Problem Set 3: Neural Networks

Po Yu Chen

November 24, 2024

1 Introduction

In this assignment, we explore the concept of activation functions, which play a critical role in neural networks by introducing non-linearity. Three commonly used activation functions—Sigmoid, ReLU, and Leaky ReLU—are compared and analyzed to understand their behavior across a range of inputs. By visualizing these functions, we aim to better understand their suitability for different types of neural network layers and how they influence the learning process.

2 Methodology

2.1 Neural Network Architecture

A neural network is composed of multiple layers of neurons: an input layer, hidden layer(s), and an output layer. Each neuron receives inputs, processes them, and passes the result to the next layer.

Neural Network Components

- **Input Layer:** The input layer consists of neurons that receive the input features from the dataset. Each neuron in this layer represents one feature.
- **Hidden Layers:** These are the intermediate layers between the input and output layers. Hidden layers apply transformations to the inputs using learned weights and activation functions.
- **Output Layer:** The output layer produces the final result of the neural network, such as a classification label or a regression output.

Each neuron in a layer receives inputs from the previous layer, processes them (typically using an activation function), and passes the resulting value to the next layer. This layered approach allows the neural network to learn complex patterns from the data.

2.2 Activation Functions

Activation functions introduce non-linearity into the network, allowing it to learn more complex patterns. Below are some of the most common activation functions used in neural networks:

Common Activation Functions

- **Sigmoid Function**: The sigmoid function has an S-shaped curve and is commonly used for binary classification tasks. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **ReLU (Rectified Linear Unit)**: ReLU outputs zero for negative inputs and passes the positive inputs unchanged. It is defined as:

$$f(x) = \max(0, x)$$

- **Leaky ReLU**: Similar to ReLU, but with a small gradient for negative inputs. This helps to prevent the "dying ReLU" problem. It is defined as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

  where α is a small positive constant, typically set to 0.01.

2.3 Forward Propagation

Forward propagation is the process of passing inputs through the network to generate an output. This process involves computing the weighted sum of inputs and applying activation functions across multiple layers.

Mathematical Representation

Consider a 3-layer neural network with an input layer, two hidden layers, and an output layer. Let:

- X be the input to the network.
- $W^{[l]}$ and $b^{[l]}$ represent the weights and biases for layer l.
- $Z^{[l]}$ be the linear combination of inputs for layer l.
- $A^{[l]}$ be the activation output for layer l.

The forward propagation steps are as follows:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = f^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = f^{[2]}(Z^{[2]})$$

$$Z^{[3]} = W^{[3]}A^{[2]} + b^{[3]}$$
$$A^{[3]} = f^{[3]}(Z^{[3]})$$

Where $f^{[l]}$ represents the activation function for layer l.

2.4 Overfitting and Regularization

Overfitting

Overfitting occurs when a model learns the noise in the training data rather than the actual patterns. This often results in a model that performs well on training data but poorly on unseen test data, as it fails to generalize.

Regularization

Regularization helps control overfitting and allows the model to generalize better. Below are some common regularization techniques:

L2 Regularization

L2 Regularization adds a penalty to the loss function based on the sum of squared weights. The regularized loss function can be expressed as:

$$J(\theta) = J_{\text{original}}(\theta) + \frac{\lambda}{2m}\sum_{i=1}^{n}\theta_i^2$$

where:

- $J(\theta)$ is the regularized loss function.
- $J_{\text{original}}(\theta)$ is the original loss function.
- λ is the regularization parameter.
- m is the number of samples.
- θ_i are the model parameters.

Dropout

Dropout is a technique where, during training, some neurons are randomly ignored ("dropped"). This helps prevent the model from relying too heavily on any one feature, thereby reducing overfitting. Each neuron is retained with a probability p:

$$\text{keep probability} = p$$

During testing, all neurons are active, but the output is scaled by the keep probability used during training to maintain consistency.

2.5 Training Neural Network

Training a neural network involves adjusting the weights and biases to minimize the loss function. This process allows the model to better fit the data and make accurate predictions.

Gradient Descent

A popular method for minimizing the loss is **gradient descent**. The idea behind gradient descent is to iteratively update the parameters (weights and biases) in the direction of the steepest descent of the loss function, effectively reducing the loss.

The update rule for a weight w is given by:

$$w := w - \alpha \frac{\partial J(w)}{\partial w}$$

where:

- α is the learning rate, which determines the size of the step to take in the direction of the gradient.
- $J(w)$ is the loss function with respect to the weight w.
- $\frac{\partial J(w)}{\partial w}$ is the gradient of the loss function with respect to w.

Similarly, biases are also updated using the same principle:

$$b := b - \alpha \frac{\partial J(b)}{\partial b}$$

The gradient tells us the direction of the steepest increase, so moving in the opposite direction decreases the loss.

2.6 Adaptive Learning Rates

Adam Optimizer

The **Adam** (Adaptive Moment Estimation) optimizer combines the advantages of two other methods: **AdaGrad** and **RMSProp**. It computes adaptive learning rates for each parameter by using both the first moment (mean) and the second moment (uncentered variance) of past gradients.

The update rules for the Adam optimizer are as follows:

1. Calculate the exponential moving averages of the gradient and its square:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

   where:

   - m_t is the first moment estimate (mean).
   - v_t is the second moment estimate (variance).
   - g_t is the gradient at time step t.
   - β_1 and β_2 are hyperparameters that control the decay rates of these moving averages.

2. Bias correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

3. Update the parameters:

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

   where:

   - α is the learning rate.
   - ϵ is a small constant to prevent division by zero.

3 Results

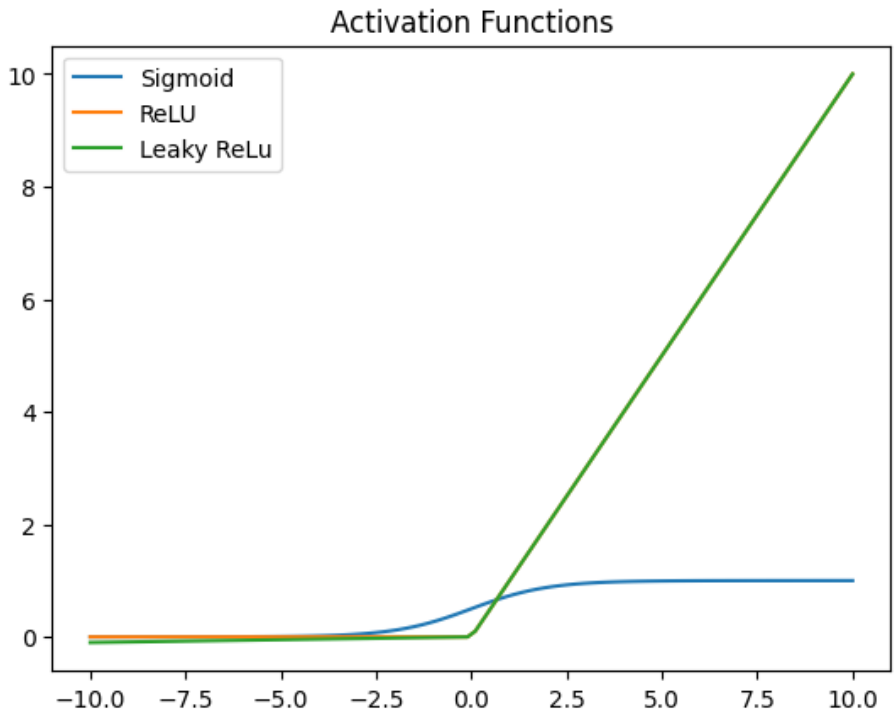

Figure 1: Comparison of Sigmoid, ReLU, and Leaky ReLU Activation Functions

The image above illustrates the behavior of three common activation functions used in neural networks: **Sigmoid**, **ReLU (Rectified Linear Unit)**, and **Leaky ReLU**. Each line in the plot represents the output of these activation functions for inputs ranging from -10 to 10.

The **sigmoid function** (depicted in blue) produces an S-shaped curve that gradually rises from 0 to 1. This function is often used in binary classification tasks because it maps input values to an output between 0 and 1, effectively serving as a probability estimator. The output asymptotically approaches 0 for large negative inputs and 1 for large positive inputs.

The **ReLU function** (depicted in orange) outputs 0 for all negative inputs and returns the input value itself for positive values. It provides a linear activation for positive inputs, which helps avoid the vanishing gradient problem that can occur in deep networks. However, the ReLU output remains at zero for negative values, which can result in inactive neurons (a problem sometimes called "dying ReLU").

The **Leaky ReLU function** (depicted in green) is a variation of ReLU that introduces a small slope for negative input values, thereby allowing a small, non-zero gradient when the input is negative. This approach helps prevent neurons from becoming inactive, which is an issue associated with traditional ReLU. The

plot shows that the green line has a small negative slope for negative inputs and follows the same behavior as ReLU for positive inputs.

The visual comparison helps to understand how each activation function behaves with different inputs, and why selecting an appropriate activation function is crucial for effective training of neural networks. ReLU and its variant, Leaky ReLU, are typically preferred for hidden layers due to their ability to mitigate the vanishing gradient problem, while the sigmoid function is more often used in output layers for binary classification.

4 Conclusion

Activation functions are essential in enabling neural networks to model complex relationships in data. The Sigmoid function is effective for binary classification, while ReLU and Leaky ReLU are often preferred for hidden layers due to their ability to mitigate issues like the vanishing gradient. By visualizing these activation functions, we gain insights into their individual characteristics and the advantages they offer for neural network training. This understanding is foundational for selecting appropriate activation functions based on the specific requirements of a model.