Python

Training Module

Artificial Intelligence
Training Material

Introduction to Neural Networks

1.1 What are Neural Networks?

- **Definition**: Neural networks are computational models inspired by the human brain. They consist of layers of interconnected "neurons," where each neuron performs a computation based on the input data it receives.
- **Biological Inspiration**: The structure of artificial neural networks is inspired by biological neurons that receive signals from other neurons, process them, and send out an output.
- **Neurons in ANN**: In a neural network, a neuron receives inputs, applies weights to them, and then passes them through an activation function to produce an output.

1.2 Applications of Neural Networks

- **Pattern Recognition**: Neural networks are widely used in tasks like image recognition, speech recognition, and handwriting analysis.
- **Classification**: Identifying which category an input belongs to (e.g., spam vs. not spam emails).
- **Regression**: Predicting continuous values, such as stock prices, from historical data.
- **Reinforcement Learning**: In complex environments like robotics or game AI, neural networks help agents make decisions.

1.3 History of Neural Networks

- **Early Beginnings**: Perceptron (1950s), the first type of neural network, was introduced by Frank Rosenblatt.
- **Challenges & Resurgence**: Neural networks faced challenges in the 1970s-1990s due to limited computational power and data. However, deep learning (a type of neural network with multiple layers) gained attention in the 2000s with advances

in hardware (GPUs) and large datasets.

---

2. The Perceptron

2.1 What is a Perceptron?

- **Definition**: A perceptron is the simplest form of a neural network and is a single-layer neural network used for binary classification tasks.
- **Structure**: It consists of an input layer, weights for each input, a bias term, and an activation function (usually a step function).
    - **Input**: A vector of values (features).
    - **Weights**: A set of weights that are multiplied with the input features.
    - **Bias**: An additional term that helps the model to make predictions even when all the inputs are zero.
    - **Activation Function**: A function that decides whether the neuron should fire based on the weighted sum of inputs.

Perceptron Working Mechanism:

1. **Input and Weights**: The perceptron receives an input vector x=(x1,x2,...,xn)x = (x_1, x_2, ..., x_n)x=(x1,x2,...,xn), each corresponding to a feature. The weights w=(w1,w2,...,wn)w = (w_1, w_2, ..., w_n)w=(w1,w2,...,wn) are learned by the model.
2. **Weighted Sum**: The perceptron calculates the weighted sum of inputs: z=w1x1+w2x2+...+wnxn+bz = w_1 x_1 + w_2 x_2 + ... + w_n x_n + bz=w1x1+w2x2+...+wnxn+b, where bbb is the bias term.
3. **Activation**: The output yyy is determined by passing the weighted sum zzz through an activation function (e.g., step function): y=step(z)={1if z≥00if z<0y = \text{step}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}y=step(z)={10if z≥0if z<0

2.2 Training a Perceptron

- **Objective**: The goal is to adjust the weights and bias so that the perceptron makes correct predictions.
- **Learning Rule (Perceptron Learning Algorithm)**:
  - The perceptron learning algorithm uses a **supervised learning** approach where the weights are updated iteratively.
  - **Error Calculation**: For each training example, the error is calculated as: Error=Target Output−Predicted Output\text{Error} = \text{Target Output} - \text{Predicted Output}Error=Target Output−Predicted Output
  - **Weight Update**: Weights are updated using the following rule: wi=wi+Δwi=wi+η·(ytarget−ypred)·xiw_i = w_i + \Delta w_i = w_i + \eta \cdot (y_{\text{target}} - y_{\text{pred}}) \cdot x_iwi=wi+Δwi=wi +η·(ytarget−ypred)·xi where η\etaη is the learning rate, ytargety_{\text{target}}ytarget is the actual target, and ypredy_{\text{pred}}ypred is the predicted output.

2.3 Limitations of the Perceptron

- **Linear Separability**: The perceptron can only solve linearly separable problems (i.e., problems where data points can be divided into two classes by a straight line or hyperplane).
- **Example**: The XOR problem is a classic example that a perceptron cannot solve because the classes are not linearly separable.

---

3. Artificial Neural Networks (ANNs)

3.1 Introduction to Artificial Neural Networks

- **Definition**: ANNs are networks of neurons arranged in layers, where each neuron performs a weighted sum of inputs, followed by an activation function.

- **Multi-Layer Perceptron (MLP)**: The most basic type of neural network that consists of:
    - **Input Layer**: Takes in the input features.
    - **Hidden Layers**: One or more layers of neurons where computations occur.
    - **Output Layer**: Produces the final predictions.
- **Why Multi-Layer Networks?**: Multi-layer networks can learn complex, non-linear relationships in data, unlike a single-layer perceptron which can only solve linearly separable problems.

3.2 Components of ANNs

- **Neurons**: Each neuron in a layer receives inputs from the previous layer, performs a weighted sum of the inputs, adds a bias term, and passes the result through an activation function.
- **Weights and Biases**: Both weights and biases are parameters learned during training. Weights control the strength of connections between neurons, while biases help adjust the output of neurons.
- **Activation Function**: Introduces non-linearity into the model. Common activation functions include:
    - **Sigmoid**: σ(x)=11+e−x\sigma(x) = \frac{1}{1 + e^{-x}}σ(x)=1+e−x1, outputs values between 0 and 1.
    - **ReLU (Rectified Linear Unit)**: ReLU(x)=max(0,x)\text{ReLU}(x) = \max(0, x)ReLU(x)=max(0,x), popular for deep networks.
    - **Tanh**: Hyperbolic tangent, outputs values between -1 and 1.
    - **Softmax**: Used in multi-class classification problems to output probability distributions.

3.3 Feedforward Propagation

- **Forward Pass**: In this phase, inputs propagate through the network:
  - The input layer sends data to the first hidden layer, which computes weighted sums, applies activation functions, and passes it to the next layer.
  - The process continues until the final output layer produces predictions.

3.4 Backpropagation and Training

- **Objective**: Adjust the weights and biases to minimize the error between predicted and actual values.
- **Backpropagation**:
  - **Loss Function**: A function that calculates the error between the network's predictions and the actual target values. Common loss functions include mean squared error (MSE) for regression and cross-entropy loss for classification.
  - **Gradient Descent**: The most common optimization algorithm for training ANNs. It involves adjusting weights in the opposite direction of the gradient of the loss function with respect to the weights.
  - **Backpropagation Algorithm**: During backpropagation, the error is propagated backward from the output layer to the input layer, and the weights are updated using the gradients calculated during this process.
    - The gradients are computed using the **chain rule of calculus**.

3.5 Types of Neural Networks

- **Feedforward Neural Networks (FNN)**: The basic form of ANN where
- information moves in one direction—from input to output.
- **Convolutional Neural Networks (CNN)**: Special type of neural network used primarily for image data.

- **Recurrent Neural Networks (RNN)**: Neural networks designed for sequential data like text or time series.

---

4. Training Deep Neural Networks

4.1 Challenges in Training Deep Networks

- **Vanishing and Exploding Gradients**: In deep networks, gradients can become extremely small or large, making it difficult to update weights properly. Techniques like **ReLU activation** and **batch normalization** can help alleviate this issue.
- **Overfitting**: Deep networks can memorize the training data instead of learning general patterns. Techniques like **dropout**, **L2 regularization**, and **early stopping** are used to prevent overfitting.

4.2 Optimization Techniques

- **Gradient Descent Variants**:
    - **Stochastic Gradient Descent (SGD)**: Updates weights using one training example at a time.
    - **Mini-Batch Gradient Descent**: Combines the benefits of batch gradient descent and SGD.
    - **Adam Optimizer**: An adaptive learning rate optimization method that adjusts learning rates for each parameter.
