HOMEWORK 4

Daniel Szabo
9074762189

1. (a) The 0-1 loss is

$$\mathbb{E}[\mathbb{1}[\hat{x} \neq x]] = \mathbb{P}(\hat{x} \neq x) = 1 - \mathbb{P}(\hat{x} = x) = 1 - \theta_x$$

   (b) The 0-1 loss in this case is

$$\begin{aligned}\mathbb{E}[\mathbb{1}[\hat{x} \neq x]] = \mathbb{P}(\hat{x} \neq x) &= 1 - \sum_k \mathbb{P}(\hat{x} = x, x = k) \\ &= 1 - \sum_k \mathbb{P}(\hat{x} = x)\mathbb{P}(x = k) = 1 - \sum_k \mathbb{P}(\hat{x} = k)^2 \\ &= 1 - \sum_k \theta_k^2 = 1 - \|\boldsymbol{\theta}\|_2^2\end{aligned}$$

2. Under this, we want to minimize

$$\mathbb{E}[c_{x\hat{x}}] = \sum_i \sum_j c_{ij}\mathbb{P}(\hat{x} = i)\mathbb{P}(x = j) = \sum_i \sum_j c_{ij}\theta_i \mathbb{P}(\hat{x} = j) = \sum_j \mathbb{P}(\hat{x} = j) \sum_i c_{ij}\theta_i$$

   Let C be the loss matrix with entries c_{ij}, and $\hat{\boldsymbol{\theta}}$ the parameter estimator. Then

$$\mathbb{E}[c_{x\hat{x}}] = \hat{\boldsymbol{\theta}}^T C\boldsymbol{\theta},$$

   so the $\hat{\boldsymbol{\theta}}$ that minimizes this is just $C\boldsymbol{\theta}$.

3. (a) Let the distribution of y have pdf ρ.

$$\begin{aligned}\mathbb{E}[(x_t - y_t)^2] &= \int_0^1 \int_0^1 (x - y)^2 \rho(x)\rho(y)dxdy \\ &= \int_0^1 \int_0^1 ((x - \mu) - (y - \mu))^2 \rho(x)\rho(y)dxdy \\ &= \int_0^1 (x - \mu)^2 \rho(x)dx + 2\int_0^1 \int_0^1 (x - \mu)(y - \mu)\rho(x)\rho(y)dxdy + \int_0^1 (y - \mu)^2 \rho(y)dy \\ &= \sigma^2 + \int_0^1 (x - \mu)\rho(x) \int_0^1 (y - \mu)\rho(y)dy + \int_0^1 (x - \mu)^2 \rho(x)dx \\ &= \sigma^2 + \int_0^1 (x - \mu)^2 \rho(x)dx\end{aligned}$$

   This is minimized at $x_t = \mu$ deterministically, as it is strictly greater than zero and achieves zero at $x_t = \mu$. Thus the optimal payment for T rounds is just $T\sigma^2$.

   (b) There are many possible such measurements. We could measure according to the sample variance at time t, $\hat{\sigma}_t^2 = \frac{1}{t-1}\sum_{i=1}^t (y_t - \hat{\mu}_t)^2$, where $\hat{\mu}_t = \frac{1}{t}\sum_{i=1}^t y_t$ is the sample mean. The quantity to compare to would then be $T\hat{\sigma}_t^2$.

4. (a) Each language only has 10 samples in this case, so the prior probabilities are just each $1/3$. The formula would be the same as what's given, namely

$$\hat{p}_\alpha(y = c_k) = \frac{\sum_{i=1}^{30} \mathbb{1}[y^{(i)} = c_k] + 1/2}{30 + 3/2} = \frac{10 + 1/2}{30 + 3/2} = \frac{1}{3}.$$

   I believe a more appropriate prior for this problem would be to sum over the number of characters in each language, but by the definitions given this is our prior.

(b) The class conditional parametersare given by the formula in the problem, namely

$$\hat{p}_\alpha(c_s|y=e) = \frac{\sum_{i=1}^{30}\sum_{j=1}^{M_i}\mathbb{1}[x_j^{(i)}=c_s, y^{(i)}=e]+1/2}{\sum_{s'\in S}\sum_{i=1}^{30}\sum_{j=1}^{M_i}\mathbb{1}[x_j^{(i)}=c_{s'}, y^{(i)}=e]+27/2}.$$

The outputs for θ_e, θ_j, and θ_s are
θ_e = [0.06030001 0.01115931 0.02155701 0.0220206 0.10559952 0.01897414 0.01751714 0.04731945 0.05553164 0.00142389 0.00374185 0.0290407 0.02056359 0.05804828 0.06460479 0.01678864 0.00056293 0.05394218 0.0663267 0.08030067 0.02672274 0.00930494 0.01553032 0.00115898 0.01387463 0.00062916 0.17745621]
θ_j = [1.33652313e-01 1.10225058e-02 5.56441609e-03 1.74729754e-02 6.10668084e-02 3.93407762e-03 1.42122984e-02 3.22169059e-02 9.84228247e-02 2.37462343e-03 5.82314372e-02 1.45312777e-03 4.03685983e-02 5.75225944e-02 9.24685451e-02 8.86053518e-04 1.06326422e-04 4.34166224e-02 4.27786638e-02 5.78061315e-02 7.16285664e-02 2.48094985e-04 2.00248095e-02 3.54421407e-05 1.43540670e-02 7.83271310e-03 1.10898458e-01]
θ_s = [1.04942283e-01 8.26292823e-03 3.76628601e-02 3.98910655e-02 1.14226472e-01 8.63429579e-03 7.21072014e-03 4.54925262e-03 5.00417789e-02 6.65366880e-03 2.78525671e-04 5.31365085e-02 2.59028874e-02 5.43744004e-02 7.27570947e-02 2.43555225e-02 7.70587689e-03 5.95116517e-02 6.60105840e-02 3.57441278e-02 3.38253954e-02 5.91093368e-03 9.28418903e-05 2.50673104e-03 7.89156067e-03 2.69241482e-03 1.65227617e-01]

(c) (Jumping to 4.) The bag-of-characters vector is
$[164, 32, 53, 57, 311, 55, 51, 140, 140, 3, 6, 85, 64, 139, 182, 53, 3, 141, 186, 225, 65, 31, 47, 4, 38, 2, 489]$

(d) The output was:
$\log(\hat{p}(x|y=e)) = -300482.66167312715$
$\log(\hat{p}(x|y=j)) = -331013.25871754845$
$\log(\hat{p}(x|y=s)) = -312320.3717697917$

(e) Instead of just taking the values without dividing by $p(x)$, I normalized the values using numpy's way of adding exponents in log space. I then could take the true values, which were $p(y=e|x) = 1.0, p(y=j|x) = 0.0, p(y=s|x) = 0.0$.

(f) The confusion matrix was surprisingly

10	10	10
0	0	0
0	0	0

which just classified everything as English. As far as I can tell the main reason this happened is because our priors were uniform, but it could have been a number of other reasons as well.

(g) The prediction is still completely the same. This is because of our key (and in this case incorrect) assumption that the characters are drawn for iid random variables. Because of this, the joint probability splits into a product, which commutes and therefore is independent of the order of the characters. In reality, two neighboring characters are not independent of each other, which is hopefully why my Naïve Bayes classifier failed.

5. The derivation of the backpropagation updates are all present in the code, but I will try to succinctly write them here as well. Most importantly, $\frac{\delta L}{\delta \hat{y}}\frac{\delta \hat{y}}{\delta z_2} = \hat{y} - y$. Then taking the derivative with respect to the weights (biases are similar) we get

$$\begin{aligned}
\frac{\delta L}{\delta a_1} &= \frac{\delta L}{\delta z_2} W_2 \\
\frac{\delta L}{\delta W_2} &= \frac{\delta L}{\delta z_2} a_1 \\
\frac{\delta L}{\delta z_1} &= \frac{\delta L}{\delta a_1}\sigma'(z_1) = \frac{\delta L}{\delta a_1}\sigma(z_1)(1-\sigma(z_1)) \\
\frac{\delta L}{\delta W_1} &= \frac{\delta L}{\delta z_1} X.
\end{aligned}$$

I used a single plot to plot the learning curves of my version and the pytorch version with the same batch size of 128 and different learning rates. My version worked best with a learning rate of 0.1, while the torch

version was quickest with a learning rate of 1. Both started with random initial weights. The learning curves (torch in blue, mine in red) are in Figure 1. My code was painfully slow and caused my computer to crash a few times, so I didn't run it again to add axis labels and a title. The final accuracy on the test set for my network was

$$53.046875\%$$

and the final accuracy for the torch version was

$$91.0\%.$$

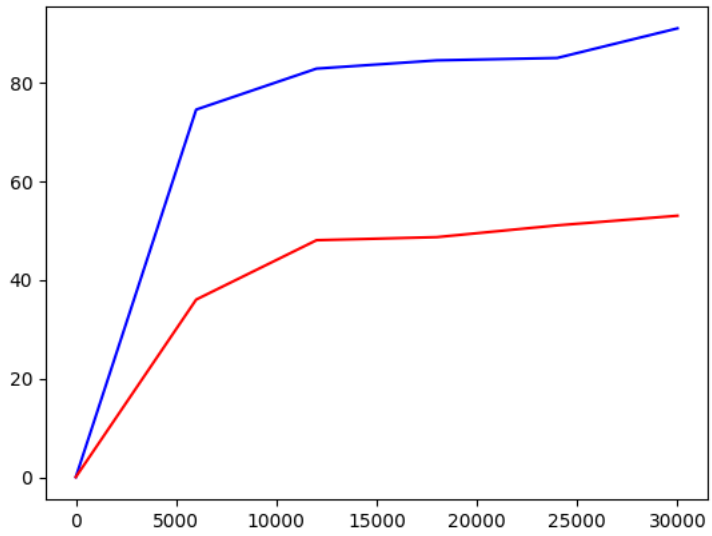

Figure 1: The learning curves for my (red) and pytorch's (blue) 2 layer neural networks on the MNIST dataset. The x axis represents the number of samples processed, and y the percentage accuracy on the validation set.

Next I took the pytorch version (which was much faster and converged better) and changed my weight initializations. For one, I initialized the weights uniformly at random between -1 and 1, and for the other at a constant 0. As I had noticed on my own implementation as well, initializing to 0 makes convergence much slower. The final accuracy for the random initialization was

$$66.4\%$$

and for the 0 initialization it was

$$68.8\%.$$

The two learning curves are shown in Figure 2.

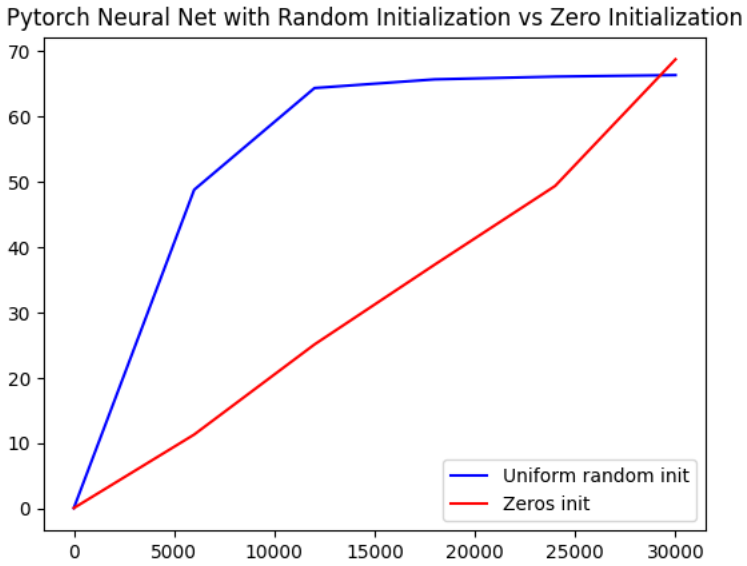

Figure 2: The learning curves for the different weight initializations.