2252654 NLP Lab 6 Math Exercise

Phuong Huynh

March 2025

1 Problem 1

1.1 Forward Pass

1.1.1 Hidden Layer Activations

The activation function used is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

For the hidden neurons:

$$\begin{aligned} z_1 &= w_1 i_1 + w_2 i_2 + b_1 \\ &= (0.15)(0.05) + (0.20)(0.10) + 0.35 \\ &= 0.0075 + 0.02 + 0.35 = 0.378 \\ h_1 &= \sigma(0.378) = \frac{1}{1 + e^{-0.378}} \approx 0.593 \end{aligned}$$

$$\begin{aligned} z_2 &= w_3 i_1 + w_4 i_2 + b_1 \\ &= (0.25)(0.05) + (0.30)(0.10) + 0.35 \\ &= 0.0125 + 0.03 + 0.35 = 0.393 \\ h_2 &= \sigma(0.393) = \frac{1}{1 + e^{-0.393}} \approx 0.597 \end{aligned}$$

1.1.2 Output Layer Activations

$$\begin{aligned} z_3 &= w_5 h_1 + w_6 h_2 + b_2 \\ &= (0.40)(0.593) + (0.45)(0.597) + 0.60 \\ &= 0.237 + 0.269 + 0.60 = 1.106 \\ o_1 &= \sigma(1.106) = \frac{1}{1 + e^{-1.106}} \approx 0.751 \end{aligned}$$

$$\begin{aligned} z_4 &= w_7 h_1 + w_8 h_2 + b_2 \\ &= (0.50)(0.593) + (0.55)(0.597) + 0.60 \\ &= 0.297 + 0.328 + 0.60 = 1.225 \\ o_2 &= \sigma(1.225) = \frac{1}{1 + e^{-1.225}} \approx 0.773 \end{aligned}$$

1.2 Loss Gradient Computation

The Mean Squared Error (MSE) loss function is:

$$MSE = \frac{1}{2}\sum(o_{\text{exp}} - o_{\text{pred}})^2 \tag{2}$$

The gradient of MSE with respect to each output neuron is:

$$\frac{\partial MSE}{\partial o_i} = (o_i - o_{i,\text{exp}}) \tag{3}$$

1.3 Backward Pass

1.3.1 Output Layer Gradients

$$\begin{aligned}\delta_{o_1} &= (o_1 - o_{1,\text{exp}}) \cdot o_1 \cdot (1 - o_1) \\ &= (0.751 - 0.01) \times 0.751 \times 0.249 = 0.138\end{aligned}$$

$$\begin{aligned}\delta_{o_2} &= (o_2 - o_{2,\text{exp}}) \cdot o_2 \cdot (1 - o_2) \\ &= (0.773 - 0.99) \times 0.773 \times 0.227 = -0.038\end{aligned}$$

1.3.2 Hidden Layer Gradients

$$\begin{aligned}\delta_{h_1} &= h_1(1 - h_1)\sum_k \delta_k w_{jk} \\ &= 0.593 \times 0.407 \times [(0.138 \times 0.40) + (-0.038 \times 0.50)] \\ &= 0.009\end{aligned}$$

$$\begin{aligned}\delta_{h_2} &= h_2(1 - h_2)\sum_k \delta_k w_{jk} \\ &= 0.597 \times 0.403 \times [(0.138 \times 0.45) + (-0.038 \times 0.55)] \\ &= 0.01\end{aligned}$$

1.4 Weight and Bias Updates

Using the weight update rule:

$$w' = w - \eta \cdot \delta \cdot \text{input} \tag{4}$$

1.4.1 Output Layer Updates

$$\begin{aligned}w_5' &= 0.40 - 0.5 \times (0.138 \times 0.593) = 0.359 \\ w_6' &= 0.45 - 0.5 \times (0.138 \times 0.597) = 0.409 \\ w_7' &= 0.50 - 0.5 \times (-0.038 \times 0.593) = 0.511 \\ w_8' &= 0.55 - 0.5 \times (-0.038 \times 0.597) = 0.561 \\ b_2' &= 0.60 - 0.5 \times (0.138 + (-0.038)) = 0.55\end{aligned}$$

1.4.2 Hidden Layer Updates

$$
\begin{aligned}
w_1' &= 0.15 - 0.5 \times (0.009 \times 0.05) = 0.149775 \\
w_2' &= 0.20 - 0.5 \times (0.009 \times 0.10) = 0.19955 \\
w_3' &= 0.25 - 0.5 \times (0.01 \times 0.05) = 0.24975 \\
w_4' &= 0.30 - 0.5 \times (0.01 \times 0.10) = 0.2995 \\
b_1' &= 0.35 - 0.5 \times (0.009 + 0.01) = 0.3405
\end{aligned}
$$

2 Problem 2:

2.1 Forward Pass

$$
\begin{aligned}
h_1 &= w_1 i_1 + w_2 i_2 \\
&= (0.11)(2) + (0.21)(3) = 0.22 + 0.63 = 0.85 \\
h_2 &= w_3 i_1 + w_4 i_2 \\
&= (0.12)(2) + (0.08)(3) = 0.24 + 0.24 = 0.48 \\
out &= w_5 h_1 + w_6 h_2 \\
&= (0.14)(0.85) + (0.15)(0.48) = 0.119 + 0.072 = 0.191
\end{aligned}
$$

2.2 Backward Pass

$$
\begin{aligned}
\delta &= (out - y) = (0.191 - 1) = -0.809 \\
\delta_{h_1} &= \delta \times w_5 = (-0.809)(0.14) = -0.11326 \\
\delta_{h_2} &= \delta \times w_6 = (-0.809)(0.15) = -0.12135
\end{aligned}
$$

Weight Updates:

$$
\begin{aligned}
w_5' &= w_5 - \eta \cdot \delta \cdot h_1 = 0.14 - (0.05)(-0.809)(0.85) = 0.174 \\
w_6' &= w_6 - \eta \cdot \delta \cdot h_2 = 0.15 - (0.05)(-0.809)(0.48) = 0.169
\end{aligned}
$$

$$
\begin{aligned}
w_1' &= w_1 - \eta \cdot \delta_{h_1} \cdot i_1 = 0.11 - (0.05)(-0.11326)(2) = 0.12133 \\
w_2' &= w_2 - \eta \cdot \delta_{h_1} \cdot i_2 = 0.21 - (0.05)(-0.11326)(3) = 0.2267 \\
w_3' &= w_3 - \eta \cdot \delta_{h_2} \cdot i_1 = 0.12 - (0.05)(-0.12135)(2) = 0.132135 \\
w_4' &= w_4 - \eta \cdot \delta_{h_2} \cdot i_2 = 0.08 - (0.05)(-0.12135)(3) = 0.0982
\end{aligned}
$$

3 Problem 3:

The softmax function is defined as:

$$
y_i = \frac{e^{x_i}}{\sum_k e^{x_k}} \tag{5}
$$

Taking the derivative:

$$
\frac{\partial y_i}{\partial x_j} = \frac{(e^{x_i})' \sum_k e^{x_k} - e^{x_i}(e^{x_j})'}{(\sum_k e^{x_k})^2}
$$

For diagonal elements ($i = j$):

$$\begin{aligned}\frac{\partial y_i}{\partial x_i} &= \frac{e^{x_i}\sum_k e^{x_k} - e^{x_i}(e^{x_i})}{(\sum_k e^{x_k})^2} \\ &= \frac{e^{x_i}}{\sum_k e^{x_k}}\left(\frac{\sum_k e^{x_k} - e^{x_j}}{\sum_k e^{x_k}}\right) \\ \frac{\partial y_i}{\partial x_i} &= y_i(1 - y_i) \end{aligned} \tag{6}$$

For off-diagonal elements ($i \neq j$):

$$\begin{aligned}\frac{\partial y_i}{\partial x_i} &= \frac{0 * \sum_k e^{x_k} - e^{x_i}(e^{x_j})}{(\sum_k e^{x_k})^2} \\ &= \frac{e^{x_i}}{\sum_k e^{x_k}}\left(\frac{-e^{x_j}}{\sum_k e^{x_k}}\right) \\ \frac{\partial y_i}{\partial x_j} &= -y_i y_j \end{aligned} \tag{7}$$

General expression:

$$\frac{\partial y_i}{\partial x_j} = \begin{cases} y_i(1 - y_i), & \text{if } i = j \\ -y_i y_j, & \text{if } i \neq j \end{cases} \tag{8}$$

4 Question 4:

4.1 Can the model achieve perfect accuracy with $h(x) = c \cdot x$

Since this activation function is linear, the entire network behaves like a linear model. However, the dataset is clearly not linearly separable, so no linear model can classify it perfectly.
Conclusion: No, perfect accuracy is impossible with this activation function.

4.2 Can the model achieve perfect accuracy with the sign function?

This activation function introduces non-linearity by producing binary outputs (0 or 1). With four hidden units, the network can construct a piecewise linear decision boundary, forming a box-like shape that separates the two classes.
Conclusion: Yes, perfect accuracy is possible. The model can use hidden units to approximate the boundary around class 1.

Hidden Layer Weights

The weight matrix for the hidden layer is:

$$W_{\text{hidden}} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$$

The bias vector for the hidden layer is:

$$b_{\text{hidden}} = \begin{bmatrix} -1 \\ 2 \\ -1 \\ 2 \end{bmatrix}$$

Output Layer Weights

The weight vector for the output layer is:

$$W_{\text{output}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The bias for the output layer is:

$$b_{\text{output}} = -3.5$$

4.3 Gradient of ELU:

Differentiating ELU(x) piecewise:
- For $x \geq 0$:

$$ELU'(x) = \frac{d}{dx}x = 1$$

- For $x < 0$:

$$ELU'(x) = \frac{d}{dx}\left(\alpha(e^x - 1)\right) = \alpha e^x$$

Thus, the gradient of ELU is:

$$ELU'(x) = \begin{cases} 1, & x \geq 0 \\ \alpha e^x, & x < 0 \end{cases}$$

4.4 Advantage of ELU over ReLU:

One major advantage of ELU over ReLU is that ELU allows negative values, which helps avoid dying neurons.

- ReLU suffers from the "dying ReLU" problem, where neurons can become inactive if they output zero for all future inputs.
- ELU avoids this issue because for negative inputs, it smoothly approaches $-\alpha$ instead of being exactly zero.
- This leads to better gradient flow and more stable training, especially in deep networks.

5 Problem 5:

5.1 Value of n_x

The input feature vector $x^{(i)}$ has 300 features. Therefore:

$$n_x = 300$$

5.2 Number of Weights and Bias Dimension in Logistic Regression

A logistic regression model computes:

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$$

where:
- The weight vector w has 300 elements:

$$w \in \mathbb{R}^{300\times 1}$$

- The bias b is a scalar:

$$b \in \mathbb{R}^{1\times 1}$$

5.3 Dimension of $\hat{y}^{(i)}$ and Expression in Terms of $x^{(i)}$, w, and b

The output $\hat{y}^{(i)}$ is a probability:

$$\hat{y}^{(i)} \in \mathbb{R}^{1\times 1}$$

The prediction formula is:

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$$

where:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

5.4 Number of Weights and Biases in a Single Hidden Layer Neural Network

The neural network has:
- 300 input features - 80 hidden neurons - 1 output neuron

Hidden Layer (Input to Hidden): Each hidden neuron has 300 input connections and 1 bias:

$$W_{\text{hidden}} \in \mathbb{R}^{80\times 300}, \quad b_{\text{hidden}} \in \mathbb{R}^{80}$$

Total parameters in the hidden layer:

$$(300 \times 80) + 80 = 24080$$

Output Layer (Hidden to Output): Each of the 80 hidden neurons connects to 1 output neuron, with 1 bias:

$$W_{\text{output}} \in \mathbb{R}^{1\times 80}, \quad b_{\text{output}} \in \mathbb{R}$$

Total parameters in the output layer:

$$(80 \times 1) + 1 = 81$$

Total Trainable Parameters:

$$24080 + 81 = 24161$$

5.5 Problem with Using ReLU Before Sigmoid

If we apply ReLU before the sigmoid:

$$\hat{y} = \sigma(\text{ReLU}(z))$$

where z is the pre-activation of the output neuron.
Problem: - ReLU forces all values of z to be non-negative. - The sigmoid function maps non-negative inputs to values in $[0.5, 1]$. - This means the model can **never predict class 0** correctly.

6 Problem 6:

6.1 Can Logistic Regression Achieve 100% Training Accuracy?

Logistic regression finds a linear decision boundary to separate the two classes. The decision boundary for logistic regression is determined by:

$$P(Y = 1 \mid X) = \frac{1}{1 + e^{-(w^T X + b)}}$$

To achieve 100% accuracy, there must exist a linear boundary that perfectly separates the two distributions. However, because the data is sampled from two Gaussian distributions with overlapping variance ($\sigma_0^2 = 2$, $\sigma_1^2 = 3$), there will always be some probability that a sample from $Y = 0$ falls into the region of $Y = 1$, and vice versa.
Since logistic regression is a **linear classifier**, it cannot perfectly classify two distributions with significant overlap. Therefore, **logistic regression cannot achieve 100% training accuracy**.

6.2 Which Techniques Improve Training Accuracy?

(d) Using a 2-Hidden Layer Feedforward Network with ReLU
By introducing non-linearity (ReLU activations), the neural network can learn complex decision boundaries that separate the two Gaussian distributions more effectively. A sufficiently deep neural network with non-linearity can approximate the Bayes-optimal decision boundary, leading to a significant improvement in training accuracy.

7 Problem 7:

7.1 Shapes of $W^{[2]}$, $b^{[2]}$, and Hidden Layer Output with Batch Processing

- The output layer maps D_a hidden units to K classes:

- $W^{[2]} \in \mathbb{R}^{K \times D_a}$
- $b^{[2]} \in \mathbb{R}^{K \times 1}$
- $z^{[2]} \in \mathbb{R}^{K \times 1}$

- For a batch of m samples $X \in \mathbb{R}^{D_x \times m}$:

- $a^{[1]}$ has shape $\mathbb{R}^{D_a \times m}$
- $z^{[2]}$ has shape $\mathbb{R}^{K \times m}$
- $\hat{y}$ has shape $\mathbb{R}^{K \times m}$

7.2 Compute $\frac{\partial \hat{y}_k}{\partial z_k^{[2]}}$

By differentiating the softmax function:

$$\hat{y}_k = \frac{\exp(z_k^{[2]})}{\sum_{j=1}^{K} \exp(z_j^{[2]})}$$

Applying the derivative:

$$\frac{\partial \hat{y}_k}{\partial z_k^{[2]}} = \hat{y}_k(1 - \hat{y}_k)$$

7.3 $\frac{\partial \hat{y}_k}{\partial z_i^{[2]}}$ for $i \neq k$

For $i \neq k$:

$$\frac{\partial \hat{y}_k}{\partial z_i^{[2]}} = -\hat{y}_k \hat{y}_i$$

7.4 Compute $\frac{\partial L}{\partial z_i^{[2]}}$

Since the label y is one-hot encoded, assume $y_k = 1$, then:

$$L = -\log(\hat{y}_k)$$

Differentiating:

$$\frac{\partial L}{\partial z_i^{[2]}} = \begin{cases} \hat{y}_i - 1, & \text{if } i = k \\ \hat{y}_i, & \text{otherwise} \end{cases}$$

7.5 Compute $\frac{\partial z^{[2]}}{\partial a^{[1]}}$ (denoted as δ_1)

$$\delta_1 = \frac{\partial z^{[2]}}{\partial a^{[1]}} = W^{[2]}$$

7.6 Compute $\frac{\partial a^{[1]}}{\partial z^{[1]}}$ (denoted as δ_2)

LeakyReLU is defined as:

$$a_j^{[1]} = \begin{cases} z_j^{[1]}, & \text{if } z_j^{[1]} > 0 \\ 0.01 z_j^{[1]}, & \text{if } z_j^{[1]} < 0 \end{cases}$$

Thus,

$$\delta_2 = \frac{\partial a^{[1]}}{\partial z^{[1]}} = \begin{cases} 1, & z_j^{[1]} > 0 \\ 0.01, & z_j^{[1]} < 0 \end{cases}$$

7.7 Compute $\frac{\partial L}{\partial W^{[1]}}$ and $\frac{\partial L}{\partial b^{[1]}}$

Denote:

$$\delta_0 = \frac{\partial L}{\partial z^{[2]}}$$

Using chain rule:

$$\frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}} = \delta_0 \delta_1 \delta_2 x^T$$

$$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}} = \delta_0 \delta_1 \delta_2$$

7.8 Numerical Stability in Softmax Computation

The original softmax formula:

$$\hat{y}_i = \frac{\exp(z_i^{[2]})}{\sum_{j=1}^{K} \exp(z_j^{[2]})}$$

can cause numerical instability when $z_i^{[2]}$ values are large. Specifically, computing $\exp(z_i^{[2]})$ for large $z_i^{[2]}$ leads to overflow.
A common trick is to subtract the maximum value $m = \max_i z_i^{[2]}$:

$$\hat{y}_i = \frac{\exp(z_i^{[2]} - m)}{\sum_{j=1}^{K} \exp(z_j^{[2]} - m)}$$

This helps because $z_i^{[2]} - m$ shifts all values into a range where exponentiation does not overflow, thus preventing numerical instability.

8 Problem 8:

8.1 Shuffling the Training Data in BGD

Yes, shuffling the training data would help. Since the dataset is collected sequentially (guitar clips first, then violin clips, etc.), training without shuffling causes the model to see only one instrument type at a time per epoch. This can lead to poor generalization and ineffective updates. Shuffling ensures that each batch contains a mix of different instrument types, leading to better convergence.

8.2 Choosing MBGD Over SGD

Mini-batch gradient descent (MBGD) balances efficiency and convergence stability. Unlike SGD, which updates weights after each sample and can be noisy, MBGD updates weights using a batch of samples, leading to a more stable gradient estimate while still benefiting from faster convergence than batch gradient descent (BGD).

8.3 Choosing MBGD Over BGD

BGD computes gradients using the entire dataset, making updates slow, especially for large datasets. MBGD, on the other hand, updates the parameters more frequently (after each mini-batch), which speeds up training while still maintaining a stable gradient estimate.

8.4 Identifying the Training Loss Curves

- **A - BGD**: The loss decreases smoothly because BGD computes the exact gradient using the entire dataset.
- **B - MBGD**: The loss decreases steadily but with small fluctuations, indicating mini-batches are being used.
- **C - SGD**: The loss fluctuates significantly, suggesting weight updates are made after each sample, which introduces high variance.

8.5 Sign of a Learning Rate Being Too Large

A too-large learning rate may cause the loss to oscillate wildly or even diverge instead of decreasing.

8.6 Sign of a Learning Rate Being Too Small

If the learning rate is too small, training will be excessively slow, and the loss may decrease very gradually, taking many epochs to converge.

8.7 Gradient Descent with Momentum Update Rule

Momentum helps accelerate gradient descent by accumulating past gradients. The update rule is:

$$v_t = \beta v_{t-1} + (1 - \beta)\frac{\partial J}{\partial W} \tag{9}$$

$$W = W - \alpha v_t \tag{10}$$

where:

- v_t is the velocity (accumulated gradient),
- β is the momentum coefficient,
- α is the learning rate,
- $\frac{\partial J}{\partial W}$ is the gradient of the cost function.

8.8 How Momentum Speeds Up Learning

Momentum smooths out updates by allowing gradients to accumulate, preventing oscillations and helping the model converge faster, especially in ravines or highly curved loss surfaces.