HOMEWORK 4

Devansh Goenka
Student ID: 908 335 1354

Answers

Answer 1

We know that :
$\mathbb{E}[\mathbf{1}[x = \hat{x}]] = 1 * P(x \neq \hat{x}) + 0 * P(x = \hat{x}) = P(x \neq \hat{x})$
Also, $P(x \neq \hat{x}) = 1 - P(x = \hat{x})$.
Hence, $\mathbb{E}[\mathbf{1}[x = \hat{x}]] = 1 - P(x = \hat{x})$.

Now, if we use the first strategy, we know that:
$P(x = \hat{x}) = \theta_{max}$, the value with the highest probability, and hence:
$\mathbb{E}[\mathbf{1}[x = \hat{x}]] = 1 - \theta_{max}$.

Now, if we use the second strategy, since the randomness is independent, for the 2 variables to be the same, we must multiply the probability of their occurrences:
$P(\hat{x} = x) = \sum_{i=1}^{k} P(x = x_i) * P(x = x_i)$
And we know simply, $P(x = x_i) = \theta_i$
Therefore, $\mathbb{E}[\mathbf{1}[x = \hat{x}]] = 1 - \sum_{i=1}^{k} P(x = x_i) * P(x = x_i) = 1 - \sum_{i=1}^{k} \theta_i^2$.

Since the values of θ is in $(0, 1)$ and we want to minimize the expected loss, we can safely say that the first strategy is better than the second strategy.

Answer 2

We can say:
$\mathbf{E}[c_{x\hat{x}}] = \sum_{i=1}^{k} \sum_{j=1}^{k} P(x = x_i) * P(\hat{x} = x_j) * c_{ij}$
We know: $P(x = x_i) = \theta_i$
Thus, $\mathbf{E}[c_{x\hat{x}}] = \sum_{i=1}^{k} \sum_{j=1}^{k} \theta_i * P(\hat{x} = x_j) * c_{ij}$
Here, we cannot simply select the outcome with highest probability as the expected loss depends on the conditional cost.

Thus, we can see that when we select $\hat{x}$ such that this expectation is minimized.

Answer 3

(i) : If we knew the distribution that the adversary is drawing the samples from, we could just predict the mean (μ) of the distribution at every round t.
This would imply that we end up paying :
$\sum_{t=1}^{T} (x_t - \mu)^2$ for the T timesteps in expecation which can be written as $\sigma^2 T$.

(ii) : Given that we arrived at the optimal strategy in the previous situation, when we knew the distribution. Now, we can freeze that as our optimal solution and then try to minimize the expectation of the chosen strategy w.r.t. the optimal one.
More formally, we want to minimize:

$$\mathbb{E}[\sum_{t=1}^{T}(x_t - y)^2] - \sigma^2 T$$

We can then select the strategy that converges to the optimal solution as T tends to infinity.

Answer 4.1

Since the number of training samples are the same for all 3 classes *y=e, j, s*, the prior probability is the same for all 3 classes.
The formula for the prior probability for all 3 classes is:

$$\hat{p}(y = c_k) = \frac{10 + 0.5}{30 + 3 * 0.5} = 0.33$$

Therefore:

$$\hat{p}(y = e) = 0.33$$

$$\hat{p}(y = j) = 0.33$$

$$\hat{p}(y = s) = 0.33$$

Answer 4.2

To estimate the class-conditional probability for each character, let:

$$\theta_e = \begin{bmatrix} \theta_{<space>,e} \\ \theta_{a,e} \\ \theta_{b,e} \\ \vdots \\ \theta_{z,e} \end{bmatrix}$$

Then, from our data, we have:

$$\theta_e = \begin{bmatrix} 0.179 \\ 0.060 \\ 0.011 \\ 0.021 \\ 0.021 \\ 0.105 \\ 0.018 \\ 0.017 \\ 0.047 \\ 0.055 \\ 0.001 \\ 0.003 \\ 0.028 \\ 0.020 \\ 0.057 \\ 0.064 \\ 0.016 \\ 0.001 \\ 0.053 \\ 0.066 \\ 0.080 \\ 0.026 \\ 0.009 \\ 0.015 \\ 0.001 \\ 0.013 \\ 0.001 \end{bmatrix}$$

Answer 4.3

The class-conditional probabilities for Japanese and Spanish are:

$$\theta_j = \begin{bmatrix} 0.123 \\ 0.131 \\ 0.010 \\ 0.005 \\ 0.017 \\ 0.060 \\ 0.003 \\ 0.014 \\ 0.031 \\ 0.097 \\ 0.002 \\ 0.057 \\ 0.001 \\ 0.039 \\ 0.057 \\ 0.056 \\ 0.091 \\ 0.001 \\ 0.001 \\ 0.042 \\ 0.042 \\ 0.056 \\ 0.070 \\ 0.002 \\ 0.019 \\ 0.014 \\ 0.007 \end{bmatrix}$$

$$\theta_s = \begin{bmatrix} 0.168 \\ 0.104 \\ 0.008 \\ 0.037 \\ 0.039 \\ 0.113 \\ 0.008 \\ 0.007 \\ 0.004 \\ 0.049 \\ 0.006 \\ 0.001 \\ 0.052 \\ 0.025 \\ 0.054 \\ 0.072 \\ 0.024 \\ 0.007 \\ 0.059 \\ 0.065 \\ 0.035 \\ 0.033 \\ 0.005 \\ 0.000 \\ 0.002 \\ 0.007 \\ 0.002 \end{bmatrix}$$

Answer 4.4

For the given text document, here is the bag-of-characters vector x:

$$x = \begin{bmatrix} x_{<space>} \\ x_a \\ x_b \\ \vdots \\ x_z \end{bmatrix} = \begin{bmatrix} 498 \\ 164 \\ 32 \\ 53 \\ 57 \\ 311 \\ 55 \\ 51 \\ 140 \\ 140 \\ 3 \\ 6 \\ 85 \\ 64 \\ 139 \\ 182 \\ 53 \\ 3 \\ 141 \\ 186 \\ 225 \\ 65 \\ 31 \\ 47 \\ 4 \\ 38 \\ 2 \end{bmatrix}$$

Answer 4.5

For the given vector x, we will be using log probability as the actual values can possibly underflow. Here are the values:

$$log\ \hat{p}(x|y = e) = -7841.865$$
$$log\ \hat{p}(x|y = j) = -8749.114$$
$$log\ \hat{p}(x|y = s) = -8467.282$$

Answer 4.6

For the x of e10.txt, here are the posterior log-probabilities for all 3 classes:

$$log\ \hat{p}(y = e|x) = -2613.955$$

$$log\ \hat{p}(y = j|x) = -2916.371$$

$$log\ \hat{p}(y = s|x) = -2822.427$$

Moreover, since the posterior probability of $y = e$ is the highest, the model predicts that this document belongs to the English class.

Answer 4.7

After testing the trained Naive Bayes model on the given test set, here is the confusion matrix for all 3 classes:

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

Thus, our model has learned to classify the test set with a 100% accuracy.

Answer 4.8

Even after shuffling a text document's characters, the posterior probability of it remained unchanged. This is because Naive Bayes assumes conditional independence of the input features with respect to the output label. Basically, $\hat{p}(x_i|y)$ and $\hat{p}(x_j|y)$ are assumed to be independent. Since our input features are characters, their likelihood does not change after random shuffling as their empirical count still remains the same, and since they can be factored into the product of independent likelihoods of all features, the posterior probability remains unchanged.

Answer 5.1

To start our backpropagation process, let us first formally write down the cross-entropy loss function:

$$\ell(y, s) = -\sum_{i=1}^{k} y_i\ log\ (s_i)$$

Here, s is the output of the final layer and s_i is the i'th output of the softmax activation at the final layer and y represents a one-hot encoded vector of the ground truth with $y_i \in [0, 1]$.
Now, we need the derivative of this loss w.r.t to the weighted input z to the final softmax layer.

$$\frac{\partial \ell}{\partial z_j} = -\frac{\partial}{\partial z_j}\sum_{i=1}^{k} y_i . \frac{\partial}{\partial z_j}(\log s_i).$$

We know that $\log s_i = z_i - \log(\sum_{j=1}^{k} e^{z_j})$. After performing standard calculus, it can be seen that: $\frac{\partial}{\partial z_j}(\log s_i) = \mathbf{1}(i = j) - s_j$
After plugging this in the previous equation, we get:

$$\frac{\partial \ell}{\partial z_j} = -\frac{\partial}{\partial z_j}\sum_{i=1}^{k} y_i . \mathbf{1}(i = j) - s_j.$$

Which can then be re-written after some operations as:

$$\frac{\partial \ell}{\partial z_j} = s_j . \sum_{i=1}^{k} y_i - y_j$$

Since y is one hot encoded, the term $\sum_{i=1}^{k} y_i = 1$ and therefore we finally get :

$$\frac{\partial \ell}{\partial z_j} = s_j - y_j$$

Which can then be written in vector notation as :

$$\frac{\partial \ell}{\partial z} = s - y$$

Now that we found the gradient of the loss w.r.t to the input to the final layer, we can apply chain rule to calculate the gradient w.r.t to the weights of the second layer W_2.

$$\frac{\partial \ell}{\partial W_2} = \frac{\partial \ell}{\partial z} * \frac{\partial z}{\partial W_2}$$

And since $z = W_2 a$, where a is the output of the first layer, we simply get:

$$\frac{\partial \ell}{\partial W_2} = (s - y)^T a$$

Thus, we have our gradient w.r.t to the weight matrix W_2 and we can change the weights after scaling with the learning rate.
Now, we must continue the backpropagation to compute the gradient for W_1 as well.
So,

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial \ell}{\partial z} * \frac{\partial z}{\partial a} * \frac{\partial a}{\partial z_1} * \frac{\partial z_1}{\partial W_1}$$

Here, z_1 is the weighted input to the first hidden layer and a is the output of that layer. Now:
$\frac{\partial z}{\partial a} = W_2^T$ and
$\frac{\partial a}{\partial z_1} = \sigma'(z_1) = \sigma(z_1) * \sigma(1 - z_1)$
And finally, $\frac{\partial z_1}{\partial W_1} = X$
Where X is the input to the network.
Therefore,

$$\frac{\partial \ell}{\partial W_1} = (y - s) W_2^T \sigma(z_1) * \sigma(1 - z_1) X$$

Thus, we computed the gradient for both our W_1 and W_2 weight matrices and we can go ahead and apply gradient descent to update these weights.

Answer 5.2

After implementing the above described network in numpy, here was the test error and learning curve after training the model on the MNIST dataset:

Model Hyperparameters:
Learning rate (λ) = 0.01
Epochs = 20
Batch size = 32

The weights were initialized by sampling randomly from a normal distribution.
The reported cross entropy loss on the test set containing 10k test images from the MNIST dataset is: 0.463076. The accuracy on the test set at this loss was 89.35%.
The learning curve is:

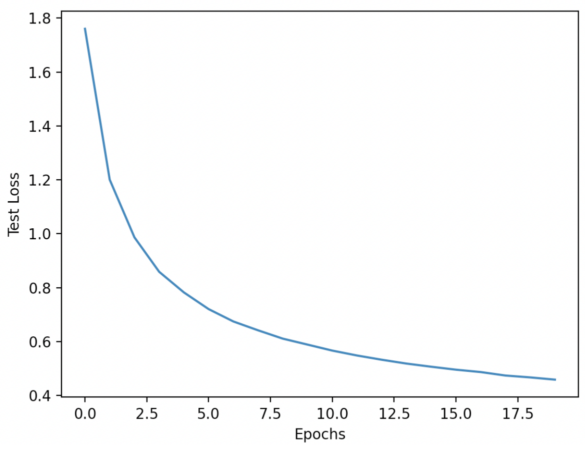

Figure 1: Learning curve with numpy and linear algebra implementation of network

Answer 5.3

The same model when implemented in PyTorch with the same hyperparameters, the reported metrics are:
Cross entropy loss on the same test set = 0.219881
The learning curve is:

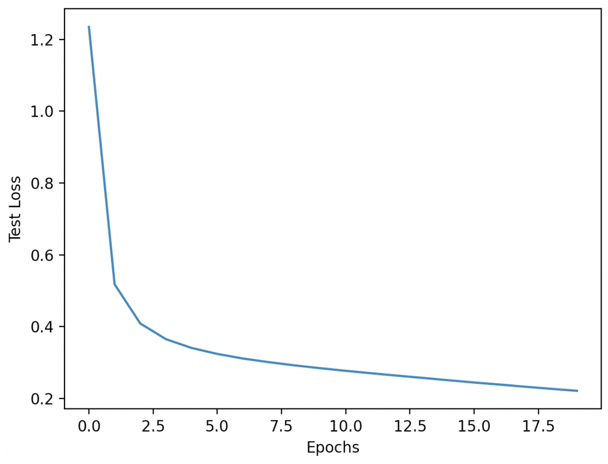

Figure 2: Learning curve with PyTorch implementation

Answer 5.4

Using the numpy implementation of the network, the two mentioned scenarios was tried out. Here are the results for both of them:
(a) : When all weights are initialized to zero, it was seen that the loss decays slowly, with it being stuck in a local minima for some epochs before gradually starting to move downwards. The overall test loss was significantly higher than the test loss for the random initialization case.

Test Loss = 2.0145
The learning curve is:

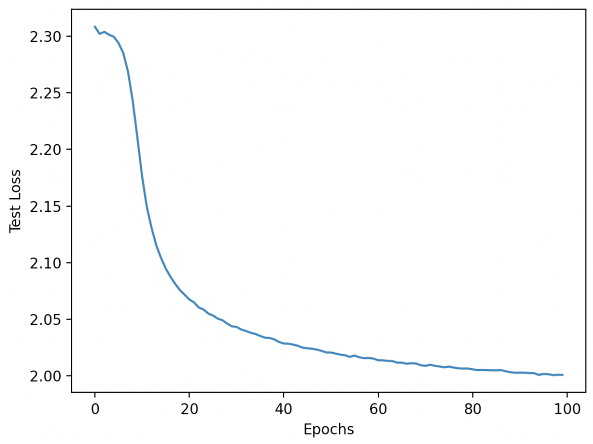

Figure 3: Learning curve when weights are initialized to zero

(b) : When all weights are initialized randomly to values between -1 and 1, it was seen that the loss decays much faster than when compared to the zero initialization case. However, it must be said that it still decays slower than the PyTorch implementation.

Test Loss = 0.304283
The learning curve is:

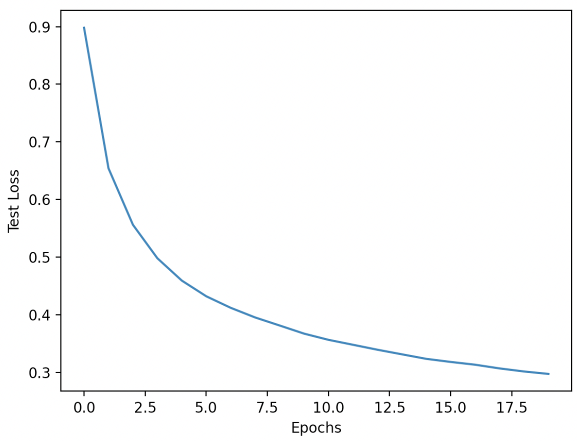

Figure 4: Learning curve when weights are initialized to between -1 and 1