HOMEWORK 4

Kincannon Wilson
908 462 4023

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.

   In this strategy, our prediction $\hat{x}$ is incorrect if our observation actually belongs to some other class x, $\hat{x} \neq x$. x will belong to class i with probability θ_i. Thus, in expectation:

   $$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_{i=1}^{k} 1 * \theta_i \quad s.t. \quad i \neq \hat{x}.$$

   Since

   $$\sum_{i=1}^{k} \theta_i = 1,$$

   we know that

   $$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \theta_x.$$

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)

   By definition of expectation,

   $$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_{i=1}^{k} 1 * P(x = i \wedge \hat{x} \neq i)$$

   (by independence of x and $\hat{x}$, as well as using the substitution from part 1 above)

   $$= \sum_{i=1}^{k} \theta_i (1 - \theta_i)$$

$$= \sum_{i=1}^{k} \theta_i - \sum_{i=1}^{k} \theta_i^2$$

$$= 1 - \sum_{i=1}^{k} \theta_i^2$$

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 $\hat{x}$.

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

(By independence of the world and our prediction)

$$= \sum_{j=1}^{k} \sum_{i=1}^{k} c_{ij} P(x = i) P(\hat{x} = j)$$

$$= \sum_{j=1}^{k} \sum_{i=1}^{k} c_{ij} \theta_i \theta_j$$

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.

   For each language c_k,

   $$P_\alpha(Y = c_k) = \frac{10 + \frac{1}{2}}{30 + 3 * \frac{1}{2}},$$

   since there are 10 files for each language in the training set, $\alpha = \frac{1}{2}$, $N = 10 + 10 + 10 = 30$, and $K_L = len(['e', 's', 'j']) = 3$.

   My printed output:

```
Prior probability for English: 0.3333333333333333
Prior probability for Spanish: 0.3333333333333333
Prior probability for Japanese: 0.3333333333333333
```

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.

   My printed output:

```
Conditional probabilities for English:
 [0.06016851 0.01113497 0.02151    0.02197258 0.10536924 0.01893276
 0.01747894 0.04721626 0.05541054 0.00142078 0.00373369 0.02897737
 0.02051875 0.05792169 0.0644639  0.01675202 0.0005617  0.05382455
 0.06618206 0.08012556 0.02666446 0.00928465 0.01549645 0.00115645
 0.01384437 0.00062779 0.17924996]
```

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

   My printed output:

```
Conditional probabilities for Spanish:
 [0.08315116 0.00962198 0.02979144 0.03116373 0.10976719 0.01357928
 0.01214316 0.02513204 0.05254592 0.00410091 0.00193078 0.04137612
 0.02324913 0.0559926  0.06863042 0.02063221 0.00422857 0.05666278
 0.06598159 0.05710958 0.03030206 0.00751568 0.00751568 0.00183504
 0.01073896 0.00167547 0.17362651]
Conditional probabilities for Japanese:
 [0.09841005 0.01000427 0.02216719 0.02679129 0.09424617 0.01053024
 0.01272175 0.02720768 0.06650157 0.0035393  0.01931822 0.02885132
 0.02843493 0.05622336 0.07570594 0.01443114 0.00292567 0.05232246
 0.05852445 0.05707805 0.04294277 0.00522677 0.0113411  0.00126012
 0.01180132 0.00356121 0.15793165]
```

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.

   My printed output:

```
[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.

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

then

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

$$= \ln((\theta_{1,y})^{x_1} * (\theta_{2,y})^{x_2} * ...)$$

$$= \ln((\theta_{1,y})^{x_1}) + \ln((\theta_{2,y})^{x_2}) + ...$$

$$= x_1 \ln(\theta_{1,y}) + x_2 \ln(\theta_{2,y}) + ...$$

So

$$\hat{p}(x|y) = e^{x_1 \ln(\theta_{1,y}) + x_2 \ln(\theta_{2,y}) + ...}$$

My printed output:

```
p(x | y=e) = e^-7841.865447060635
p(x | y=s) = e^-7938.345720103334
p(x | y=j) = e^-8001.2162734346075
```

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.

From Bayes Rule:

$$P(y|x) = \frac{P(x|y) * P(y)}{P(x)}$$

$$\text{argmax} P(y|x) = \text{argmax} \frac{P(x|y) * P(y)}{P(x)} = \text{argmax} P(x|y) * P(y)$$

where $P(y) = \frac{1}{3}$ for all three languages as previously found in 2.1. Thus, the predicted class label is the language $y \in \{e, s, j\}$ that maximizes $P(x|y) * P(y)$. From my output in 2.5, it is easy to see that $p(x|y = e)$ is the highest value. Thus, the predicted class label of x is 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	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.

   This shuffling will not affect the output of the classifier at all. Naive Bayes makes the assumption of class conditional independence. This means that the classifier assumes for a given label y, the value of each item in the input (x_i) is independent of each other item's value ($x_j, j \neq i$). Mathematically,

$$\begin{aligned} P(X_1, \ldots, X_K, Y) &= P(X_1, \ldots, X_K|Y)P(Y) \\ &= \left(\prod_{k=1}^{K} P(X_k|Y)\right) P(Y) \end{aligned}$$

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)

   Assume that l is the loss function, Z_i is the output of $w_ix_i + b_i$, and A_i is the activated output $\sigma(Z_i)$. Suppose that the gradient of the loss function with respect to the output values of the layer to the right is known, i.e. $\frac{dl}{dZ_{i+1}}$ is known. Then, using the chain rule, the gradient of the loss function with respect to the weights and outputs of the previous layer can be computed as follows:

$$\frac{dl}{dZ_i} = \frac{dl}{dZ_{i+1}} \frac{dZ_{i+1}}{dA_i} \frac{dA_i}{dZ_i}$$

   By substituting $Z_i = w_ix_i + b_i$ and $A_i = \sigma(Z_i)$ and taking the gradients, we find

$$\frac{dl}{dZ_i} = \frac{dl}{dZ_{i+1}} w_i\sigma(Z_i)(1 - \sigma(Z_i))$$

   Similarly,

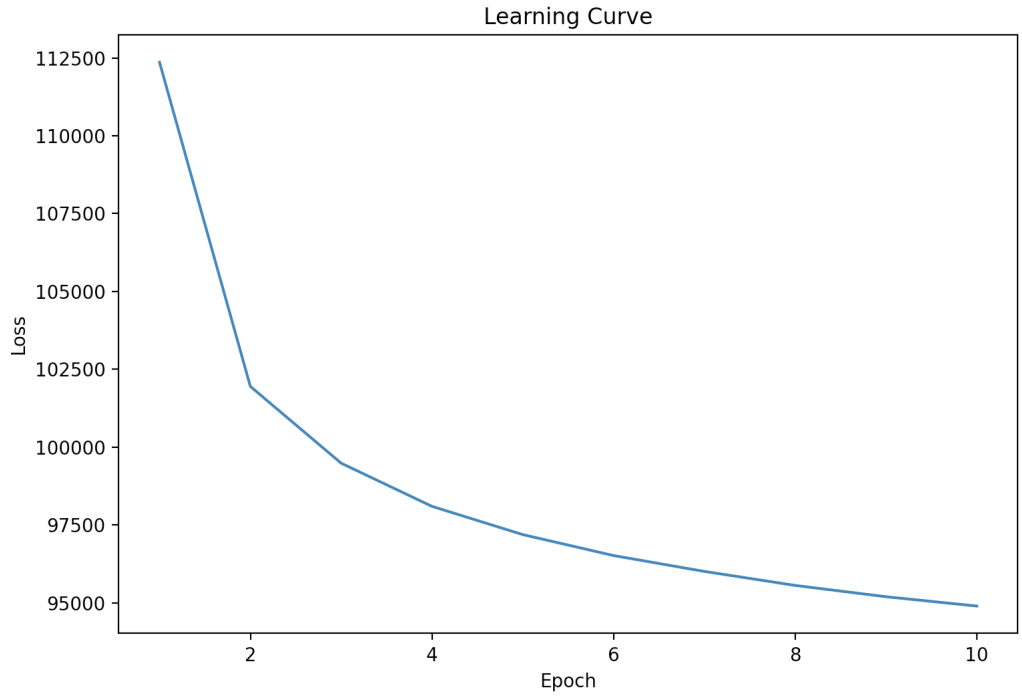
$$\begin{aligned} \frac{dl}{dw_i} &= \frac{dl}{dZ_{i+1}} \frac{dZ_{i+1}}{dw_i} \\ &= \frac{dl}{dZ_{i+1}} A_i \end{aligned}$$

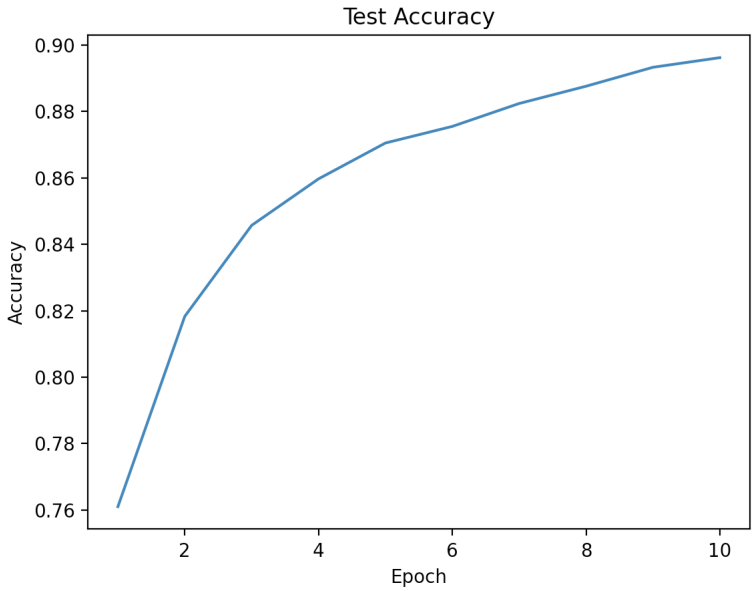
   Finally,

$$\begin{aligned} \frac{dl}{db_i} &= \frac{dl}{dZ_{i+1}} \frac{dZ_{i+1}}{db_i} \\ &= \frac{dl}{dZ_{i+1}} \end{aligned}$$

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)

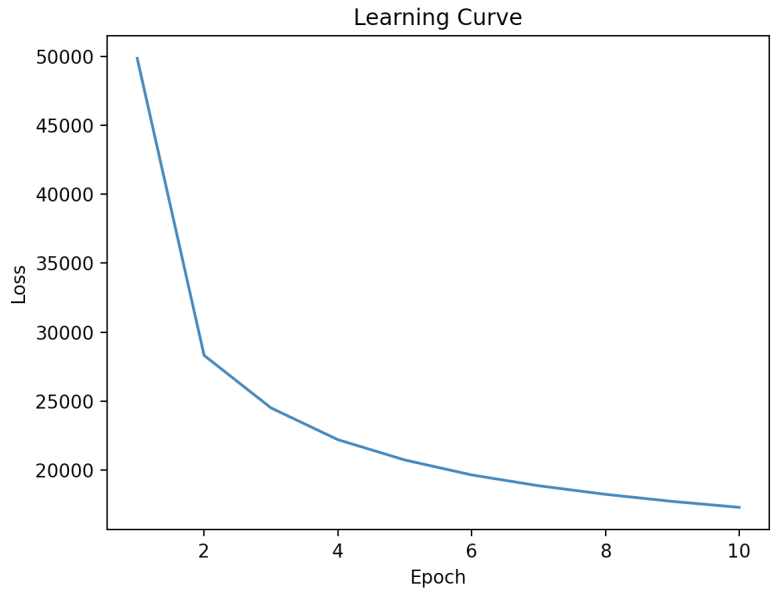
   My implementation with epochs = 10, learning rate = 0.001.

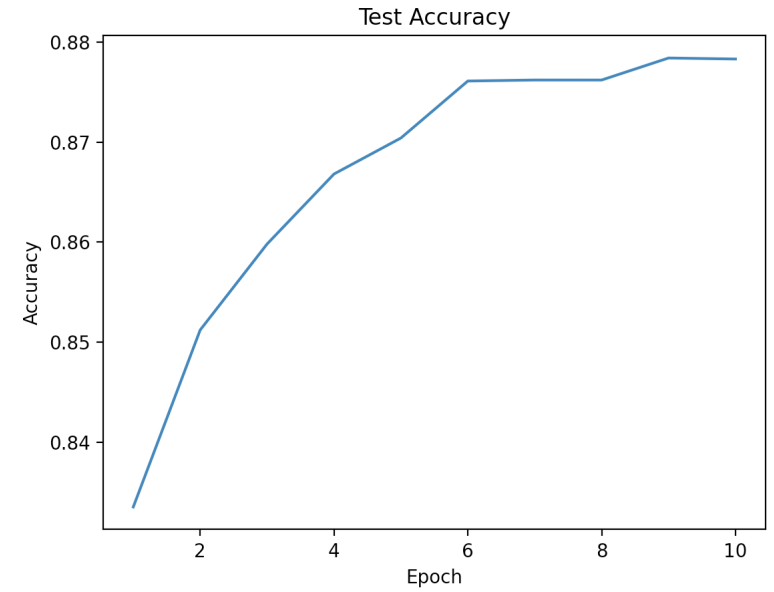




3. Implement the same network in PyTorch (or any other framework). You can use all the features of the framework e.g. auto-grad etc. Evaluate it on MNIST dataset, report test errors, and learning curve. (2 pts)

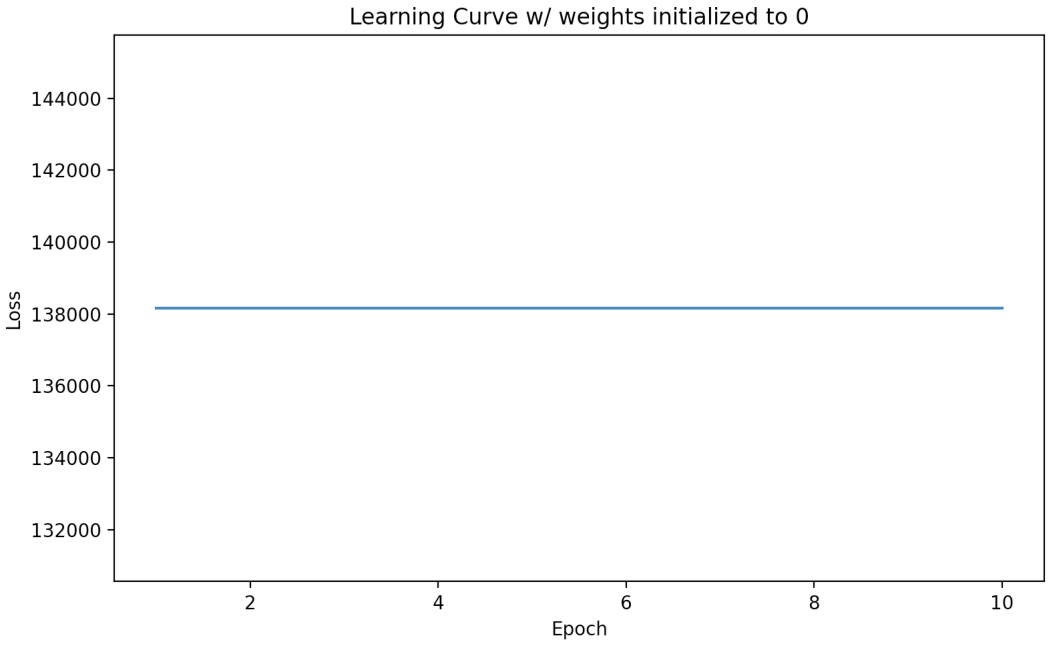
   Pytorch implementation with epochs = 10, learning rate = 0.001.

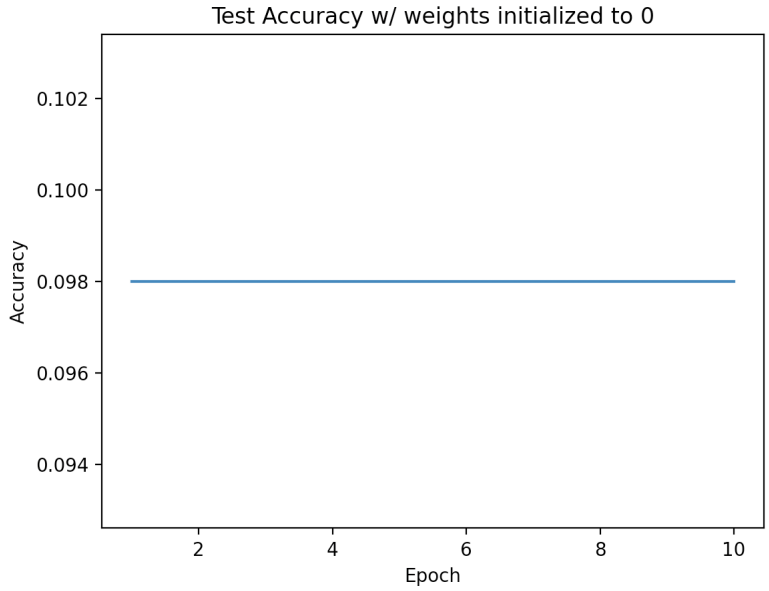




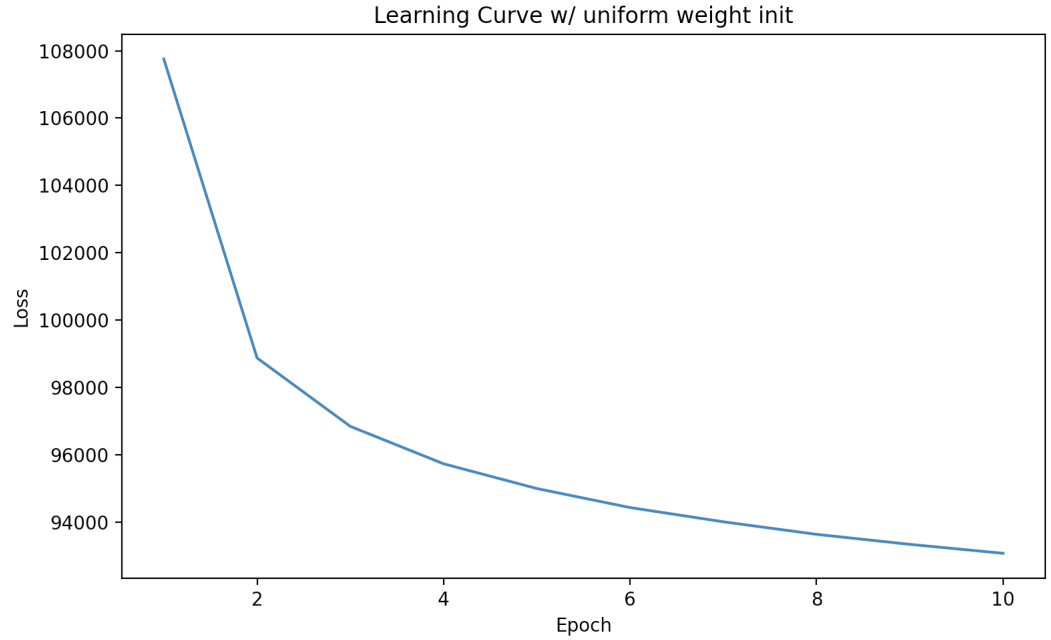
4. Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)

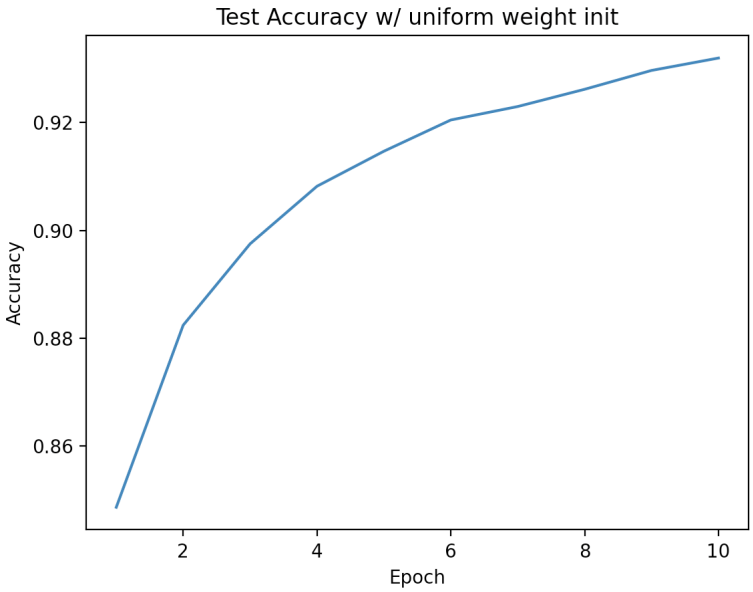
   Pytorch implementation with all weights and biases initialized to 0 (having zero initial weights prevents the network from learning. If all weights are the same, then the backpropagated errors will be the same, and consequently, all of the weights will be updated by the same amount so no learning occurs):





My custom implementation with all weights and biases initialized uniformly in between -1 and 1:





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)