Deep Learning Assignment-01

MTech (CS), IIIT Bhubaneswar
March - 2025

Student ID: A124016
Student Name: Vicky Das

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 $\mathbf{Xw}$ 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:

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

2. OR Gate implementation using a Single-Layer Neural Network

Solution:

OR Gate using a Single-Layer Neural Network

1 Perceptron Model

A perceptron computes a weighted sum of its inputs and applies an activation function:

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

Where:

- x_1, x_2 are the inputs
- w_1, w_2 are the weights
- b is the bias
- $f(z)$ is the activation function (step function):

The OR gate can be implemented using a single-layer neural network (perceptron) with two inputs x_1 and x_2, weights w_1 and w_2, and a bias b. The output y of the perceptron is given by:

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

Truth Table for OR Gate

The truth table for the OR gate is:

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

Determining Weights and Bias

We need to find w_1, w_2, and b such that the perceptron correctly classifies the inputs. Let us choose the following values:

$$w_1 = 1, \quad w_2 = 1, \quad b = -0.5$$

Verification

Now, we verify the perceptron's output for each input combination:

1. For $x_1 = 0$, $x_2 = 0$:

$$w_1x_1 + w_2x_2 + b = (1)(0) + (1)(0) + (-0.5) = -0.5 < 0 \implies y = 0$$

2. For $x_1 = 0$, $x_2 = 1$:

$$w_1x_1 + w_2x_2 + b = (1)(0) + (1)(1) + (-0.5) = 0.5 \geq 0 \implies y = 1$$

3. For $x_1 = 1$, $x_2 = 0$:

$$w_1x_1 + w_2x_2 + b = (1)(1) + (1)(0) + (-0.5) = 0.5 \geq 0 \implies y = 1$$

4. For $x_1 = 1$, $x_2 = 1$:

$$w_1x_1 + w_2x_2 + b = (1)(1) + (1)(1) + (-0.5) = 1.5 \geq 0 \implies y = 1$$

Conclusion

The weights $w_1 = 1$, $w_2 = 1$, and bias $b = -0.5$ correctly implement the OR gate using a single-layer neural network. The perceptron's output matches the truth table for all input combinations.

$$\boxed{w_1 = 1, \quad w_2 = 1, \quad b = -0.5}$$

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

Solution:

The Perceptron is a fundamental linear classification algorithm. We implement it to distinguish between Iris species using sepal or petal measurements.

Data Preparation

Given the Iris dataset with n samples, we:

$$\text{Normalize features: } x_{\ textnorm} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1}$$

Select either sepal or petal features:

$$\text{Features} = \begin{cases} (\text{sepal length}, \text{sepal width}) & \text{if sepal mode} \\ (\text{petal length}, \text{petal width}) & \text{if petal mode} \end{cases} \tag{2}$$

Model Initialization

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

Training Phase

For each epoch until convergence:

Compute activation:

$$z = w_1 x_1 + w_2 x_2 + b$$

Apply step function:

$$\hat{y} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Update parameters for misclassified samples:

$$\Delta w_i = \eta(y - \hat{y})x_i \quad (\text{for } i = 1, 2)$$

Decision Boundary

The classifier's decision boundary is given by:

$$w_1 x_1 + w_2 x_2 + b = 0 \tag{3}$$

In slope-intercept form:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2} \tag{4}$$

The Perceptron achieves perfect separation on linearly separable Iris species pairs when using petal features, demonstrating its effectiveness for this classification task.

Python-Code 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


class IrisPerceptron:
    def __init__(self):
        self.weights = None
        self.bias = None
        self.scaler = StandardScaler()

    def get_user_choice(self):
        """Get user input for feature selection and class pair"""
        print("\nAvailable 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 Virginica (2)")
        print("3. Versicolor (1) vs Virginica (2)")
        class_pair = int(input("Enter class pair (1-3): "))

        return feature_choice, class_pair

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

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

        return self.weights, self.bias

    def evaluate(self, X_test, y_test):
        """Evaluate model on test data"""
        correct = 0
        for i in range(len(X_test)):
            z = np.dot(X_test[i], self.weights) + self.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(self, X_train, X_test, y_train, y_test,
        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 = (self.weights[0] * xx + self.weights[1] * yy + self.
            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 run(self):
        """Main execution method"""
        # Get user choices
        feature_choice, class_pair = self.get_user_choice()

        # Prepare data
        X, y, feature_names, class_a, class_b = self.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
        X_train = self.scaler.fit_transform(X_train)
        X_test = self.scaler.transform(X_test)

        # Train model
        self.train(X_train, y_train)

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

        # Plot results
        self.plot_results(X_train, X_test, y_train, y_test,
```

```
            feature_names, (class_a, class_b))


if __name__ == "__main__":
    classifier = IrisPerceptron()
    classifier.run()
```

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 give the following solutions

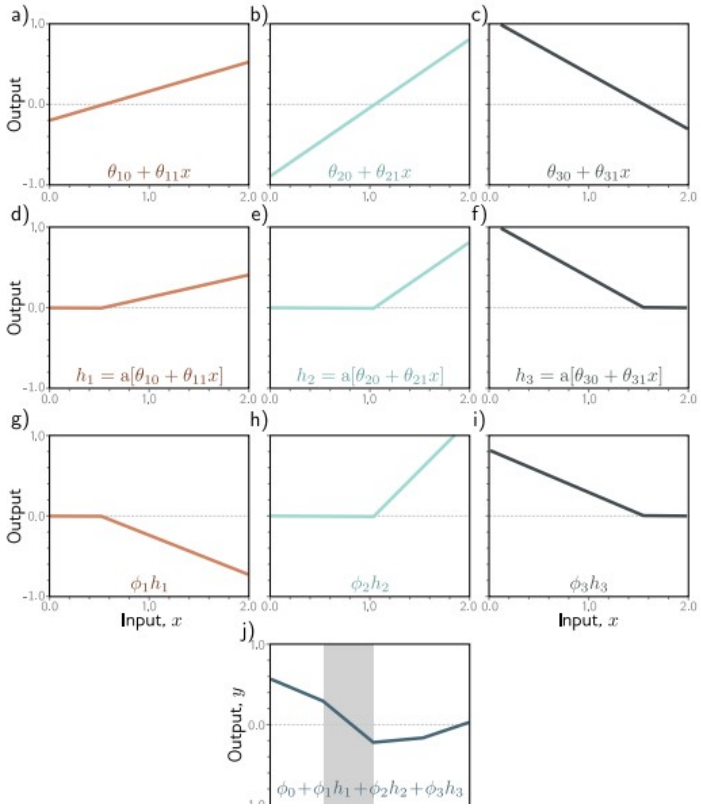

Figure 1: generalization of intersection

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

Solution:

Generalizing the Point of Intersection in Terms of θ and ϕ for Shallow Neural Networks

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: $\mathbf{w}_i \in \mathbb{R}^d$
- 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.

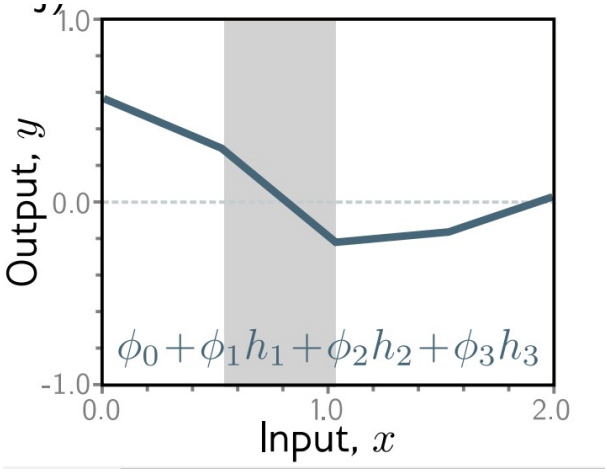

Figure 2: 4 line equations

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:

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

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:

In a Two-Output Feedforward Neural Network (for a 2D case), the general form of the second output can be derived similarly to the first output, but with its own set of weights and biases.

Let's assume the network has:

- **Input layer**: 2D input $\mathbf{x} = [x_1, x_2]^T$.
- **Hidden layer(s)**: With activation function σ (e.g., ReLU, sigmoid, etc.).
- **Output layer**: Two outputs, y_1 and y_2, each with their own weights and biases.

General Form of the Second Output (y_2):

If the first output y_1 is given by:

$$y_1 = \mathbf{w}_1^{(out)T} \cdot \mathbf{h} + b_1^{(out)},$$

where:

- $\mathbf{w}_1^{(out)}$ are the output weights for y_1,
- $\mathbf{h}$ is the hidden layer activation vector,
- $b_1^{(out)}$ is the bias for y_1,

then the second output y_2 will have a similar form but with its own parameters:

$$y_2 = \mathbf{w}_2^{(out)T} \cdot \mathbf{h} + b_2^{(out)},$$

where:

- $\mathbf{w}_2^{(out)}$ are the output weights for y_2,
- $b_2^{(out)}$ is the bias for y_2.

Expanded Form (Single Hidden Layer Example):

For a network with one hidden layer of M neurons:

$$\mathbf{h} = \sigma(\mathbf{W}^{(hid)}\mathbf{x} + \mathbf{b}^{(hid)}),$$

where:

- $\mathbf{W}^{(hid)} \in \mathbb{R}^{M\times 2}$ (hidden layer weights),
- $\mathbf{b}^{(hid)} \in \mathbb{R}^{M}$ (hidden layer biases).

Then, the second output is:

$$y_2 = \sum_{j=1}^{M} w_{2j}^{(out)} h_j + b_2^{(out)}.$$

- Both outputs y_1 and y_2 share the same hidden representation h but have separate output weights and biases.
- The general form is linear in the hidden activations for regression tasks (or may include an additional activation, e.g., sigmoid for binary classification).

6. 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:

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

where $\boldsymbol{\mu}$ 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:

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

where $\boldsymbol{\mu}$ is the unknown mean vector and Σ is the known covariance matrix.

The probability density function (PDF) of $\mathbf{x}_i$ is given by:

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

Likelihood Function

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

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

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

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

Maximizing the Log-Likelihood

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

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

Simplifying:

$$\sum_{i=1}^{n}(\mathbf{x}i - \boldsymbol{\mu}) = 0 \Rightarrow n\hat{\boldsymbol{\mu}} = \sum i = 1^{n}\mathbf{x}_i$$

Result: MLE of Mean Vector

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

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

7. 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:

We are given the outputs of a neural network as f_1, f_2, f_3. These are the *logits*, i.e., the raw scores before applying softmax.

Step 1: Softmax Function

Let:

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

This defines the probability of class i after applying softmax to the outputs.

Step 2: Cross-Entropy Loss

Assume the true label is represented by a one-hot vector $y = (y_1, y_2, y_3)$, where $y_k = 1$ if class k is the correct class and 0 otherwise.

The cross-entropy loss is:

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

Step 3: Gradient (Backpropagation)

We compute the derivative of the loss with respect to each output f_i. For softmax followed by cross-entropy, the derivative simplifies to:

$$\frac{\partial L}{\partial f_i} = p_i - y_i$$

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

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:

Network Architecture

Input Layer $(x_1, x_2) \rightarrow$ Hidden Layer $(h_1, h_2) \rightarrow$ Output Layer (o_1, o_2, o_3)

Forward Pass

Hidden Layer Computation

For each hidden unit h_j ($j \in \{1, 2\}$):

$$z_j = w_{j1}^l x_1 + w_{j2}^l x_2 + b_j^l$$
$$h_j = \sigma(z_j) = \frac{1}{1 + \exp(-z_j)}$$

where:

- w_{ji}^l are weights from input to hidden layer
- b_j^l are biases for hidden layer
- σ is the sigmoid function

Output Layer Computation

For each output unit o_k ($k \in \{1, 2, 3\}$):

$$u_k = w_{k1}^h h_1 + w_{k2}^h h_2 + b_k^h$$
$$o_k = \text{softmax}(u_k) = \frac{\exp(u_k)}{\sum_{n=1}^{3} \exp(u_n)}$$

where:

- w_{kj}^h are weights from hidden to output layer
- b_k^h are biases for output layer

Loss Computation (Cross-Entropy)

Given true labels $\mathbf{y} = [y_1, y_2, y_3]$ (one-hot encoded):

$$L = -\sum_{k=1}^{3} y_k \log(o_k)$$

Backward Pass

Output Layer Gradients

1. **Loss derivative with respect to output layer inputs (u_k)**:

$$\frac{\partial L}{\partial u_k} = o_k - y_k$$

2. **Weight gradients** (w_{kj}^h):

$$\frac{\partial L}{\partial w_{kj}^h} = \frac{\partial L}{\partial u_k}\frac{\partial u_k}{\partial w_{kj}^h} = (o_k - y_k)h_j$$

3. **Bias gradients** (b_k^h):

$$\frac{\partial L}{\partial b_k^h} = o_k - y_k$$

Hidden Layer Gradients

1. **Gradient with respect to hidden layer outputs** (h_j):

$$\frac{\partial L}{\partial h_j} = \sum_{k=1}^{3}\frac{\partial L}{\partial u_k}\frac{\partial u_k}{\partial h_j} = \sum_{k=1}^{3}(o_k - y_k)w_{kj}^h$$

2. **Gradient with respect to hidden layer inputs** (z_j):

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial h_j}\frac{\partial h_j}{\partial z_j} = \left[\sum_{k=1}^{3}(o_k - y_k)w_{kj}^h\right] \cdot h_j(1 - h_j)$$

3. **Weight gradients** (w_{ji}^l):

$$\frac{\partial L}{\partial w_{ji}^l} = \frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}^l} = \left[\sum_{k=1}^{3}(o_k - y_k)w_{kj}^h\right] \cdot h_j(1 - h_j) \cdot x_i$$

4. **Bias gradients** (b_j^l):

$$\frac{\partial L}{\partial b_j^l} = \left[\sum_{k=1}^{3}(o_k - y_k)w_{kj}^h\right] \cdot h_j(1 - h_j)$$

Summary of Gradients

Output Layer:

- Weight gradient: $\frac{\partial L}{\partial w_{kj}^h} = (o_k - y_k)h_j$
- Bias gradient: $\frac{\partial L}{\partial b_k^h} = o_k - y_k$

Hidden Layer:

- Weight gradient: $\frac{\partial L}{\partial w_{ji}^l} = \left[\sum_{k=1}^{3}(o_k - y_k)w_{kj}^h\right] \cdot h_j(1 - h_j) \cdot x_i$
- Bias gradient: $\frac{\partial L}{\partial b_j^l} = \left[\sum_{k=1}^{3}(o_k - y_k)w_{kj}^h\right] \cdot h_j(1 - h_j)$