HOMEWORK 4

Venkata Saikiranpatnaik Balivada
Net ID: vbalivada
Wisc ID: 9084550244

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.

Please find the code and images for the homework at https://github.com/pao214/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.
Let the outcome with the maximum likelihood be h. Our error becomes $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \mathbb{E}[\mathbb{1}_{\hat{x}=x}] = 1 - \mathbb{E}[\mathbb{1}_{x=h}] = 1 - P(x = h) = \boxed{1 - \theta_h}$.
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)
$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \mathbb{E}[\mathbb{1}_{\hat{x}=x}] = 1$. Now, $\hat{x}$ is not a constant and is generated independently of the true x. So, by independence assumption, we have $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \sum_{i=1}^{k} \mathbb{E}[\mathbb{1}_{\hat{x}=i}]\mathbb{E}[\mathbb{1}_{x=i}] = 1 - \sum_{i=1}^{k} \theta_i \theta_i = \boxed{1 - \sum_{i=1}^{k} \theta_i^2}$.

2 Best Prediction 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 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 $\hat{x}$.
For prediction $\hat{x} = j$, the error is $\mathbb{E}[c_{ij}\,\mathbb{1}_{x=i}] = \sum_{i=1}^{k} c_{ij}\theta_i$ since $c_{jj} = 0$. We pick the prediction that gives the minimum error, i.e. $\hat{x} = argmin_{j \in [k]} \sum_{i=1}^{k} c_{ij}\theta_i$. These errors correspond to columns of $\theta^T C$ where C is the matrix c_{ij}. We pick the minimal column, i.e. $\hat{x} = \arg\min \sum_{i=1}^{k} c_{ij}\theta_i$.

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.)

   Formula for prior class probability $\hat{p}(y = lang) = \frac{\mathbb{1}_{y=lang}+\alpha}{\sum_{lang}\mathbb{1}_{y=lang}+3\alpha}$, where $\alpha = \frac{1}{2}$. Plugging in the values, we have $\hat{p}(y = e) = \frac{10+\frac{1}{2}}{30+\frac{3}{2}} = \boxed{\frac{1}{3}}$. Similarly, $\hat{p}(y = j) = \frac{1}{3}$ and $\hat{p}(y = s) = \frac{1}{3}$.

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 for $\hat{p}(c_i \mid y = e) = \frac{\sum_j \mathbb{1}_{char_j=c_i}+\alpha}{\sum_j \mathbb{1}_{char_j}+27\alpha}$, where $\alpha = \frac{1}{2}$. See table 1 for class conditional probabilities computed using this formula.

character	$p(c \mid e)$
a	0.060
b	0.011
c	0.022
d	0.022
e	0.105
f	0.019
g	0.017
h	0.047
i	0.055
j	0.001
k	0.004
l	0.029
m	0.021
n	0.058
o	0.065
p	0.017
q	0.001
r	0.054
s	0.066
t	0.080
u	0.027
v	0.009
w	0.016
x	0.001
y	0.014
z	0.001
SPACE	0.178

Table 1: Class conditional probability for each character in English Documents

3. Print θ_j, θ_s and include in final report the class conditional probabilities for Japanese and Spanish.

See table 2 for the distribution of characters in Japanese documents and table 3 for character distribution in Spanish documents.

character	$p(c \mid j)$
a	0.134
b	0.011
c	0.006
d	0.018
e	0.061
f	0.004
g	0.014
h	0.032
i	0.099
j	0.002
k	0.058
l	0.001
m	0.041
n	0.058
o	0.093
p	0.001
q	0.000
r	0.044
s	0.043
t	0.058
u	0.072
v	0.000
w	0.020
x	0.000
y	0.014
z	0.008
SPACE	0.107

Table 2: Class conditional probability for each character in Japanese Documents

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.

Find the character counts in table 4.

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 class conditional likelihood values for e10.txt are $p(x|y = e) = \exp(-7831.53), p(x|y = j) = \exp(-8787.15), p(x|y = s) = \exp(-8458.14)$.

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.

Class aposteriori for e10.txt, $p(y = e|x) = \frac{p(x|y=e) \times p(e)}{p(x)} = \frac{\frac{1}{3}\exp(-7831.53)}{\frac{1}{3}\exp(-7831.53)+\frac{1}{3}\exp(-8787.15)+\frac{1}{3}\exp(-8458.14)} = \frac{\exp(-7831.53)}{\exp(-7831.53)+\exp(-8787.15)+\exp(-8458.14)}, p(y = j|x) = \frac{\exp(-8787.15)}{\exp(-7831.53)+\exp(-8787.15)+\exp(-8458.14)}, p(y =$

character	$p(c \mid s)$
a	0.105
b	0.008
c	0.038
d	0.040
e	0.114
f	0.009
g	0.007
h	0.005
i	0.050
j	0.007
k	0.000
l	0.053
m	0.026
n	0.054
o	0.073
p	0.024
q	0.008
r	0.059
s	0.066
t	0.036
u	0.034
v	0.006
w	0.000
x	0.003
y	0.008
z	0.003
SPACE	0.166

Table 3: Class conditional probability for each character in Spanish Documents

$s|x) = \frac{\exp(-8458.14)}{\exp(-7831.53)+\exp(-8787.15)+\exp(-8458.14)}$. The class with maximum aposteriori is our prediction, hence the predicted class is $\boxed{English}$.

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.

   See table 5 for the confusion matrix. Our model is a perfect predictor for the given test dataset.

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 naive assumption means that the occurrence of one character is independent of any other character given the document class. This independence also includes the relative ordering between various characters and hence we give up the ordering information by constructing a bag-of-characters counts vector for each character for each document. Thus, scrambling the characters does $\boxed{NOT}$ affect our model.

character	character count
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
SPACE	492

Table 4: Character count for the test document e10.txt

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

Table 5: Confusion matrix for English, Japanese, Spanish texts 10-19.txt

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,

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

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

   Let's define the following symbols in forward propagation for convenience.

$$
\begin{aligned}
N &= \text{batch dimension} \\
X &= \text{each row is one data point} \\
lin_1 &= X \times W_1^T + b_1, \text{ Hidden Layer Linear Operation} \\
h_1 &= \sigma(lin_1) \\
out &= h_1 \times W_2^T + b_2, \text{ Output Layer} \\
f &= softmax(out) \\
loss &= cross\_entropy(f)
\end{aligned}
$$

We now do a backward pass and start with loss derivative w.r.t. the output layer. Say, the true label is l, then the loss as a function of output is,

$$
\begin{aligned}
loss &= -\log(\frac{\exp(out_l)}{\sum_i \exp(out_i)}) \\
&= \log(\sum_i \exp(out_i)) - \log(\exp(out_l)) \\
&= \log(\sum_i \exp(out_i)) - out_l
\end{aligned}
$$

Now, we differentiate with out_j for arbitrary j,

$$
\begin{aligned}
\nabla_j loss &= \frac{1}{\sum_i \exp(out_i)} \times \exp(out_j) - \mathbb{1}_{j=l} \\
&= \frac{\exp(out_j)}{\sum_i \exp(out_i)} - \mathbb{1}_{j=l} \\
&= f(j) - y(j)
\end{aligned}
$$

If we reduce loss using mean across the batch dimension, we also divide the gradient by N. Therefore, the derivative w.r.t. the output layer is $\nabla_{out} loss = \boxed{\frac{f-y}{N}}$.

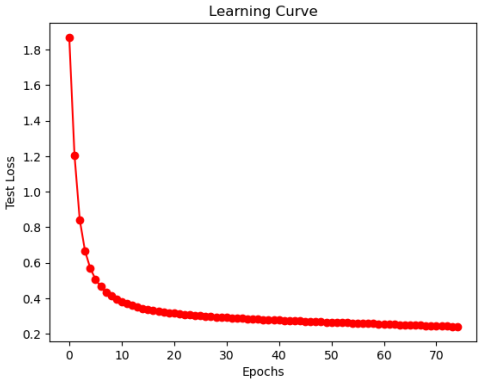
Now, we compute the weight gradients for the output layer. The formula for output is $out_j = \sum_i W_2(i,j)h_1(i) + b_2(j)$. Hence, the gradient is $out(j)h_1(i)$, making the gradient $\nabla_{W_2} loss = \boxed{\nabla_{out} loss \times h_1^T}$. The gradient w.r.t. bias is simply the gradient of the output, i.e. $\nabla_{b_2} loss = \boxed{\nabla_{out} loss}$. When processed in a batch, we simply sum the gradients across the batch dimension for each of these parameters.

Finally, let's discuss the hidden layer weights. First, the gradient w.r.t. the hidden layer activations is simply $\nabla_{h(i)} loss = \sum_j out(j) W_2(i,j)$. Then, we propagate the gradient through the sigmoid non-linearity, i.e. $\nabla_{lin_1(i)} loss = \nabla_{h_1(i)} \times h_1(i) \times (1 - h_1(i))$. From this, we can simply compute the gradient to be $\nabla_{W_1} loss = \boxed{\nabla_{lin_1} loss \times X}$ and $\nabla_{b_1} loss = \boxed{\nabla_{lin_1} loss}$. As usual, we sum these gradients along the batch dimension.

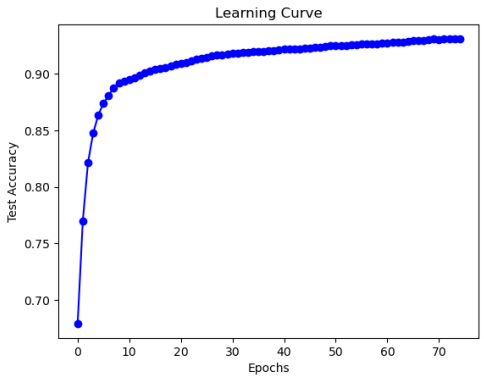
We now have all the gradients necessary for (stochastic) gradient descent.

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)

   You can find the code for this in the provided GitHub link at the start of the report.



(a) Learning Curve showing loss as training progresses for 75 epochs



(b) Learning Curve showing accuracy as training progresses for 75 epochs

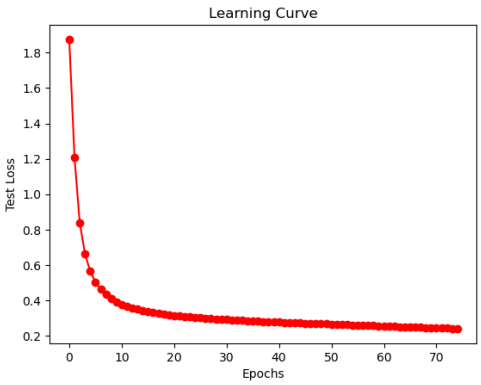
Figure 1: Learning Curves for manual gradients

For training, we used a learning rate of 0.01, a batch size of $N = 64$, and trained for 75 epochs in total.

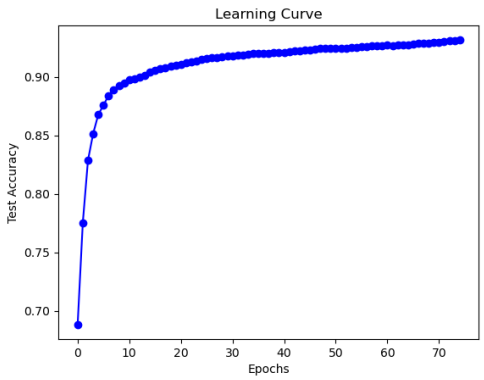
The network is the same as described in the question and it achieves a test error of 0.07 or 7%. The learning curves over the course of training are shown in the figure 1.

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)

The hyperparameters are the same as before. The test error is similar at $0.07 = 7\%$. See figure 2 for loss and accuracy.



(a) Learning Curve showing loss as training progresses for 75 epochs



(b) Learning Curve showing accuracy as training progresses for 75 epochs

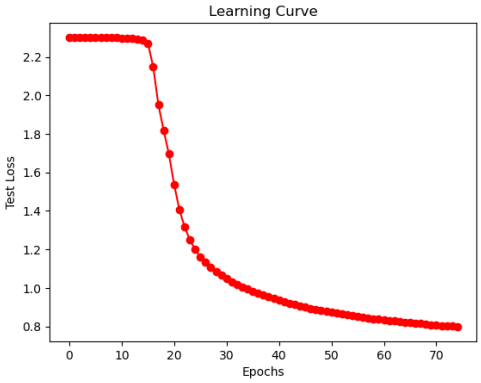
Figure 2: Learning Curves for autograd

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)

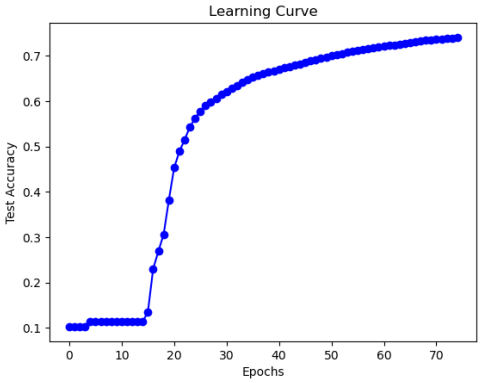
We report test error and learning curves for manual gradients case. The plots are very similar for the autograd case too.

When weights are initialized to zero, the test error is around $0.26 = 26\%$. See figure 3 for learning curves.

When weights are initialized uniformly randomly between $(-1, 1)$, the test error is back to $0.07 = 7\%$. See figure 4 for the learning curves.

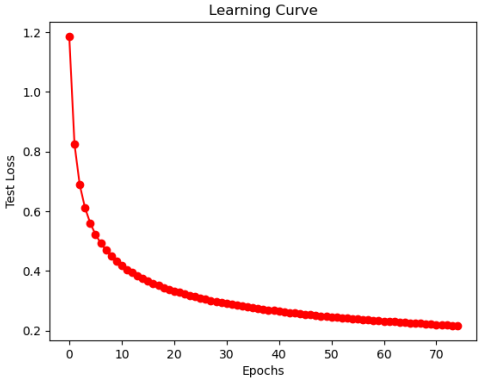


(a) Learning Curve showing loss as training progresses for 75 epochs

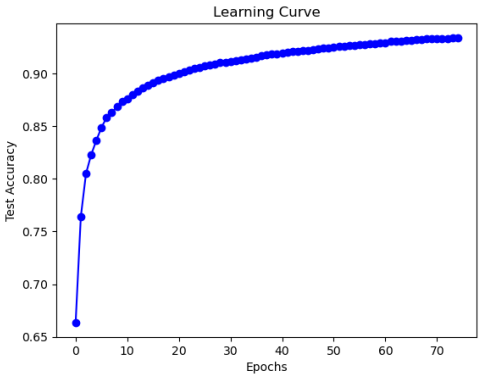


(b) Learning Curve showing accuracy as training progresses for 75 epochs

Figure 3: Learning Curves for zero-initialized weights



(a) Learning Curve showing loss as training progresses for 75 epochs



(b) Learning Curve showing accuracy as training progresses for 75 epochs

Figure 4: Learning Curves for random initialized weights

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