Artificial Neural Networks (ANN) - A Beginner's Guide

1 Introduction

Artificial Neural Networks (ANNs) are a fundamental concept in deep learning, consisting of multiple layers of neurons that process and learn from data. They are useful for tasks such as classification, regression, and pattern recognition.

2 Architecture of ANN

An ANN typically consists of the following layers:

2.1 Input Layer

- Accepts input features in a structured format. - In this implementation, we assume an input size of 784 (28x28 images flattened for MNIST dataset).

2.2 Hidden Layers

- Contain neurons that learn from the input data through weighted connections.
- Activation functions introduce non-linearity to help model complex patterns.
- Common activation functions:

- **ReLU**: $f(x) = \max(0, x)$
- **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Tanh**: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

2.3 Output Layer

- Produces predictions based on the learned weights and biases. - Uses softmax function to compute class probabilities:

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1}$$

3 Implementation of ANN in Python

Below is a simple implementation of an ANN from scratch using NumPy.

3.1 ANN Model

```
import numpy as np
from tool import cross_entropy_loss, d_cross_entropy_loss
from util import relu, d_relu, softmax

class ANN:
    def __init__(self, input_size=784, hidden1_size=128, hidden2_size=64, output_size=10):
        # Initialize weights and biases
        self.W1 = np.random.randn(input_size, hidden1_size) * 0.01
        self.b1 = np.zeros((1, hidden1_size))
        self.W2 = np.random.randn(hidden1_size, hidden2_size) * 0.01
        self.b2 = np.zeros((1, hidden2_size))
        self.W3 = np.random.randn(hidden2_size, output_size) * 0.01
        self.b3 = np.zeros((1, output_size))

    def forward(self, X):
        """Performs forward propagation."""
        self.X = X.reshape(1, -1)  # Flatten input

        self.Z1 = np.dot(self.X, self.W1) + self.b1
        self.A1 = relu(self.Z1)

        self.Z2 = np.dot(self.A1, self.W2) + self.b2
        self.A2 = relu(self.Z2)

        self.Z3 = np.dot(self.A2, self.W3) + self.b3
        self.A3 = softmax(self.Z3)

        return self.A3

def train(self, X_train, y_train, epochs=10, lr=0.01):
    for epoch in range(epochs):
        loss = 0
        for X, y in zip(X_train, y_train):
            output = self.forward(X)
            loss += cross_entropy_loss(y, output)
            self.backward(y, lr)
        print(f"Epoch {epoch+1}, Loss: {loss/len(X_train)}")
```

4 Key Takeaways

- ANNs consist of an input layer, hidden layers, and an output layer. - Activation functions introduce non-linearity, enabling the network to learn complex patterns. - Training involves forward propagation, loss computation, backpropagation, and weight updates. - The softmax function is used in the final layer for classification tasks.

5 Important Equations

1. **ReLU Activation Function:**

$$f(x) = \max(0, x) \tag{2}$$

2. **Softmax Function:**

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3}$$

3. **Cross-Entropy Loss:**

$$L = -\sum_i y_i \log(\hat{y}_i) \tag{4}$$