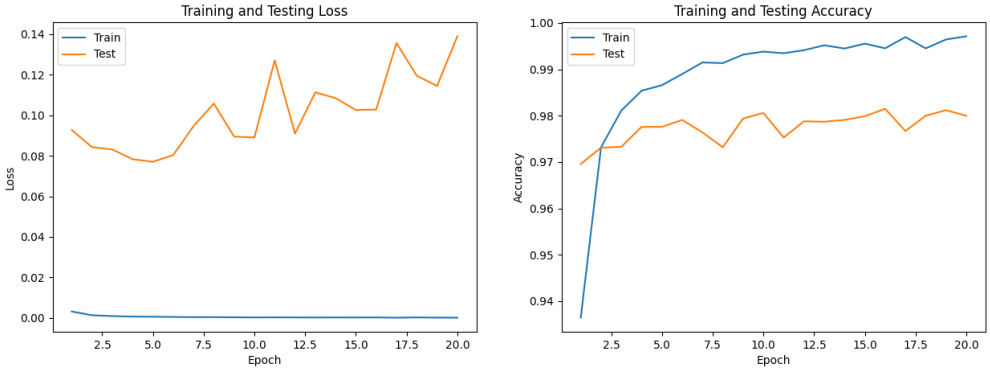
2. Implement it in NumPy or PyTorch using basic linear algebra operations. (e.g. You are not allowed to use auto-grad, built-in optimizer, model, etc. in this step. You can use library functions for data loading, processing, etc.). Evaluate your implementation on MNIST dataset, report test errors and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

   Hyper Parameters:
   Batch Size: 64
   Learning Rate: 0.1



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)

   Hyper Parameters:
   Batch Size: 64
   Learning Rate: 0.001

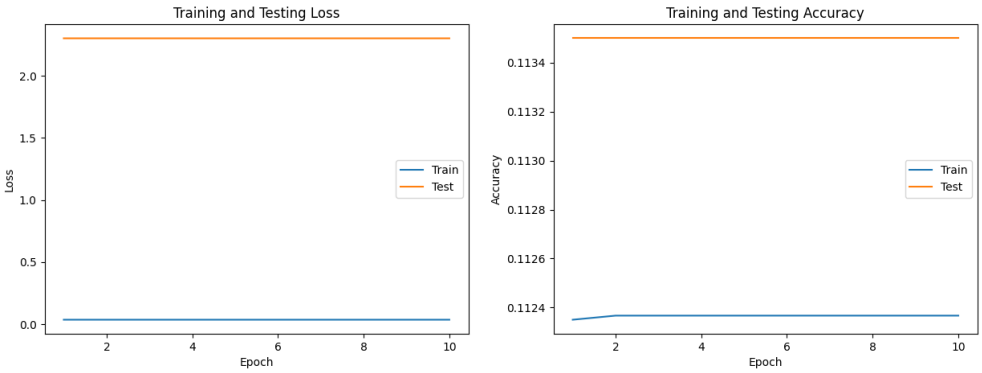


Epoch	Test Error (%)
1	3.0399999999999983%
2	2.6900000000000035%
3	2.6699999999999946%
4	2.2399999999999975%
5	2.2399999999999975%
6	2.090000000000003%
7	2.3599999999999954%
8	2.6800000000000046%
9	2.059999999999995%
10	1.9399999999999973%
11	2.4700000000000055%
12	2.1199999999999997%
13	2.1299999999999986%
14	2.090000000000003%
15	2.0100000000000007%
16	1.849999999999996%
17	2.3299999999999987%
18	2.0000000000000018%
19	1.880000000000004%
20	2.0000000000000018%

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)

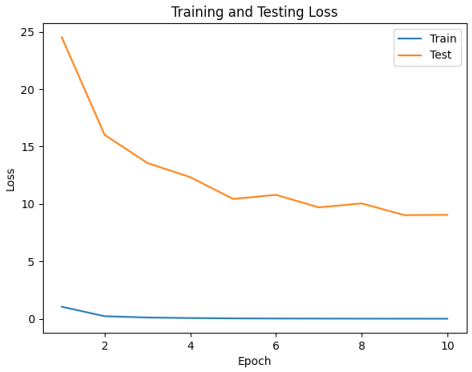
   Hyper Parameters:
   Batch Size: 64
   Learning Rate: 0.001

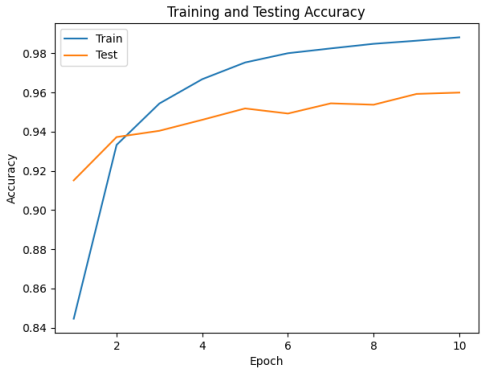
   All weights 0:



Epoch	Test Error (%)
1	88.64999999999999%
2	88.64999999999999%
3	88.64999999999999%
4	88.64999999999999%
5	88.64999999999999%
6	88.64999999999999%
7	88.64999999999999%
8	88.64999999999999%
9	88.64999999999999%
10	88.64999999999999%

Random between -1 and 1:





Epoch	Test Error (%)
1	8.479999999999999%
2	6.269999999999998%
3	5.949999999999999%
4	5.389999999999995%
5	4.810000000000003%
6	5.069999999999997%
7	4.549999999999999%
8	4.620000000000002%
9	4.069999999999996%
10	4.0000000000000036%

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 $d_1 = 300$, $d_2 = 200$, $d_3 = 100$. 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)

https://github.com/myersmt/Hw004-Comp760