Problem Set 3: Neural Networks

Ankit Jambhulkar

November 17, 2024

Overview

This problem set explores foundational concepts of neural networks, including architecture, activation functions, forward propagation, regularization, and adaptive learning. Both theoretical explanations and coding implementations are included. The Python script accompanies this report.

1 Basic Architecture of a Neural Network

A neural network consists of three main components:

- **Input Layer**: Receives input data.
- **Hidden Layers**: Perform transformations using weights, biases, and activation functions.
- **Output Layer**: Produces the final prediction or classification.

The mathematical representation of a neuron in a layer is:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

where:

- x_i: Inputs to the neuron.
- w_i: Weights associated with each input.
- b: Bias term.
- z: Output of the neuron before applying an activation function.

2 Activation Functions

Activation functions introduce non-linearity into neural networks, enabling them to learn complex patterns.

Examples of Activation Functions

- **Sigmoid**:
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
Useful for binary classification.

- **ReLU (Rectified Linear Unit)**:
$$\text{ReLU}(x) = \max(0, x)$$
Outputs zero for negative inputs and the input itself for positive inputs.

- **Leaky ReLU**:
$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$
Where α is a small constant (e.g., 0.01). It avoids the "dying ReLU" problem.

Plots of Activation Functions

The plots of activation functions are shown below:

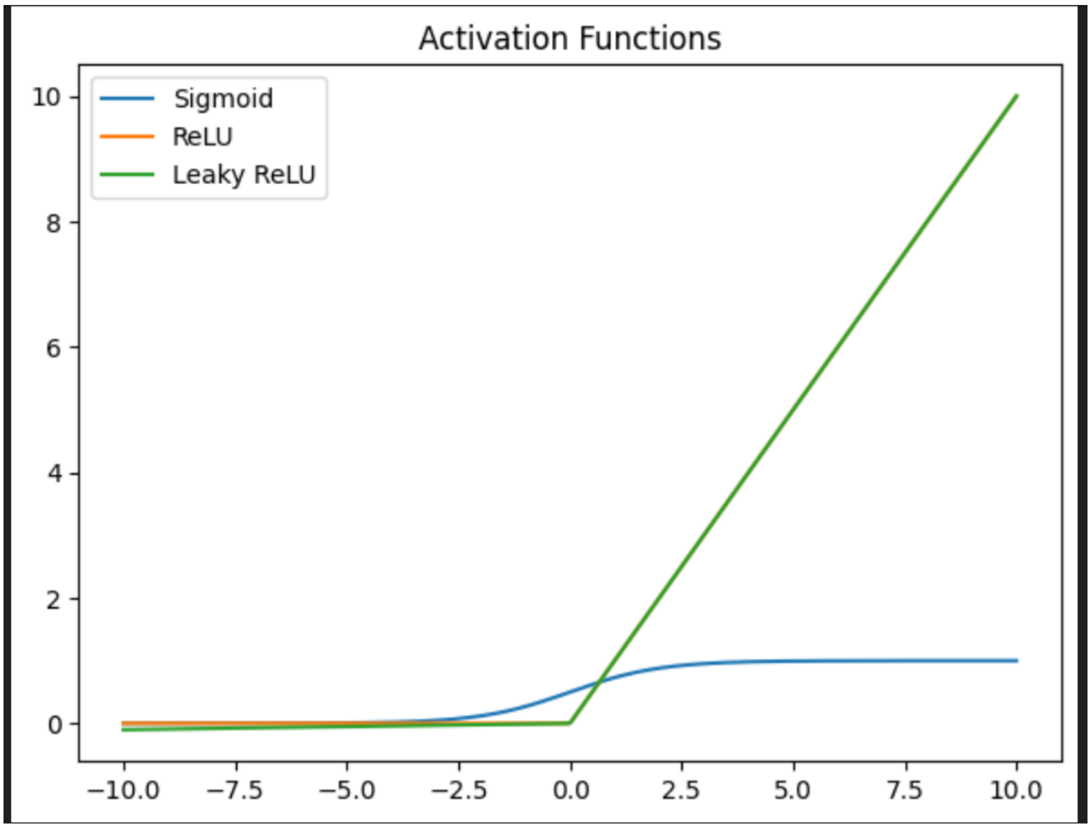

Figure 1: Plots of Activation Functions

3 Forward Propagation

Forward propagation computes the output of the neural network layer by layer. For a 3-layer network:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \phi(z^{(l)})$$

where:

- $W^{(l)}$: Weight matrix of layer l.
- $a^{(l)}$: Activation of layer l.
- $b^{(l)}$: Bias vector of layer l.
- ϕ: Activation function.

4 Overfitting and Regularization

Overfitting occurs when a model performs well on training data but poorly on unseen data. Regularization methods reduce overfitting.

L2 Regularization

Adds a penalty to the loss function based on the sum of squared weights:

$$\text{Loss}_{\text{reg}} = \text{Loss} + \lambda \sum_{i=1}^{n} w_i^2$$

where λ controls the regularization strength.

5 Training a Neural Network

Training involves updating weights and biases to minimize the loss function.

Gradient Descent

Gradient descent updates parameters iteratively:

$$w \leftarrow w - \eta \frac{\partial \text{Loss}}{\partial w}$$

where:

- w: Weight to be updated.
- η: Learning rate (step size).
- $\frac{\partial \text{Loss}}{\partial w}$: Gradient of the loss with respect to w.

6 Adaptive Learning Rates

Adaptive learning methods like Adam optimize gradient descent by adjusting the learning rate. The update rule is:

$$w_t \leftarrow w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

where:

- $\hat{m}_t, \hat{v}_t$: Estimates of the first and second moments of gradients.
- ϵ: Small constant to prevent division by zero.

Conclusion

This report covered the theoretical aspects of neural networks, their components, and training techniques. The Python script includes all coding exercises, including activation functions, forward propagation, and optimization methods.