HOMEWORK 4

Yelun Bao
ybao35@wisc.edu

1 Best Prediction Under 0-1 Loss

1. Under Strategy 1,
$$\mathbb{E}[\mathbb{1}[\hat{x} \neq x]] = \Pr[\hat{x} \neq x] = 1 - \max_i \theta_i.$$

2. Under Strategy 2,
$$\mathbb{E}[\mathbb{1}[\hat{x} \neq x]] = \sum_{i=1}^{k} \theta_i(1 - \theta_i).$$
To prove it, when minicing, we generate x_i with probability θ_i, and it is correct with probability θ_i or uncorrect probability $1 - \theta_i$.

2 Best Prediction Under Different Misclassification Losses

Let our strategy be $\hat{x} \sim$ multinomial$(p_1, p_2, \cdots, p_k)$.

$$\mathbb{E}[\mathbb{1}[\hat{x} \neq x]] = \sum_{i=1}^{k} \theta_i \sum_{j=1}^{k} p_j c_{ij} = \sum_{j=1}^{k} p_j \sum_{i=1}^{k} \theta_i c_{ij}.$$

Let $t_j = \sum_{i=1}^{k} \theta_i c_{ij}$. To minimize $\mathbb{E}[\mathbb{1}[\hat{x} \neq x]]$, we just need to find the minimal t_j and let corresponding $p_j = 1$. That is, $\hat{x} \in \arg\min_x t_x$.

3 Best Prediction Under Online Learning

1. Assume we draw our prediction x from certain distribution with mean μ_x and variance σ_x^2. The payments in T rounds in expectation is
$$\int_0^1 \int_0^1 p_x p_y (x - y)^2 dy dx = \int_0^1 p_x \left(\int_0^1 p_x p_y (x - y)^2 dy \right) dx$$
$$= \int_0^1 p_x \left(x^2 - 2\mu x + \mathbb{E}[y^2] \right) dx = \mathbb{E}[x^2] - 2\mu\mu_x + \mathbb{E}[y^2]$$
$$= \mu_x^2 + \sigma_x^2 - 2\mu\mu_x + \mu^2 + \sigma^2 = (\mu_x - \mu)^2 + \sigma_x^2 + \sigma^2.$$
To minimize it, we should let $\mu_x = \mu$ and $\sigma_x^2 = 0$. That is, always predict $x = \mu$. The payments in T rounds in expectation is σ^2.

2. In order to measure the benchmark, we can just compute the difference between our payments in T rounds in expectation and the optimal, i.e. payments in T rounds in expectation minus σ^2.

4 Language Identification with Naive Bayes

All of the answers can be found in Jupyter notebook named ”4. Language Identification with Naive Bayes”. Each question is labelled with a question number comment.

5 Simple Feed-Forward Network

1. Different from normal index subscript, to make it easier for me, suppose that

$$\mathbf{W}_1 = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & \cdots & w_{d1}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & \cdots & w_{d2}^{(1)} \\ \cdots \\ w_{1d_1}^{(1)} & w_{2d_1}^{(1)} & \cdots & w_{dd_1}^{(1)} \end{bmatrix}, \mathbf{W}_2 = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & \cdots & w_{d_1 1}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & \cdots & w_{d_1 2}^{(2)} \\ \cdots \\ w_{1k}^{(2)} & w_{2k}^{(1)} & \cdots & w_{d_1 k}^{(2)} \end{bmatrix}.$$

Then, let $z = W_2\sigma(W_1 x)$ and $h = W_1 x$. Assume the label of x is l, i.e $y_l = 1$. We can derive that

$$\frac{\partial L}{\partial w_{ij}^{(2)}} = \frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}^{(2)}} = (\hat{y}_j - \mathbb{1}(j = l))\sigma(h_i).$$

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = \sum_{t=1}^{k}(\frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial \sigma(h_j)})\sigma'(h_j)\frac{\partial h_j}{\partial w_{ij}^{(1)}} = \sigma(h_j)(1-\sigma(h_j))x_i\sum_{t=1}^{k}(\hat{y}_t - \mathbb{1}(t = l))w_{jt}^{(2)}.$$

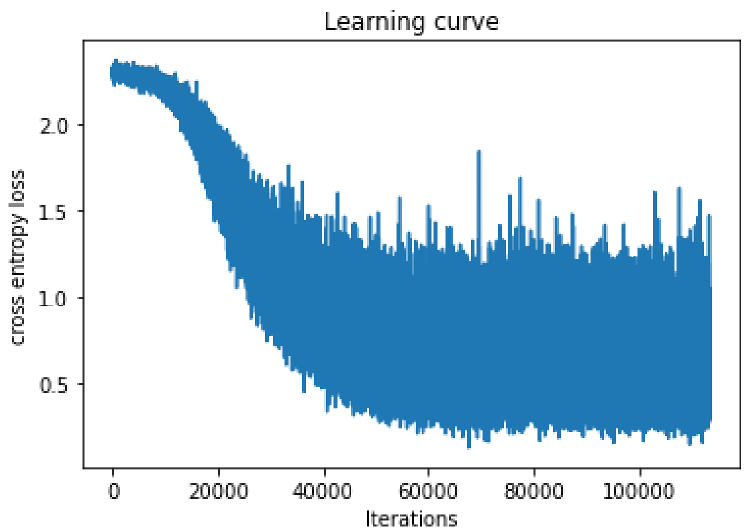
We can write the gradient as outer product.

$$\nabla_{W_2} L = \sigma(h) \otimes \hat{y} - \sigma(h) \otimes y.$$

$$\nabla_{W_1} L = (W_2^T\hat{y} - W_2[l]) \odot \sigma(h) \odot (1 - \sigma(h)) \otimes x,$$

where $\otimes$ is outer product and $\odot$ is element-wise product. Then, the weight can be backpropagation updated in this way.

2. In this part, all of the codes can be found in Jupyter notebook named "5. Simple Feed-Forward Network (Backpropagation)". We set learning rate $lr = 0.001$ and batch size as 16. Randomly initialize all weights from $[-\frac{1}{28}, \frac{1}{28}]$. Test accuracy is 0.8124 (Test error 0.1876). Learning curve is shown below:



3. In this part, all of the codes can be found in Jupyter notebook named "5. Simple Feed-Forward Network (Pytorch)". We set learning rate $lr = 0.001$ and batch size as 64. All weights are initialized by nn.Linear() default.

   Test accuracy is 0.9265 (Test error 0.0735). Learning curve is shown below:

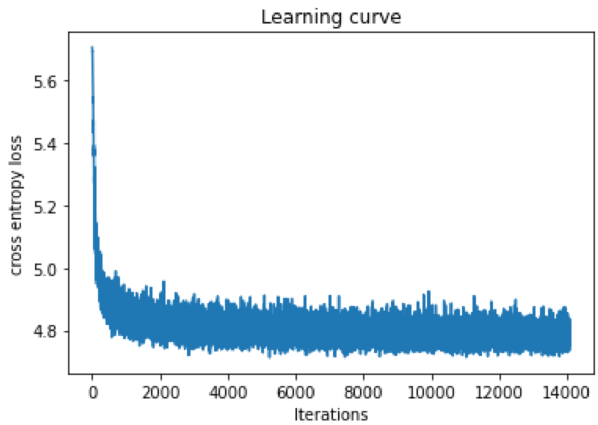


Figure 1: Question 5.3

4. In this part, all of the codes can be found in Jupyter notebook named "5. Simple Feed-Forward Network (Pytorch)". Use the same hyperparameters as the previous part. Only changing the initialization method.

   Test accuracy is 0.9245 (Test error 0.0755) for zero-initialization and 0.9228 (Test error 0.0772) for $[-1, 1]$-initialization. Learning curves are shown below: We can find that it takes us longer time if we initialize

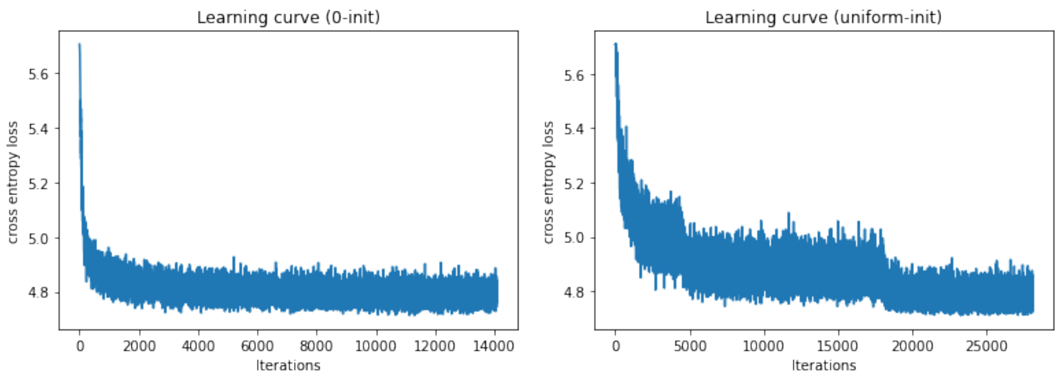


Figure 2: Question 5.4

   weights uniformly from [-1,1], while zero-initialization does make much difference compared with default initialization. We even need to extend the training epochs to let the latter training converge.