Mathematical Report on a Simple Neural Network

Your Name

2024-02-25

Introduction

Neural networks are a cornerstone of machine learning, enabling computers to learn from observational data. This report provides an overview of a simple neural network, including its architecture, mathematical foundations, and an example implemented in R.

Neural Network Architecture

A simple neural network consists of an input layer, one or more hidden layers, and an output layer. Each layer contains neurons (nodes) that are connected to neurons in the next layer. These connections have associated weights that are adjusted during training.

Architecture Diagram

Include a diagram of the neural network architecture here.

Mathematical Foundations

Activation Function

The activation function introduces non-linearity into the network, enabling it to learn complex patterns. Common activation functions include sigmoid, tanh, and ReLU.

Sigmoid Function

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

Forward Propagation

Forward propagation involves calculating the input to each neuron in the hidden layers and the output layer.

Mathematical Representation

Given an input vector $x^{(i)}$, the output of a neuron j in layer l, $a_j^{(l)}$, is calculated as:

$$a_j^{(l)} = \sigma\left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right)$$

where $w_{jk}^{(l)}$ is the weight from neuron k in layer $(l-1)$ to neuron j in layer l, $b_j^{(l)}$ is the bias term for neuron j in layer l, and σ is the activation function.

Backpropagation

Backpropagation adjusts the weights of the connections in the network to minimize the difference between the actual output and the predicted output.

Mathematical Representation

The goal of backpropagation is to compute the gradient of the loss function with respect to each weight in the network, which involves calculating the partial derivatives of the loss function L with respect to each weight $w_{jk}^{(l)}$.

1. **Error term for output layer ($\delta^{(L)}$):**

For the output layer L, the error term for each neuron j is given by:

$$\delta_j^{(L)} = \frac{\partial L}{\partial a_j^{(L)}} \cdot \sigma'(z_j^{(L)})$$

where L is the loss function, $a_j^{(L)}$ is the activation of neuron j in the output layer, and $\sigma'(z_j^{(L)})$ is the derivative of the activation function with respect to the neuron's input $z_j^{(L)}$.

2. **Error term for hidden layers ($\delta^{(l)}$):**

The error term for each neuron j in a hidden layer l is calculated based on the error terms of the layer above $(l+1)$:

$$\delta_j^{(l)} = \left(\sum_k w_{jk}^{(l+1)} \delta_k^{(l+1)}\right) \cdot \sigma'(z_j^{(l)})$$

3. **Gradient of the loss function with respect to weights:**

The partial derivative of the loss function with respect to a weight $w_{jk}^{(l)}$ is given by:

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}$$

Weights are updated using gradient descent, where η is the learning rate:

$$w_{jk}^{(l)} = w_{jk}^{(l)} - \eta \frac{\partial L}{\partial w_{jk}^{(l)}}$$

Enhanced Mathematical Details for Backpropagation

To provide a clearer understanding of the backpropagation algorithm, we first need to define a loss function and then show how this loss is related to the activations in the output layer. Finally, we'll detail the role of the derivative of the activation function in the calculation of gradients.

Loss Function

A commonly used loss function for classification problems is the cross-entropy loss. For a neural network that outputs probabilities, the cross-entropy loss for a single training example with true label y and predicted probabilities $\hat{y}$ is defined as:

$$L(\hat{y}, y) = -\sum_{j} y_j \log(\hat{y}_j)$$

where y_j is the true probability of class j (usually 0 or 1 in a one-hot encoded vector), and $\hat{y}_j$ is the predicted probability of class j.

Relation Between Loss and Output Layer Activations

Considering the output layer activations $a_j^{[L]}$ as $\hat{y}_j$, the loss can be directly related to these activations for a binary classification (simplifying the notation for clarity):

$$L(a^{[L]}, y) = -y \log(a^{[L]}) - (1 - y) \log(1 - a^{[L]})$$

This expression shows how the loss depends on the activations of the output layer. The goal of training is to adjust the weights and biases to minimize this loss.

Derivative of Loss with Respect to Activations

To understand the gradient $\frac{\partial L}{\partial a_j^{[L]}}$, consider the derivative of the binary cross-entropy loss with respect to the activation $a^{[L]}$:

$$\frac{\partial L}{\partial a_j^{[L]}} = -\frac{y_j}{a_j^{[L]}} + \frac{1 - y_j}{1 - a_j^{[L]}}$$

This gradient tells us how changes in the output layer activation $a_j^{[L]}$ affect the loss. It is the starting point for backpropagating errors through the network.

Role of Activation Function Derivative

The derivative of the activation function $\sigma'(z_j^{[L]})$ appears in the backpropagation formula because we need to understand how changes in the weighted input $z_j^{[L]}$ (before activation) affect the loss. This is captured by the chain rule in calculus, which allows us to decompose the derivative of the loss with respect to $z_j^{[L]}$ through the activation function:

$$\frac{\partial L}{\partial z_j^{[L]}} = \frac{\partial L}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \frac{\partial L}{\partial a_j^{[L]}} \cdot \sigma'(z_j^{[L]})$$

Here, $\sigma'(z_j^{[L]})$ is the derivative of the activation function with respect to its input, showing how changes in the input to the activation function affect its output.

Combining these components, the error term for the output layer becomes:

$$\delta_j^{[L]} = \frac{\partial L}{\partial a_j^{[L]}} \cdot \sigma'(z_j^{[L]})$$

This formulation highlights the chain of dependencies from the loss through the output layer activations and back to the inputs of the neurons in the output layer, forming the basis for updating weights to minimize the loss.

Conclusion

The detailed mathematical derivation provided above elucidates the foundational principles behind the backpropagation algorithm. By understanding the relationship between the loss function, the activations in the output layer, and the derivative of the activation function, we gain insight into how neural networks learn and adjust their parameters to improve prediction accuracy.

Derivation of the Backpropagation Algorithm

Backpropagation is a systematic way of computing the gradient of the loss function with respect to each weight in the network. This section provides a step-by-step derivation of the backpropagation algorithm.

Preliminaries

Consider a neural network with L layers, each layer l having $n^{[l]}$ neurons. Let $w_{jk}^{[l]}$ denote the weight from the k^{th} neuron in layer $l-1$ to the j^{th} neuron in layer l, and $b_j^{[l]}$ the bias of the j^{th} neuron in layer l. The activation of the j^{th} neuron in layer l is represented as $a_j^{[l]}$, and the input to this neuron before activation as $z_j^{[l]}$. The network uses a loss function L to measure the error between the predicted output and the actual output.

Forward Propagation

1. Input layer activations ($a^{[0]}$) are set to the feature values.
2. For each layer $l = 1, \ldots, L$, compute:
   - Input to neurons: $z_j^{[l]} = \sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}$
   - Activation: $a_j^{[l]} = \sigma(z_j^{[l]})$

Backpropagation

Step 1: Error in Output Layer

The first step in backpropagation is to compute the error in the output layer, $\delta^{[L]}$, which measures how far the actual network output is from the desired output.

$$\delta_j^{[L]} = \frac{\partial L}{\partial a_j^{[L]}} \cdot \sigma'(z_j^{[L]})$$

where σ' is the derivative of the activation function.

Step 2: Backpropagate the Error

For each layer $l = L-1, L-2, \ldots, 1$, compute the error term $\delta^{[l]}$:

$$\delta_j^{[l]} = \left(\sum_k w_{kj}^{[l+1]} \delta_k^{[l+1]} \right) \cdot \sigma'(z_j^{[l]})$$

This equation calculates the error for each neuron based on the error of the neurons in the next layer and the derivative of the activation function, effectively backpropagating the error through the network.

Step 3: Compute Gradient

The gradient of the loss function with respect to the weights and biases is calculated using the errors computed in the previous step.

- For the weights:
$$\frac{\partial L}{\partial w_{jk}^{[l]}} = a_k^{[l-1]} \delta_j^{[l]}$$
- For the biases:
$$\frac{\partial L}{\partial b_j^{[l]}} = \delta_j^{[l]}$$

Step 4: Update Parameters

Finally, the weights and biases are updated using gradient descent:

$$w_{jk}^{[l]} = w_{jk}^{[l]} - \eta \frac{\partial L}{\partial w_{jk}^{[l]}}$$

$$b_j^{[l]} = b_j^{[l]} - \eta \frac{\partial L}{\partial b_j^{[l]}}$$

where η is the learning rate.

This step-by-step derivation outlines how the backpropagation algorithm computes the gradient of the loss function with respect to each parameter in the network, enabling the neural network to learn from the data by iteratively updating its weights.

Example Implementation in R

This section includes a practical example of implementing a simple neural network in R.

Installing Necessary Packages

```r
# Install necessary packages
# install.packages("neuralnet")
```

```r
# Load and prepare data
data <- iris # Placeholder for actual data loading and preprocessing steps
```

```r
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.3.2
```

```r
# Define the neural network
nn <- neuralnet(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=data, hidden=c(

# View the neural network
plot(nn)
```

```r
predictions <- compute(nn, data[,-5])
# predictions
```