HOMEWORK 4

Batchu S S S S Harish Babu
sbatchu2(NetID) - 9084603043

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.
https://github.com/bharishb/ECE760_spring2022/tree/main/HW4

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.
   Given the θ , and predicting x with the highest probability θ is like giving constant output to all the observations world generates. Therefore
   $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}]$ is nothing but the chances where world observation is not the x with highest probability which is therefore
   $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \arg\max_x \theta_x$

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)
   In this case, Loss is nothing but both the multinomials give different outputs. This probability can be simplified to total minus the probability when they give same outcome.

   $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = P[\hat{x} \neq x] = 1 - P[\hat{x} = x] = 1 - \sum_{i=1}^{k} P_{\theta i}[\hat{x} = x] = 1 - \sum_{i=1}^{k} P_{\theta i}[\hat{x}] * P_{\theta i}[x] = 1 - \sum_{i=1}^{k} \theta_i^2$
   $[P(X, Y) = P(x) * P(Y)]$ since X,Y are independent

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 $\hat{x}$.
By trying to formulize the expectation with probabilities, we can try to find the optimal $\hat{x}$

$$\mathbb{E}[c_{x\hat{x}}] = \sum_{i=1}^{k} \sum_{j=1}^{k} c_{ij} P(x = i, \hat{x} = j) = \sum_{i=1}^{k} \sum_{j=1}^{k} c_{ij} P(x = i) * P(\hat{x} = j)$$

$$\mathbb{E}[c_{x\hat{x}}] = \sum_{i=1}^{k} \sum_{j=1}^{k} c_{ij} \theta_i * P(\hat{x} = j) = \sum_{j=1}^{k} \sum_{i=1, i \neq j}^{k} c_{ij} \theta_i * P(\hat{x} = j) = \sum_{j=1}^{k} P(\hat{x} = j) \sum_{i=1, i \neq j}^{k} c_{ij} \theta_i$$

To minimize the loss, we need to minimize the $\sum_{i=1,i\neq j}^{k} c_{ij}\theta_i$. Assuming to predict only one value , that means prob = 1 which makes $\hat{x} = argmin_j \sum_{i=1,i\neq j}^{k} c_{ij}\theta_i$

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.
   with alpha = 0.5

   $$P_\alpha(Y = c_k) = \frac{(\sum_{i=1}^{30} \mathbb{1}[y^{(i)} = c_k]) + 0.5}{30 + 3 * 0.5} = \frac{10 + 0.5}{30 + 3 * 0.5},$$

   $\hat{p}(y = e) = 0.333$
   $\hat{p}(y = s) = 0.333$
   $\hat{p}(y = j) = 0.333$

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.

   $$P_\alpha(a_s \mid y = e) = \frac{(\sum_{i=1}^{30}\sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = e]) + 0.5}{(\sum_{b_s \in S}\sum_{i=1}^{30}\sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = e]) + 0.5 * 27},$$

   array([[6.01685115e-02, 1.11349744e-02, 2.15099950e-02, 2.19725756e-02, 1.05369238e-01, 1.89327606e-02, 1.74789361e-02, 4.72162564e-02, 5.54105402e-02, 1.42078308e-03, 3.73368578e-03, 2.89773666e-02, 2.05187510e-02, 5.79216917e-02, 6.44639022e-02, 1.67520238e-02, 5.61704940e-04, 5.38245498e-02, 6.61820585e-02, 8.01255576e-02, 2.66644639e-02, 9.28465224e-03, 1.54964480e-02, 1.15645135e-03, 1.38443747e-02, 6.27787874e-04, 1.79249959e-01]])

'a': 0.0601685114819098, 'b': 0.011134974392863043, 'c': 0.021509995043779945, 'd': 0.021972575582355856, 'e': 0.1053692383941847, 'f': 0.018932760614571286, 'g': 0.017478936064761277, 'h': 0.047216256401784236, 'i': 0.055410540227986124, 'j': 0.001420783082768875, 'k': 0.0037336857756484387, 'l': 0.028977366595076822, 'm': 0.020518751032545846, 'n': 0.057921691723112505, 'o': 0.06446390219725756, 'p': 0.01675202378985627, 'q': 0.0005617049396993227, 'r': 0.053824549810011564, 's': 0.06618205848339666, 't': 0.08012555757475633, 'u': 0.026664463902197257, 'v': 0.009284652238559392, 'w': 0.015496448042293078, 'x': 0.001156451346439782, 'y': 0.013844374690236246, 'z': 0.0006277878737815959, ' ': 0.1792499586981662

```
charcount = np.zeros([3,20,27])
likelihood = np.zeros([3,27]);
alpha = 0.5;
for i in range(3):
likelihood[i]= (np.sum(charcount[i][:10],axis=0) + alpha)/(np.sum(np.sum(charcount[i][:10],axis=0)) + 27*alpha)
```

3. Print θ_j, θ_s, the class conditional probabilities for Japanese and Spanish.
   For Japanese :
   array([1.31765610e-01, 1.08669066e-02, 5.48586603e-03, 1.72263182e-02, 6.02047591e-02, 3.87854223e-03, 1.40116706e-02, 3.17621161e-02, 9.70334393e-02, 2.34110207e-03, 5.74094133e-02, 1.43261470e-03, 3.97987351e-02, 5.67105769e-02, 9.11632132e-02, 8.73545547e-04, 1.04825466e-04, 4.28037318e-02, 4.21747790e-02, 5.69901115e-02, 7.06174220e-02, 2.44592753e-04, 1.97421294e-02, 3.49418219e-05, 1.41514379e-02, 7.72214263e-03, 1.23449457e-01])

   'a': 0.1317656102589189, 'b': 0.010866906600510151, 'c': 0.005485866033054963, 'd': 0.01722631818022992, 'e': 0.06020475907613823, 'f': 0.003878542227191726, 'g': 0.014011670568503443, 'h': 0.03176211607673224, 'i': 0.09703343932352633, 'j': 0.0023411020650616725, 'k': 0.05740941332681086, 'l': 0.0014326146496530277, 'm': 0.03979873510604843, 'n': 0.05671057688947902, 'o': 0.09116321324993885, 'p': 0.0008735455466648031, 'q': 0.00010482546559977637, 'r': 0.04280373178657535, 's': 0.0421747789929767, 't': 0.056990111464411755, 'u': 0.07061742199238269, 'v': 0.0002445927530661449, 'w': 0.01974212935462455, 'x': 3.4941821866592126e-05, 'y': 0.01415143785596981, 'z': 0.00772214263251686, ' ': 0.12344945665466997

   For Spanish :
   array([1.04560451e-01, 8.23286362e-03, 3.75258241e-02, 3.97459221e-02, 1.13810860e-01, 8.60287996e-03, 7.18448398e-03, 4.53270019e-03, 4.98597021e-02, 6.62945947e-03, 2.77512257e-04, 5.29431717e-02, 2.58086399e-02, 5.41765595e-02, 7.24923684e-02, 2.42669051e-02, 7.67783910e-03, 5.92951189e-02, 6.57704049e-02, 3.56140730e-02, 3.37023219e-02, 5.88942678e-03, 9.25040856e-05, 2.49761031e-03, 7.86284728e-03, 2.68261848e-03, 1.68264932e-01])

   'a': 0.10456045141993771, 'b': 0.008232863618143134, 'c': 0.03752582405722919, 'd': 0.039745922111559924, 'e': 0.1138108599796491, 'f': 0.00860287996053159, 'g': 0.0071844839813758445, 'h': 0.0045327001942585795, 'i': 0.049859702136844375, 'j': 0.006629459467793161, 'k': 0.0002775122567913416, 'l': 0.052943171656748174, 'm': 0.02580863988159477, 'n': 0.054176559464709693, 'o': 0.07249236841293824, 'p': 0.02426690512164287, 'q': 0.0076778391045604 51, 'r': 0.05929511886774999, 's': 0.06577040485954797, 't': 0.03561407295488884, 'u': 0.03370232185254849, 'v': 0.00588942678301625, 'w': 9.250408559711388e-05, 'x': 0.0024976103111220747, 'y': 0.007862847275754679, 'z': 0.0026826184823163022, ' ': 0.16826493170115014

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. 'a': 164, 'b': 32, 'c': 53, 'd': 57, 'e': 311, 'f': 55, 'g': 51, 'h': 140, 'i': 140, 'j': 3, 'k': 6, 'l': 85, 'm': 64, 'n': 139, 'o': 182, 'p': 53, 'q': 3, 'r': 141, 's': 186, 't': 225, 'u': 65, 'v': 31, 'w': 47, 'x': 4, 'y': 38, 'z': 2, ' ': 498
   $x = [164, 32, 53, 57, 311, 55, 51, 140, 140, 3, 6, 85, 64, 139, 182, 53, 3, 141, 186, 225, 65, 31, 47, 4, 38, 2, 498]$

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.
Log Probabilities

$$\hat{p}(x \mid y) = [-7841.86544706 - 8771.43307908 - 8467.28204401]$$

for i in range(3):
for j in range(27):
prob[i] = prob[i] + x[j]*np.log(likelihood[i][j])
problabel = np.log(1/3);

$\hat{p}(x \mid y = e) = -7841.865447060635$
$\hat{p}(x \mid y = j) = -8771.433079075032$
$\hat{p}(x \mid y = s) = -8467.282044010557$

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.
Log Probabilities
print(prob+problabel) (adding up since log probs)

$$\hat{p}(y \mid x) = [-7842.96405935 - 8772.53169136 - 8468.3806563]$$

$\hat{p}(y = e \mid x) = -7842.964059349303$
$\hat{p}(y = j \mid x) = -8772.5316913637$
$\hat{p}(y = s \mid x) = -8468.380656299225$
np.argmax(prob+problabel)
File : ./e10.txt Prediction : e

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.

	English	Spanish	Japanese
English			
Spanish			
Japanese			

pred = np.zeros([3, 10])
for i in range(3):
for j in range(10):
pred[i][j] = predict(charcount[i][j+10]);(in test documents from 10 to 19)
Output labels :

[[0.0.0.0.0.0.0.0.0.0.], [1.1.1.1.1.1.1.1.1.1.], [2.2.2.2.2.2.2.2.2.2.]]

	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.

   Naive Bayes Classifier depends on frequency of letters in a document. As seen above code snippets, char-count is used to find frequency to characters in the documents which will be same if the documents are shuffled too. So, the order of the letters and characters doesnot matter. It assumes conditional independence of features.

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

   where $x = (x_1, \ldots, x_d)$

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 x and Y be the inputs and the true label for the Neural Network

$$W_1X = Z_1 \quad A_1 = \sigma(Z_1)$$
$$W_2A_1 = Z_2 \quad A_2 = \sigma(Z_2)$$
$$W_3A_2 = Z_3 \quad \hat{Y} = g(Z_3)$$

$$L'_{W_3} = \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} * \frac{\partial Z_3}{\partial W_3}$$

$$Softmax derivative:$$

$$\frac{\partial \hat{Y}}{\partial Z_3} = \frac{(\sum_{i=1}^{k} exp(z_i)) * exp(z_j) - exp(z_j) * exp(z_j)}{(\sum_{i=1}^{k} exp(z_i))^2} = \hat{Y} - \hat{Y}^2 if Label$$

$$\frac{\partial \hat{Y}}{\partial Z_3} = \frac{(\sum_{i=1}^{k} exp(z_i)) * 0 - exp(z_j) * exp(z_j)}{(\sum_{i=1}^{k} exp(z_i))^2} = 0 - \hat{Y}^2 if NotLabel \quad (1)$$

$$Therefore$$

$$\frac{\partial \hat{Y}}{\partial Z_3} = (Y - \hat{Y}) * (\hat{Y})$$

$$\frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} = \frac{\partial Y * log(\hat{Y})}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} = -Y * [\frac{1}{\hat{y}}]. * [\hat{y} * (y - \hat{y})]$$

$$= [\hat{y} - y] = \hat{Y} - Y$$

$$L'_{W_3} = \frac{\partial L}{\partial W_3} = (\hat{Y} - Y) * A_2^T$$

$$
\begin{aligned}
L'_{W_2} &= \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} * \frac{\partial Z_3}{\partial A_2} * \frac{\partial A_2}{\partial Z_2} * \frac{\partial Z_2}{\partial W_2} \\
L'_{W_2} &= \frac{\partial L}{\partial W_2} = diag(A2') * W_3^T * (\hat{Y} - Y) * A_1^T \\
&= diag(A2 * (1 - A2)) * W_3^T * (\hat{Y} - Y) * A_1^T
\end{aligned} \tag{2}
$$

$$
\begin{aligned}
L'_{W_1} &= \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} * \frac{\partial Z_3}{\partial A_2} * \frac{\partial A_2}{\partial Z_2} * \frac{\partial Z_2}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} \\
L'_{W_1} &= \frac{\partial L}{\partial W_1} = diag(A1') * W_2^T * diag(A2') * W_3^T * (\hat{Y} - Y) * X^T \\
&= diag(A1 * (1 - A1)) * W_2^T * diag(A2 * (1 - A2)) * W_3^T * (\hat{Y} - Y) * X^T
\end{aligned} \tag{3}
$$

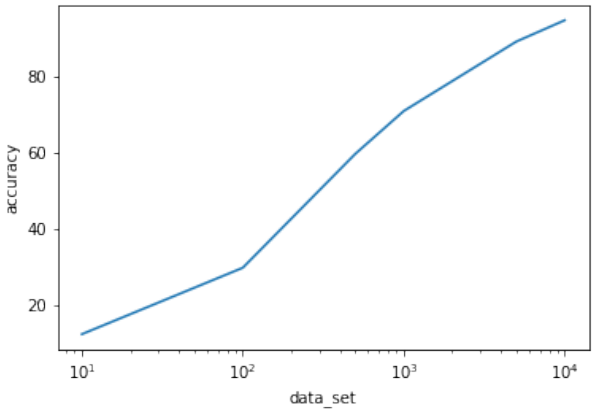
$$
\begin{aligned}
W_1(t+1) &= W_1(t) - \alpha * L'_{W_1} \\
W2(t+1) &= W_2(t) - \alpha * L'_{W_2} \\
W_3(t+1) &= W_3(t) - \alpha * L'_{W_3}
\end{aligned}
$$

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)

data set	test error(out of 10000)
100	7011
500	4065
1000	2978
5000	1045
10000	535

learning rate = 0.001

number of epochs = 20
batch size = 1

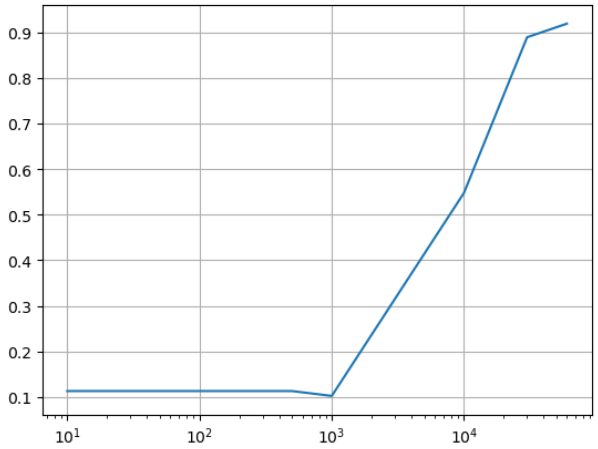


3. Implement the same network in PyTorch (or any other framework). You can use all the features of the framework e.g. auto-grad etc. Evaluate it on MNIST dataset, report test errors, and learning curve. (2 pts)

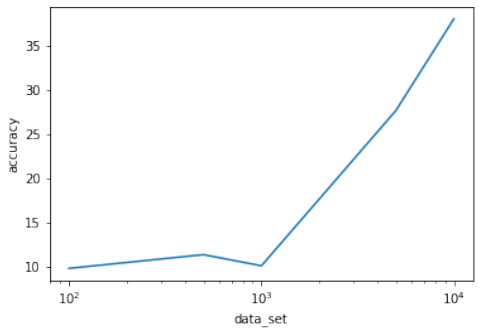
data size	test error(out of 10000)
10	8900
100	8870
500	8870
1000	8800
10000	4520
30000	1110
60000	810

learning rate = 0.01

number of epochs = 20
Accuracy = 92 percent
batch size = 32

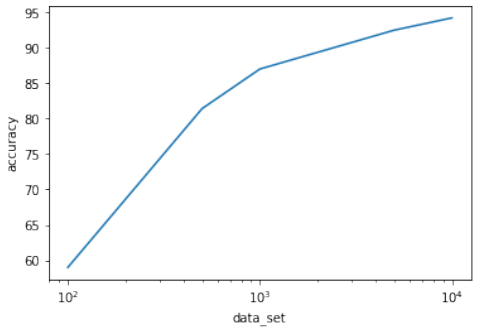


4. Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)
   alpha = learning rate = 0.08
   num epochs = 50
   Implementation = pytorch
   1. Weights Initialized to 0



dataset	testerror(out of 10000)
100	9025
500	8990
1000	8865
5000	7134
10000	6251

   1. Weights Initialized to [-1,1]



dataset	testerror(out of 10000)
100	4100
500	1858
1000	1300
5000	752
10000	578

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)