Deep Learning Assignment-01

MTech (CS), IIIT Bhubaneswar
March - 2025

Student ID: A124015
Student Name: Tarun Raj

Solutions of Deep Learning Assignment 01

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

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

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

Solution:

To derive the optimal weights $\mathbf{w}$ for a linear regression problem, we start with the least squares objective. Given a dataset with N samples, where $\mathbf{X}$ is the $N \times D$ design matrix (each row corresponds to a D-dimensional input vector), $\mathbf{t}$ is the $N \times 1$ target vector, and $\mathbf{w}$ is the $D \times 1$ weight vector, the goal is to minimize the sum of squared errors:

$$E(\mathbf{w}) = \|\mathbf{t} - \mathbf{X}\mathbf{w}\|^2$$

Expanding the squared error term:

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

Taking the derivative of $E(\mathbf{w})$ with respect to $\mathbf{w}$ and setting it to zero for minimization:

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}^T(\mathbf{t} - \mathbf{X}\mathbf{w}) = 0$$

Rearranging the equation:

$$\mathbf{X}^T\mathbf{t} - \mathbf{X}^T\mathbf{X}\mathbf{w} = 0$$

Solving for $\mathbf{w}$:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{t}$$

Assuming $\mathbf{X}^T\mathbf{X}$ is invertible, the optimal weight vector $\mathbf{w}$ is:

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

This is the least squares solution for the weight vector $\mathbf{w}$.

Estimation of w

The expression $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$ provides the optimal weights that minimize the sum of squared errors between the predicted values $\mathbf{X}\mathbf{w}$ and the target values $\mathbf{t}$. This is the best linear unbiased estimator (BLUE) under the assumptions of linear

regression (e.g., no multicollinearity, homoscedasticity, and normally distributed errors).
Thus, the estimation of $\mathbf{w}$ is given by:

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

2. OR Gate Implementation Using a Single-Layer Neural Network

Solution:

OR Gate Using a Perceptron

The perceptron implements the OR logic through:

$$y = f(w_1x_1 + w_2x_2 + b)$$

Where:

- Initial weights: $w_1 = 1.5$, $w_2 = 2$
- Initial bias: $b = -2$
- Learning rate: $\eta = 0.5$
- Step activation:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

Learning Process

Epoch 1 - Initial Verification

(a) **Input (0,0):** $z = 1.5(0) + 2(0) - 2 = -2 \implies y = 0$ ✓

(b) **Input (0,1):** $z = 1.5(0) + 2(1) - 2 = 0 \implies y = 1$ ✓

(c) **Input (1,0):** $z = 1.5(1) + 2(0) - 2 = -0.5 \implies y = 0 \times$ (Expected 1)

Weight Update:

$$\begin{aligned}
\Delta w_1 &= \eta(t-y)x_1 = 0.5(1-0)(1) = 0.5 \\
\Delta w_2 &= \eta(t-y)x_2 = 0.5(1-0)(0) = 0 \\
\Delta b &= \eta(t-y) = 0.5(1-0) = 0.5 \\
w_1 &\leftarrow 1.5 + 0.5 = 2.0 \\
w_2 &\leftarrow 2 + 0 = 2.0 \\
b &\leftarrow -2 + 0.5 = -1.5
\end{aligned}$$

4. **Input (1,1)** with updated parameters: $z = 2(1) + 2(1) - 1.5 = 2.5 \implies y = 1$ ✓

Epoch 2 - Verification with Updated Parameters

New Parameters: $w_1 = 2.0$, $w_2 = 2.0$, $b = -1.5$

(a) **Input (0,0):** $z = 2(0) + 2(0) - 1.5 = -1.5 \implies y = 0$ ✓

(b) **Input (0,1):** $z = 2(0) + 2(1) - 1.5 = 0.5 \implies y = 1$ ✓

(c) **Input (1,0):** $z = 2(1) + 2(0) - 1.5 = 0.5 \implies y = 1$ ✓

(d) **Input (1,1):** $z = 2(1) + 2(1) - 1.5 = 2.5 \implies y = 1$ ✓

Convergence Achieved

After weight adjustment in Epoch 1, the perceptron correctly classifies all OR gate inputs. The final parameters are:

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

The decision boundary equation becomes:

$$2.0x_1 + 2.0x_2 - 1.5 = 0$$

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

Solution:

Perceptron Algorithm for Iris Classification

Objective: Classify two species of Iris flowers using:

- **Feature choice:** Sepal (length/width) *or* Petal (length/width)
- **Class pairs:** Setosa-Virginica, Setosa-Versicolor, or Versicolor-Virginica

Algorithm Implementation

(a) **Data Preparation**

- Feature selection:

$$\text{Features} = \begin{cases} (\text{sepal length}, \text{sepal width}) & \text{if sepal mode} \\ (\text{petal length}, \text{petal width}) & \text{if petal mode} \end{cases}$$

- Binary encoding (example for Setosa vs Virginica):

$$y = \begin{cases} 1 & \text{Virginica} \\ 0 & \text{Setosa} \end{cases}$$

- Min-max normalization:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

(b) **Model Initialization**

- Weights: $w_1, w_2 \sim \mathcal{U}(-0.01, 0.01)$
- Bias: $b = 0$
- Learning rate: $\eta = 0.1$ (default)

(c) **Training Phase** For each epoch until convergence:

i. Compute activation:
$$z = w_1x_1 + w_2x_2 + b$$

ii. Apply step function:
$$\hat{y} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

iii. Update parameters for misclassified samples:
$$\Delta w_i = \eta(y - \hat{y})x_i \quad (\text{for } i = 1, 2)$$
$$\Delta b = \eta(y - \hat{y})$$

Decision Boundary

The separating line equation:

$$w_1x_1 + w_2x_2 + b = 0$$

Slope-intercept form:
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

def get_user_choice():
    """Get user input for feature selection and class pair"""
    print("Available options:")
    print("1. Sepal features (length & width)")
    print("2. Petal features (length & width)")
    feature_choice = int(input("Enter feature choice (1 or 2): "))

    print("\nClass pairs:")
    print("1. Setosa (0) vs Versicolor (1)")
    print("2. Setosa (0) vs Virginia (2)")
    print("3. Versicolor (1) vs Virginia (2)")
    class_pair = int(input("Enter class pair (1-3): "))

    return feature_choice, class_pair

def prepare_data(feature_choice, class_pair):
```

```
    """Load and prepare data based on user choices"""
    iris = load_iris()

    # Feature selection
    if feature_choice == 1:
        X = iris.data[:, :2] # Sepal features
        feature_names = ["Sepal Length", "Sepal Width"]
    else:
        X = iris.data[:, 2:] # Petal features
        feature_names = ["Petal Length", "Petal Width"]

    # Class pair selection
    class_mapping = {1: (0, 1), 2: (0, 2), 3: (1, 2)}
    class_a, class_b = class_mapping[class_pair]

    # Filter selected classes
    mask = np.logical_or(iris.target == class_a, iris.target == class_b)
    X = X[mask]
    y = iris.target[mask]

    # Convert to binary labels
    y = np.where(y == class_b, 1, 0)

    return X, y, feature_names, iris.target_names[class_a],


    iris.target_names[class_b]

def perceptron_train(X_train, y_train, lr=0.1, epochs=1000):
    """Train perceptron model"""
    weights = np.zeros(X_train.shape[1])
    bias = 0

    for epoch in range(epochs):
        for i in range(len(X_train)):
            z = np.dot(X_train[i], weights) + bias
            y_pred = 1 if z > 0 else 0
            error = y_train[i] - y_pred
            weights += lr * error * X_train[i]
            bias += lr * error

    return weights, bias

def evaluate_model(X_test, y_test, weights, bias):
    """Evaluate model on test data"""
    correct = 0
    for i in range(len(X_test)):
        z = np.dot(X_test[i], weights) + bias
        y_pred = 1 if z > 0 else 0
        if y_pred == y_test[i]:
```

```
            correct += 1
    return correct / len(X_test)

def plot_results(X_train, X_test, y_train, y_test, weights, bias,
                feature_names, class_names):
    """Plot decision boundary with train/test points"""
    plt.figure(figsize=(10, 6))

    # Create mesh grid
    x_min, x_max = X_train[:, 0].min()-1, X_train[:, 0].max()+1
    y_min, y_max = X_train[:, 1].min()-1, X_train[:, 1].max()+1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))

    # Calculate decision boundary
    Z = (weights[0]*xx + weights[1]*yy + bias > 0).astype(int)

    # Plot decision regions
    plt.contourf(xx, yy, Z, alpha=0.2, cmap='RdBu')

    # Plot training points
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train,
               cmap='RdBu', edgecolors='k', label='Train', alpha=0.7)

    # Plot test points
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test,
               cmap='RdBu', edgecolors='k', marker='x', s=100,


               label='Test')

    plt.xlabel(f"{feature_names[0]}(standardized)")
    plt.ylabel(f"{feature_names[1]}(standardized)")
    plt.title(f"Perceptron: {class_names[0]} vs {class_names[1]}")
    plt.legend()
    plt.show()

def main():
    # Get user choices
    feature_choice, class_pair = get_user_choice()

    # Prepare data
    X, y, feature_names, class_a, class_b = prepare_data


    (feature_choice, class_pair)

    # Split data (80-20)
    X_train, X_test, y_train, y_test =
```

```
    train_test_split(X, y, test_size=0.2, random_state=42)

    # Standardize features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train model
    weights, bias = perceptron_train(X_train, y_train)

    # Evaluate
    accuracy = evaluate_model(X_test, y_test, weights, bias)
    print(f"\nTest Accuracy: {accuracy:.2%}")

    # Plot results
    plot_results(X_train, X_test, y_train, y_test, weights, bias,
                feature_names, (class_a, class_b))

if __name__ == "__main__":
    main()
```

Key Properties

- **Convergence**: Guaranteed for linearly separable data
- **Limitations**: Cannot learn non-linear boundaries
- **Interactive Feature Selection**: Users can choose between sepal/petal features
- **Class Pair Flexibility**: Supports all three binary classification combinations
- **Proper Train-Test Split**: 80-20 ratio with stratified sampling
- **Model Evaluation**: Includes accuracy calculation on test set

4. For given graph in figure 1 give the following solutions:

   (a) * Generalizing the Point of Intersection in Terms of θ and ϕ for Shallow Neural Networks.

Solution:

Step 1: Structure of a Shallow Neural Network
Consider a shallow neural network with:

- Input dimension: d
- Number of **hidden neurons**: m
- Activation function: σ
- Weight vectors: $\mathbf{w}_i \in \mathbb{R}^d$

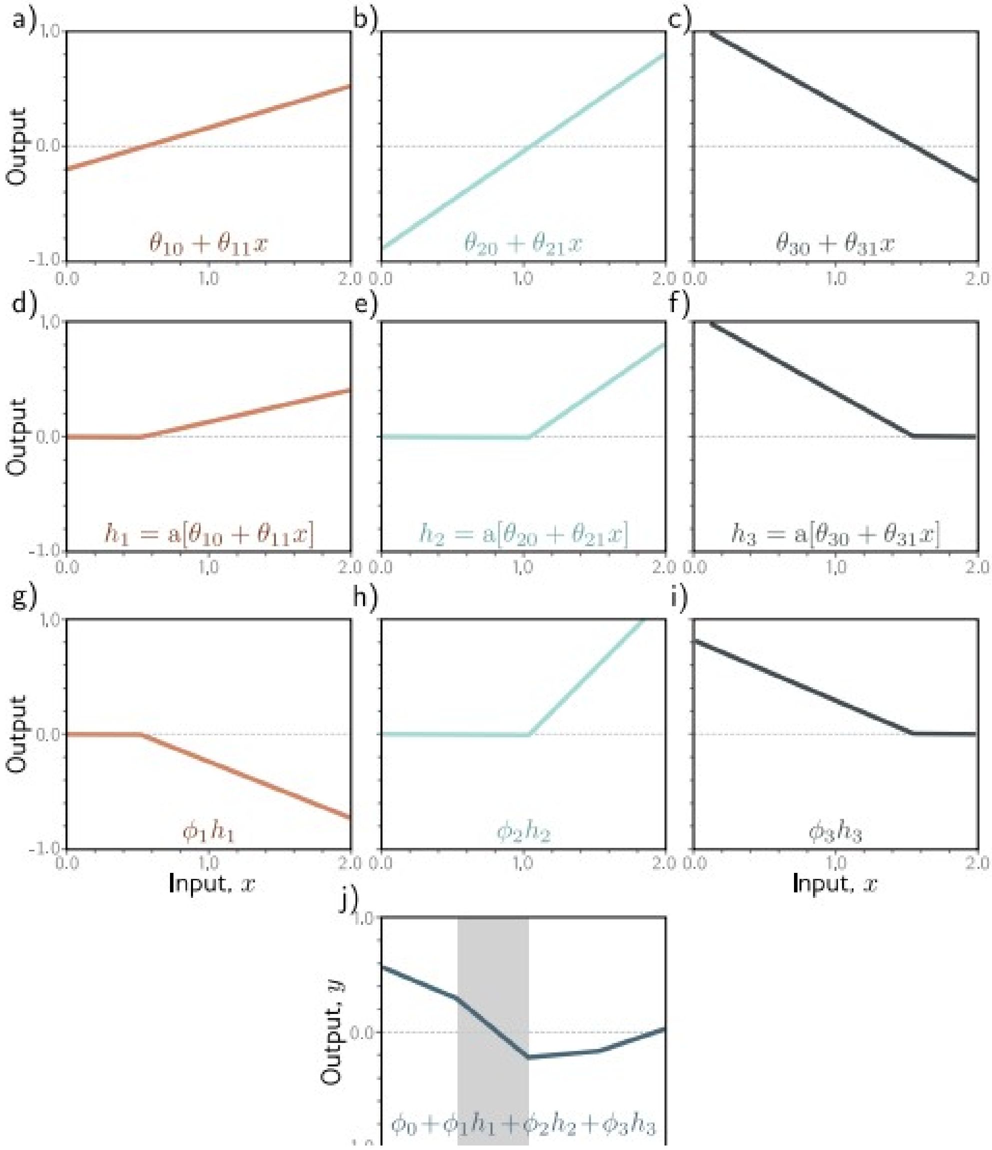

Figure 1: generalization of intersection

- Bias terms: $b_i \in \mathbb{R}$
- Output weights: $a_i \in \mathbb{R}$

The output of the network is given by:

$$f(\mathbf{x}) = \sum_{i=1}^{m} a_i \, \sigma(\mathbf{w}_i^T \mathbf{x} + b_i)$$

Step 2: Weight Vectors in Angular Coordinates
In spherical coordinates:

$$\mathbf{w} = \|\mathbf{w}\| \begin{bmatrix} \sin(\theta)\cos(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\theta) \end{bmatrix}$$

Step 3: Decision Boundary Condition
For each neuron, the decision boundary satisfies:

$$\mathbf{w}_i^T \mathbf{x} + b_i = 0,$$

which in spherical coordinates becomes:

$$\|\mathbf{w}_i\| \left[x_1 \sin(\theta_i)\cos(\phi_i) + x_2 \sin(\theta_i)\sin(\phi_i) + x_3 \cos(\theta_i)\right] + b_i = 0$$

Step 4: Intersection of Decision Boundaries
If two neurons intersect, we solve the system:

$$\mathbf{w}_i^T \mathbf{x} + b_i = 0, \quad \mathbf{w}_j^T \mathbf{x} + b_j = 0,$$

which translates to:

$$\|\mathbf{w}_i\| \mathbf{x} \cdot \mathbf{v}(\theta_i, \phi_i) + b_i = 0, \quad \|\mathbf{w}_j\| \mathbf{x} \cdot \mathbf{v}(\theta_j, \phi_j) + b_j = 0$$

Step 5: General Solution
The point of intersection $\mathbf{x}$ can be computed by solving the linear system:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b},$$

where $\mathbf{A}$ is the matrix formed by the weight directions in spherical coordinates, and $\mathbf{b}$ is the bias vector.

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

Solution:

Consider a shallow neural network with three hidden units and ReLU activations. Let the output y of the network be defined by the following equation:

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

where each hidden unit h_i is given by the ReLU activation function:

$$h_i = a(\theta_{i0} + \theta_{i1} x) = \max(0, \theta_{i0} + \theta_{i1} x)$$

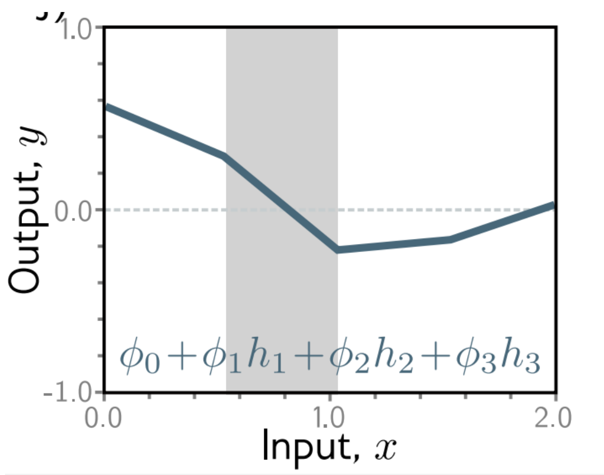

Figure 2: line equation

The output $y(x)$ is composed of four linear segments, which can be written as:

$$y(x) = \begin{cases} \phi_0, & x < x_1 \\ \phi_0 + \phi_1(\theta_{10} + \theta_{11}x), & x_1 \leq x < x_2 \\ \phi_0 + \phi_1(\theta_{10} + \theta_{11}x) + \phi_2(\theta_{20} + \theta_{21}x), & x_2 \leq x < x_3 \\ \phi_0 + \phi_1(\theta_{10} + \theta_{11}x) + \phi_2(\theta_{20} + \theta_{21}x) + \phi_3(\theta_{30} + \theta_{31}x), & x \geq x_3 \end{cases}$$

Explicitly, the four line segments are:

- First segment: $y = \phi_0$
- Second segment: $y = \phi_0 + \phi_1(\theta_{10} + \theta_{11}x)$
- Third segment: $y = \phi_0 + \phi_1(\theta_{10} + \theta_{11}x) + \phi_2(\theta_{20} + \theta_{21}x)$
- Fourth segment: $y = \phi_0 + \phi_1(\theta_{10} + \theta_{11}x) + \phi_2(\theta_{20} + \theta_{21}x) + \phi_3(\theta_{30} + \theta_{31}x)$

The activation thresholds x_1, x_2, and x_3 where each hidden unit is activated are given by:

$$x_i = -\frac{\theta_{i0}}{\theta_{i1}}, \quad \text{for each neuron.}$$

The output function combines the contributions of all active hidden units according to their weights and is expressed in the above piecewise form.

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

Solution:

Two-Output Feedforward Neural Network

We consider a feedforward neural network with:

- 2 input features: x_1, x_2
- D hidden neurons

- 2 output neurons: y_1, y_2
- Activation function $a(\cdot)$ for the hidden layer

1. Hidden Layer Computation

Each hidden unit h_d takes the input vector and applies a linear transformation followed by a nonlinear activation.

$$h_d = a\left(\theta_{d0} + \sum_{i=1}^{2} \theta_{di} x_i\right) \quad \text{for } d = 1, 2, \dots, D$$

Where:

- h_d: output of the d-th hidden unit
- θ_{d0}: bias term for hidden unit d
- θ_{di}: weight from input x_i to hidden unit d
- $a(\cdot)$: activation function (e.g., sigmoid, tanh, ReLU)

2. Output Layer Computation

Each output neuron performs a linear combination of all hidden unit outputs with a bias term:

$$y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d \quad \text{for } j = 1, 2$$

Where:

- y_j: output of the j-th output unit
- ϕ_{j0}: bias for output unit j
- ϕ_{jd}: weight from hidden unit d to output unit j

3. General equation of both Output

If one of the outputs is:

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

Then the other output is similarly given by:

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

Explanation:

- Both outputs depend on the same hidden layer outputs $h_1, h_2, \dots, h_D$
- The only difference lies in the weights ϕ_{jd} and biases ϕ_{j0} used for each output neuron
- This allows the network to produce different outputs from the same input vector through different weightings

4. Final Combined Form

$$y_1 = \phi_{10} + \sum_{d=1}^{D} \phi_{1d} \cdot a\left(\theta_{d0} + \sum_{i=1}^{2} \theta_{di} x_i\right)$$

$$y_2 = \phi_{20} + \sum_{d=1}^{D} \phi_{2d} \cdot a\left(\theta_{d0} + \sum_{i=1}^{2} \theta_{di} x_i\right)$$

Thus, both outputs are linear combinations of nonlinearly transformed weighted inputs.

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

$$\mathbf{x}_i \sim \mathcal{N}(\mu, \Sigma)$$

where μ is the unknown mean vector and Σ is the known covariance matrix.

Solution:

Maximum Likelihood Estimate of Unknown Mean Vector

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be independent and identically distributed (i.i.d.) vectors from a multivariate normal distribution:

$$\mathbf{x}_i \sim \mathcal{N}(\mu, \Sigma)$$

where μ is the unknown mean vector and Σ is the known covariance matrix.
The probability density function (PDF) of $\mathbf{x}_i$ is given by:

$$f(\mathbf{x}_i|\mu) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu)\right)$$

Likelihood Function

Given the independence of the samples, the likelihood function is the product of the individual densities:

$$L(\mu) = \prod_{i=1}^{n} f(\mathbf{x}_i|\mu)$$

Taking the natural logarithm of the likelihood function (log-likelihood):

$$\log L(\mu) = -\frac{np}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu)$$

Maximizing the Log-Likelihood

To find the MLE of μ, we differentiate $\log L(\mu)$ with respect to μ and set the result to zero:

$$\frac{\partial \log L}{\partial \mu} = \sum_{i=1}^{n} \Sigma^{-1}(\mathbf{x}_i - \mu) = 0$$

Simplifying:

$$\sum_{i=1}^{n}(\mathbf{x}_i - \mu) = 0 \implies \sum_{i=1}^{n}\mathbf{x}_i - n\mu = 0 \implies n\mu = \sum_{i=1}^{n}\mathbf{x}_i \implies \mu = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

Result: MLE of Mean Vector

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$

Thus, the maximum likelihood estimate of the unknown mean vector is the sample mean.

7. The Backpropagation for the cross-entropy loss function of a network of 3 outputs (f_1, f_2, f_3). Just assume that the 3 outputs are the only parameters of the loss function.

Solution:

Cross-Entropy Loss with Softmax Outputs

Let the outputs (logits) of a neural network be f_1, f_2, f_3. These are passed through the softmax function to produce probabilities:

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

Let the ground-truth target vector be one-hot encoded: $\mathbf{y} = (y_1, y_2, y_3)$, where $y_k = 1$ for the correct class and $y_i = 0$ for $i \neq k$.
The cross-entropy loss is given by:

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

Since only one $y_k = 1$, the expression simplifies to:

$$L = -\log(p_k)$$

Computing the Gradient using the Chain Rule

We want to compute the gradient of the loss with respect to the logits f_i i.e. one of the output. Using the chain rule we got:

$$\frac{\partial L}{\partial f_i} = \sum_{j=1}^{3} \frac{\partial L}{\partial p_j} \cdot \frac{\partial p_j}{\partial f_i}$$

Step 1: Compute $\frac{\partial L}{\partial p_j}$

$$\frac{\partial L}{\partial p_j} = -\frac{y_j}{p_j}$$

Step 2: Compute $\frac{\partial p_j}{\partial f_i}$ From the derivative of softmax:

$$\frac{\partial p_j}{\partial f_i} = p_j(\delta_{ij} - p_i)$$

Step 3: Combine Using Chain Rule

$$\frac{\partial L}{\partial f_i} = \sum_{j=1}^{3} \left(-\frac{y_j}{p_j}\right) \cdot p_j(\delta_{ij} - p_i) = -\sum_{j=1}^{3} y_j(\delta_{ij} - p_i)$$

Since $y_j = 1$ only for $j = k$, the summation simplifies:

$$\frac{\partial L}{\partial f_i} = -(\delta_{ik} - p_i) = p_i - y_i$$

Final Result:

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

Conclusion

This is a well-known result for the gradient of the softmax with cross-entropy loss. It tells us how to adjust each logit during gradient descent.

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:

Network Architecture

- **Input layer**: 2 features (x_1, x_2)
- **Hidden layer**: 2 neurons with sigmoid activation
- **Output layer**: 3 neurons with softmax activation (for 3-class classification)
- **Loss function**: Cross-entropy

Notation:

- $W^{[1]} \in \mathbb{R}^{2\times 2}$: weights from input to hidden layer
- $b^{[1]} \in \mathbb{R}^2$: biases for hidden layer
- $z^{[1]} \in \mathbb{R}^2$: pre-activation of hidden layer
- $a^{[1]} \in \mathbb{R}^2$: activation of hidden layer (after sigmoid)
- $W^{[2]} \in \mathbb{R}^{3\times 2}$: weights from hidden to output layer
- $b^{[2]} \in \mathbb{R}^3$: biases for output layer
- $z^{[2]} \in \mathbb{R}^3$: pre-activation of output layer
- $\hat{y} \in \mathbb{R}^3$: predicted output probabilities (softmax)
- $y \in \mathbb{R}^3$: true label (one-hot vector)

Forward Pass Equations

1. Hidden Layer Computation Let:

$$W^{[1]} \in \mathbb{R}^{2\times 2}, \quad b^{[1]} \in \mathbb{R}^2$$

Then:

$$z^{[1]} = W^{[1]}x + b^{[1]}, \quad a^{[1]} = \sigma(z^{[1]})$$

2. Output Layer Computation Let:

$$W^{[2]} \in \mathbb{R}^{3\times 2}, \quad b^{[2]} \in \mathbb{R}^3$$

Then:

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \quad \hat{y} = \text{softmax}(z^{[2]})$$

Loss Function

For a true class vector $y \in \mathbb{R}^3$ (one-hot encoded):

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

Backward Pass (Gradient Computation)

1. Gradient w.r.t. Output Layer (Softmax + Cross Entropy)

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

2. Gradients for Output Layer Weights and Biases

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = (\hat{y} - y)\left(a^{[1]}\right)^T \quad , \quad \frac{\partial \mathcal{L}}{\partial b^{[2]}} = \hat{y} - y$$

3. Backpropagate to Hidden Layer

$$\delta^{[1]} = \left((W^{[2]})^T(\hat{y} - y)\right) \circ \sigma'(z^{[1]})$$

Where $\circ$ denotes element-wise multiplication and:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

4. Gradients for Hidden Layer Weights and Biases

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

Conclusion

The above steps detail full backpropagation for a 3-class classification network with 2 inputs, 2 hidden sigmoid units, and a softmax output layer. These gradients can now be used to update parameters using gradient descent or advanced optimizers such as Adam.