Deep Learning Assignment-01

MTech (CS), IIIT Bhubaneswar
March - 2025

Student ID: A124014
Student Name:Soham Chakraborty

Solutions of Deep Learning Assignment

1. : For a D -dimensional input vector, show that the optimal weights can be represented by the expression: 1

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$$

What is the possible estimation of $\mathbf{w}$?

Solution:

(a) Least-squares objective. With data $\{(x_n, t_n)\}_{n=1}^N$, define

$$X = \begin{pmatrix} x_1^T \\ \vdots \\ x_N^T \end{pmatrix} \in \mathbb{R}^{N\times D}, \quad t = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix} \in \mathbb{R}^N.$$

The sum of squared errors is

$$J(w) = \sum_{n=1}^{N} \big(w^T x_n - t_n\big)^2 = \|Xw - t\|^2.$$

(b) Normal equations & solution.

$$J(w) = (Xw - t)^T(Xw - t) = w^T X^T X\, w - 2\, t^T X\, w + t^T t.$$

Taking $\nabla_w J = 0$ gives

$$2\, X^T X\, w - 2\, X^T t = 0 \quad \Longrightarrow \quad X^T X\, w = X^T t.$$

Assuming $X^T X$ is invertible,

$$\boxed{\hat{w} = (X^T X)^{-1} X^T\, t}.$$

(c) Statistical interpretation. Under the model $t = X\, w_{\text{true}} + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$, maximizing the Gaussian likelihood yields the same normal equations, so $\hat{w}$ is the MLE. Moreover

$$\mathbb{E}[\hat{w}] = w_{\text{true}}, \qquad \text{Cov}(\hat{w}) = \sigma^2 (X^T X)^{-1},$$

showing that $\hat{w}$ is unbiased with minimum variance among unbiased linear estimators.

2. : OR Gate in single neural network

Solution:

We aim to implement an OR logic gate using a single-layer perceptron. The OR gate outputs 1 when at least one of the inputs is 1, and 0 otherwise.

(a) Perceptron model. Let the neuron output:

$$y = \begin{cases} 1, & \text{if } z \geq 0, \\ 0, & \text{otherwise} \end{cases} \quad \text{where} \quad z = w_1 x_1 + w_2 x_2 + b.$$

(b) Initial parameters. Choose:

$$w_1 = 1.5, \quad w_2 = 2.0, \quad b = -2.0, \quad \eta = 0.5.$$

(c) OR truth table and neuron evaluation.

x_1	x_2	$z = w_1 x_1 + w_2 x_2 + b$	y	t
0	0	-2.0	0	0
0	1	0.0	1	1
1	0	-0.5	0	1
1	1	1.5	1	1

Only the input (1,0) is misclassified. For this, $y = 0$, $t = 1$, and we apply the perceptron update rule:

$$\Delta w_i = \eta(t - y)x_i, \quad \Delta b = \eta(t - y).$$

Update for (1,0):

$$\Delta w_1 = 0.5, \quad \Delta w_2 = 0, \quad \Delta b = 0.5.$$

New parameters:

$$w_1 = 2.0, \quad w_2 = 2.0, \quad b = -1.5.$$

(d) Final check. With updated weights, all inputs are classified correctly:

x_1	x_2	$z = w^T x + b$	y
0	0	-1.5	0
0	1	0.5	1
1	0	0.5	1
1	1	2.5	1

(e) Decision boundary. The final decision boundary is:

$$2x_1 + 2x_2 - 1.5 = 0 \quad \Rightarrow \quad x_1 + x_2 = 0.75.$$

3. :Design a Perceptron algorithm to classify Iris flowers using either sepal or petal features and create a decision boundary.

Solution:

We aim to design a perceptron algorithm to classify two classes of Iris flowers using either sepal or petal features and derive the corresponding decision boundary.

(a) Data preprocessing.

- Select two classes from the Iris dataset (e.g., Setosa vs Versicolor).
- Choose either sepal features (x_1 = sepal length, x_2 = sepal width) or petal features (x_1 = petal length, x_2 = petal width).
- Normalize features using min-max scaling:

$$x_i^{\text{norm}} = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}.$$

- Map class labels to binary values: one class as 0 and the other as 1.

(b) Perceptron model.

$$\hat{y} = \begin{cases} 1, & \text{if } z \geq 0, \\ 0, & \text{otherwise} \end{cases} \quad \text{where} \quad z = w_1 x_1 + w_2 x_2 + b.$$

(c) Initialization.

$$w_1, w_2 \sim \mathcal{U}(-0.01, 0.01), \quad b = 0, \quad \eta = 0.05.$$

(d) Learning algorithm. For each training example (x, y), compute the predicted output $\hat{y}$ and apply the following update rule if misclassified:

$$w_i \leftarrow w_i + \eta(y - \hat{y})x_i, \quad b \leftarrow b + \eta(y - \hat{y}).$$

Repeat for several epochs (iterations over the full training set), shuffling the data in each epoch.

(e) Decision boundary. After training, the decision boundary is defined by:

$$w_1 x_1 + w_2 x_2 + b = 0.$$

This can be rearranged to the line:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}.$$

This line visually separates the two classes in the feature space.

(f) Visualization. Plotting the normalized data points and the decision line allows us to visually verify how well the perceptron separates the two Iris classes.

(g) Python Implementation (Sepal or Petal-based Perceptron)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler

# Load and prepare data
iris = load_iris()
X = iris.data
y = iris.target

# Select two classes (e.g., Setosa and Versicolor) and two features (e.g., Petal)
feature_indices = [2, 3]  # Petal length and width
class_indices = y != 2     # Exclude class 'Virginica'
X = X[class_indices][:, feature_indices]
y = y[class_indices]

# Normalize features
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# Map class labels to 0 and 1
y = (y == 1).astype(int)

# Initialize weights and bias
w = np.random.uniform(-0.01, 0.01, size=2)
b = 0
eta = 0.05
epochs = 20

# Perceptron training loop
for epoch in range(epochs):
    for xi, target in zip(X, y):
        z = np.dot(w, xi) + b
        y_pred = 1 if z >= 0 else 0
        error = target - y_pred
        if error != 0:
            w += eta * error * xi
            b += eta * error

# Plotting decision boundary
plt.figure()
for i, label in enumerate(['Class 0', 'Class 1']):
    plt.scatter(X[y == i, 0], X[y == i, 1], label=label)
x_vals = np.array([0, 1])
y_vals = -(w[0] * x_vals + b) / w[1]
plt.plot(x_vals, y_vals, 'k--', label='Decision Boundary')
plt.xlabel("Feature 1 (Petal length)")
plt.ylabel("Feature 2 (Petal width)")
plt.legend()
```

```
plt.title("Perceptron Classification of Iris")
plt.show()
```

4. For the given graph, give the following solutions.

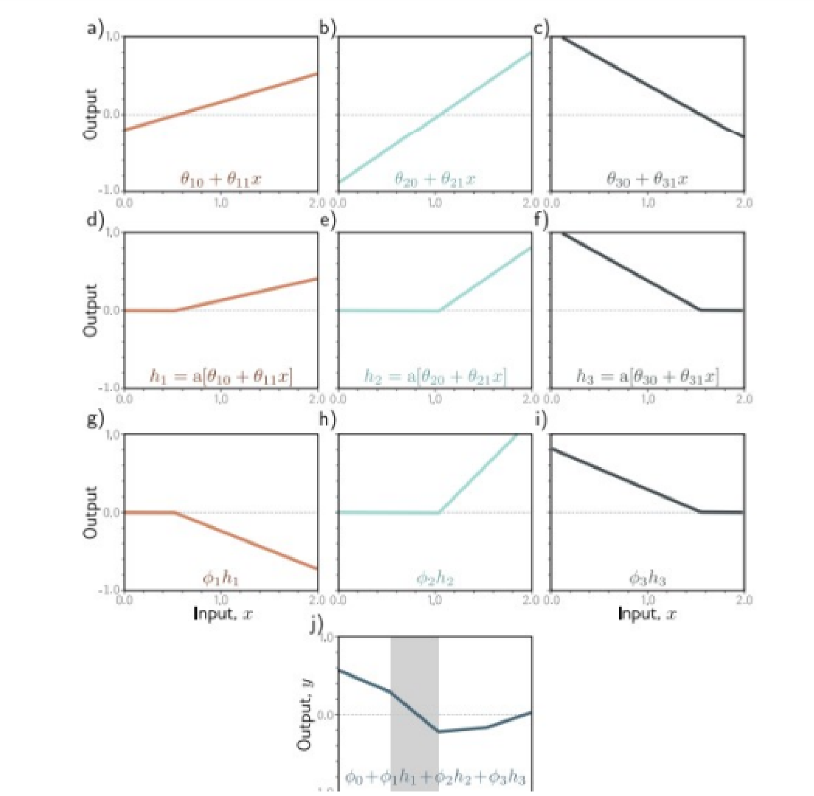

Figure 1: generalization of intersection

(a) Generalized Point of Intersection for Shallow Neural Networks for input space parameterized by spherical coordinates θ and ϕ.

Solution:
Let each hyperplane in the shallow neural network be defined by the equation:

$$w_i^T x + b_i = 0.$$

In spherical coordinates, the weight vector $w_i \in \mathbb{R}^3$ can be expressed as:

$$w_i = \|w_i\| \begin{bmatrix} \sin\theta_i \cos\phi_i \\ \sin\theta_i \sin\phi_i \\ \cos\theta_i \end{bmatrix}.$$

Given two such planes defined by weights w_i, w_j and biases b_i, b_j, the point of intersection lies along the line satisfying:

$$w_i^T x + b_i = 0, \quad w_j^T x + b_j = 0.$$

These form a linear system in x. If we define matrix $H = \begin{bmatrix} w_i^T \\ w_j^T \end{bmatrix}$ and vector $b = \begin{bmatrix} -b_i \\ -b_j \end{bmatrix}$, then the intersection point (or line, in higher dimension) is given by:

$$x = H^{-1}b \quad \text{(when } H \text{ is invertible)}.$$

This generalized intersection point allows shallow networks to capture transitions and combinations of piecewise linear segments as depicted in the plot.

(b) Give the equation of the 4 line segments in the graph in terms of θ_1, θ_2, θ_3, etc., for the figure.

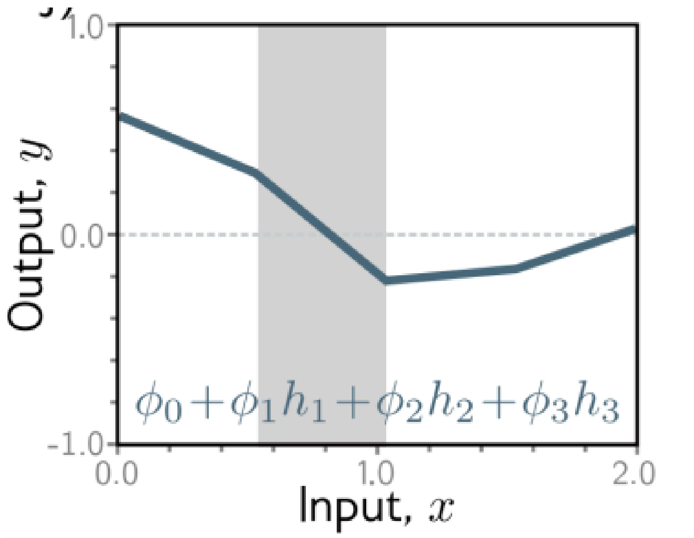

Figure 2: 4 line equations

Solution:
The output of the shallow neural network is composed of three ReLU activations:

$$h_i(x) = \sigma(\theta_{i0} + \theta_{i1}x), \quad \text{for } i = 1, 2, 3,$$

and the final output is a weighted sum:

$$y(x) = \phi_0 + \phi_1 h_1(x) + \phi_2 h_2(x) + \phi_3 h_3(x).$$

Each $h_i(x)$ introduces a kink (non-linearity) at a threshold:

$$x_i = -\frac{\theta_{i0}}{\theta_{i1}}.$$

Thus, the function $y(x)$ is piecewise linear and defined over four regions:

$$
y(x) = \begin{cases} \phi_0, & x < x_1, \\ \\ \phi_0 + \phi_1(\theta_{10} + \theta_{11}x), & x_1 \leq x < x_2, \\ \\ \phi_0 + \phi_1(\theta_{10} + \theta_{11}x) + \phi_2(\theta_{20} + \theta_{21}x), & x_2 \leq x < x_3, \\ \\ \phi_0 + \sum_{i=1}^{3} \phi_i(\theta_{i0} + \theta_{i1}x), & x \geq x_3, \end{cases}
$$

where x_1, x_2, x_3 are the activation thresholds for the three ReLU units. Each segment corresponds to a region where a new ReLU becomes active, resulting in a change in slope.

5. What will be the General Form of the second output in the Two-Output Feedforward Neural Network (2D Case) if one of the outputs is given?

Solution:

Assume a standard two-output feedforward neural network with:

- 2 input features: x_1, x_2,
- D hidden units with activation: $h_d = \sigma(\delta_{d0} + \delta_{d1}x_1 + \delta_{d2}x_2)$,
- Linear output layer with 2 outputs: y_1, y_2.

If one of the outputs (say y_1) is given in the form:

$$
y_1 = \gamma_{10} + \sum_{d=1}^{D} \gamma_{1d} h_d,
$$

then the second output will have the general form:

$$
y_2 = \gamma_{20} + \sum_{d=1}^{D} \gamma_{2d} h_d,
$$

where:

- γ_{20} is the bias term for the second output,
- γ_{2d} are the weights from hidden neuron h_d to output neuron 2.

Thus, both outputs share the same hidden layer representation but have different output weights and biases. This is standard in multi-output feedforward networks where hidden features are reused.

6. Let $x_1, x_2, \ldots, x_n$ be independent and identically distributed (i.i.d.) vectors from a multivariate normal distribution:

$$
x_i \sim \mathcal{N}(\mu, \Sigma),
$$

where μ is the unknown mean vector and Σ is the known covariance matrix. Derive the maximum likelihood estimator (MLE) for μ.

Solution:

We are given:

- Each observation $x_i \in \mathbb{R}^d$,
- $x_i \sim \mathcal{N}(\mu, \Sigma)$ i.i.d.,
- $\Sigma \in \mathbb{R}^{d\times d}$ is known and positive definite,
- $\mu \in \mathbb{R}^d$ is unknown and must be estimated.

Step 1: Write the likelihood function. The probability density function of a multivariate normal distribution is:

$$p(x_i \mid \mu) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right).$$

For n i.i.d. samples, the likelihood function is:

$$L(\mu) = \prod_{i=1}^{n} p(x_i \mid \mu).$$

Step 2: Take the log-likelihood.

$$\ell(\mu) = \log L(\mu) = -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu).$$

Step 3: Maximize the log-likelihood. Since the first two terms are constants, we minimize the quadratic term:

$$\ell(\mu) = \text{const} - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu).$$

Take the gradient of $\ell(\mu)$ with respect to μ:

$$\nabla_\mu \ell(\mu) = \sum_{i=1}^{n}\Sigma^{-1}(x_i-\mu) = \Sigma^{-1}\sum_{i=1}^{n}(x_i-\mu).$$

Set derivative to zero:

$$\sum_{i=1}^{n}(x_i-\mu) = 0 \quad \Rightarrow \quad n\mu = \sum_{i=1}^{n} x_i \quad \Rightarrow \quad \boxed{\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i}.$$

Conclusion: The MLE of μ is simply the sample mean of the observations.

7. The Backpropagation for the cross-entropy loss function of a network with 3 outputs

f_1, f_2, f_3. Assume that these outputs are the only parameters of the loss function. Derive $\frac{\partial L}{\partial f_i}$ for $i = 1, 2, 3$.

Solution:

We assume that the final output layer produces 3 scores $f_1, f_2, f_3 \in \mathbb{R}$, which are passed through the softmax function to obtain probabilities p_i for class i. Let the true label be encoded as a one-hot vector $y = [y_1, y_2, y_3]$, where exactly one $y_i = 1$, and the others are 0.

Step 1: Softmax Function

$$p_i = \frac{e^{f_i}}{\sum_{j=1}^{3} e^{f_j}}, \quad \text{for } i = 1, 2, 3.$$

Step 2: Cross-Entropy Loss

$$L = -\sum_{i=1}^{3} y_i \log p_i.$$

Step 3: Gradient of the Loss We compute the partial derivative of L with respect to f_k, using the chain rule:

$$\frac{\partial L}{\partial f_k} = \sum_{i=1}^{3} \frac{\partial L}{\partial p_i} \cdot \frac{\partial p_i}{\partial f_k}.$$

It is a known result that for softmax + cross-entropy combined, the gradient simplifies to:

$$\boxed{\frac{\partial L}{\partial f_k} = p_k - y_k.}$$

This holds because:

$$\frac{\partial p_i}{\partial f_k} = \begin{cases} p_i(1 - p_i), & \text{if } i = k, \\ -p_i p_k, & \text{if } i \neq k, \end{cases}$$

and applying this to the sum gives the desired simplification.

Final Result: For a network with 3 outputs f_1, f_2, f_3, the gradient of the loss with respect to each f_i is:

$$\boxed{\frac{\partial L}{\partial f_i} = p_i - y_i \quad \text{for } i = 1, 2, 3.}$$

8. Backpropagation for 3-class classification using a neural network with 2 inputs, 2 hidden sigmoid units, and 3 softmax output neurons. Derive the forward and backward pass expressions assuming cross-entropy loss.

Solution:

We consider a feedforward neural network with:

- Input layer: $\mathbf{x} \in \mathbb{R}^2$,
- Hidden layer: 2 units with sigmoid activation,
- Output layer: 3 units with softmax activation,
- Loss function: cross-entropy.

Forward Pass:

Let the parameters be:

- $W^{[1]} \in \mathbb{R}^{2\times 2}$, $b^{[1]} \in \mathbb{R}^2$: weights and biases for hidden layer,
- $W^{[2]} \in \mathbb{R}^{3\times 2}$, $b^{[2]} \in \mathbb{R}^3$: weights and biases for output layer.

The forward computation:

$$\begin{aligned}
z^{[1]} &= W^{[1]}x + b^{[1]}, && \text{(shape: } 2 \times 1) \\
a^{[1]} &= \sigma(z^{[1]}), && \text{(hidden layer output)} \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]}, && \text{(shape: } 3 \times 1) \\
\hat{y} &= \text{softmax}(z^{[2]}), && \hat{y}_i = \frac{e^{z_i^{[2]}}}{\sum_j e^{z_j^{[2]}}}.
\end{aligned}$$

Loss (cross-entropy):

$$L = -\sum_{i=1}^{3} y_i \log(\hat{y}_i),$$

where $y \in \{0,1\}^3$ is a one-hot vector.

Backward Pass:

Step 1: Output layer error

$$\delta^{[2]} = \frac{\partial L}{\partial z^{[2]}} = \hat{y} - y.$$

Step 2: Gradients for output layer weights and bias

$$\begin{aligned}
\frac{\partial L}{\partial W^{[2]}} &= \delta^{[2]} \cdot (a^{[1]})^T, \\
\frac{\partial L}{\partial b^{[2]}} &= \delta^{[2]}.
\end{aligned}$$

Step 3: Backpropagate to hidden layer

$$\delta^{[1]} = (W^{[2]})^T \delta^{[2]} \odot \sigma'(z^{[1]}),$$

where $\odot$ denotes element-wise multiplication and $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

Step 4: Gradients for hidden layer weights and bias

$$\frac{\partial L}{\partial W^{[1]}} = \delta^{[1]} \cdot x^T,$$
$$\frac{\partial L}{\partial b^{[1]}} = \delta^{[1]}.$$

Summary of Gradient Flow:

$$\boxed{\delta^{[2]} = \hat{y} - y, \quad \delta^{[1]} = (W^{[2]})^T (\hat{y} - y) \odot \sigma'(z^{[1]})}$$

Then compute gradients for all weights and biases accordingly.