CSE 151B/251B Deep Learning
UC San Diego, Spring 2025

Homework 1

Name: Ouyang Yingxuan UID: 10486415

1 Problem 1 Supervised Learning

Problem A: Feature Representation

Solution A: *After converting each sentence to a bag-of-words vector using the dictionary [learning, data, model, evaluation, deep], we can have the following feature vectors:*

Doc1: [2, 2, 1, 0, 0]
Doc2: [1, 1, 1, 0, 0]
Doc3: [2, 0, 0, 0, 0]
Doc4: [3, 1, 0, 0, 2]
Doc5: [0, 1, 2, 2, 0]

Here is the matrix representing all of the documents, in the matrix each row represents a document and each column to a word in the dictionary. :

Document	learning	data	model	evaluation	deep
Doc1	2	2	1	0	0
Doc2	1	1	1	0	0
Doc3	2	0	0	0	0
Doc4	3	1	0	0	2
Doc5	0	1	2	2	0

Problem B: Logistic Regressio

Solution B: *The logistic regression model is:*

$$f(x) = \sigma(w^\top x), \quad \sigma(x) = \frac{1}{1+\exp(-x)}$$

The cross-entropy loss function is:

$$L(y, f(x)) = -\sum_{i=1}^{N} \left[y^{(i)} \log(f(x^{(i)})) + (1 - y^{(i)}) \log(1 - f(x^{(i)})) \right]$$

We can denote $f(x^{(i)}) = \hat{y}^{(i)} = \sigma(w^\top x^{(i)})$.

Now we can compute the derivative of the loss with respect to w_j*:*

$$\frac{\partial L}{\partial w_j} = -\sum_{i=1}^{N} \left[\frac{y^{(i)}}{\hat{y}^{(i)}} \cdot \frac{\partial \hat{y}^{(i)}}{\partial w_j} - \frac{1 - y^{(i)}}{1 - \hat{y}^{(i)}} \cdot \frac{\partial \hat{y}^{(i)}}{\partial w_j} \right]$$

The derivative of the sigmoid function is:

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

So:

$$\frac{\partial \hat{y}^{(i)}}{\partial w_j} = \hat{y}^{(i)}(1 - \hat{y}^{(i)}) x_j^{(i)}$$

Plugging this back in:

$$\frac{\partial L}{\partial w_j} = -\sum_{i=1}^{N} \left[\frac{y^{(i)} - \hat{y}^{(i)}}{\hat{y}^{(i)}(1 - \hat{y}^{(i)})} \cdot \hat{y}^{(i)}(1 - \hat{y}^{(i)}) x_j^{(i)} \right] = \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

Therefore:

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{N} (f(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

2 Problem 2 Multi-Layer Perceptron

Problem A: Perceptron

i. Implementation of Perceptron

Solution A: *We update the parameters w and b using one misclassified point at a time. This is done by scanning through all input samples, identifying one point where the current prediction $f(x)$ does not match the true label y, and applying the update rules:*

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y(t) \cdot \mathbf{x}(t) \quad \textit{and} \quad b_{t+1} = b_t + y(t)$$

This ensures that the selected misclassified point is pushed closer to the correct side of the decision boundary.
To complete the perceptron learning algorithm, we repeatedly update the weights and bias using the `update_perceptron()` *method. We stop either when there are no misclassified points (i.e., the model has converged), or when we reach the maximum number of iterations* `max_iter`*. At each step, we check if the weights or bias changed; if not, the algorithm terminates early.*

The implementation in Python is in the Jupyter notebook.

ii. Linear Separability

Solution A.ii:

In 2D: *The smallest such dataset has* ***4 points*** *in general position (i.e., no three are collinear). This configuration can form the XOR pattern, which is not linearly separable.*

In 3D: *The smallest such dataset has* ***5 points*** *where no four are coplanar. This is the 3D analogue of XOR, where one point is inside a tetrahedron formed by the others, making linear separation impossible.*

In N-D: *The smallest non-linearly separable dataset consists of* ***N + 2 points****, assuming no subset lies within a lower-dimensional hyperplane.*

iii. Non-linearly Separable Dataset

Solution A.iii: *No, the Perceptron Learning Algorithm (PLA) will not converge on a dataset which is not linearly separable.*

The algorithm relies on finding a hyperplane that correctly classifies all points. Since there is no such hyperplane exists, the algorithm will continue to update the weights indefinitely, as there will always be at least one misclassified point. This leads to an infinite loop or until a predefined maximum number of iterations is reached.

In the visualization with the given dataset (which resembles the XOR pattern), the PLA fails to find a decision boundary that separates positive and negative points, demonstrating its non-convergence in practice.

Problem B: Multilayer Perceptron

i. MNIST Classification Implementation

Solution B: *To classify MNIST digits, we implemented a two-layer MLP with the following architecture:*

Input: 784-dimensional image vector (28x28 flattened)
Hidden Layer: Fully connected layer with 500 ReLU-activated hidden units
Output Layer: Fully connected layer with 10 logits (one per digit)

We use `torch.nn.CrossEntropyLoss()` *for training, which internally applies softmax. Therefore, we do not apply a separate softmax in the model's forward pass. The training loop involves standard PyTorch procedures: data loading, model training with SGD/Adam, and evaluation on the test set.*

The model was implemented using `nn.Module` *for modularity and readability. This implementation gives a good balance between flexibility and clarity.*

ii. Parameter Counts

Solution B: *To compute the number of trainable parameters in the MLP model:*

Layer 1: 784 input features $\times$ 500 hidden units + 500 biases = 392,000 + 500 = 392,500
Layer 2: 500 hidden units $\times$ 10 output units + 10 biases = 5,000 + 10 = 5,010

Total trainable parameters = 392,500 + 5,010 = ***397,510***

Compared to logistic regression ($784 \times 10 + 10 = 7{,}850$), this MLP model has significantly more parameters, making it more expressive but also harder to train.