Basics of Neural Network

Promothesh Chatterjee*

Understanding Neural Networks

A neural network is a machine learning model inspired by the human brain. It consists of interconnected neurons (nodes) that process data and learn patterns. The network is trained to recognize patterns, make decisions, and predict outcomes by adjusting weights and biases based on the input data.

Structure of a Neural Network

1. **Neurons (Nodes)**:
    - **Neurons** are the basic units of a neural network, similar to the neurons in the human brain.
    - Each neuron takes input, processes it, and produces an output.
    - Neurons are organized into layers.
2. **Layers**:
    - **Input Layer**: Receives the raw input data.
    - **Hidden Layers**: Layers between the input and output layers where the network processes the data.
    - **Output Layer**: Produces the network's output.

Pictorial Reference

Below is a simplified pictorial representation of a neural network with an input layer, one hidden layer, and an output layer. Each connection between neurons has a weight, and each neuron has a bias term.

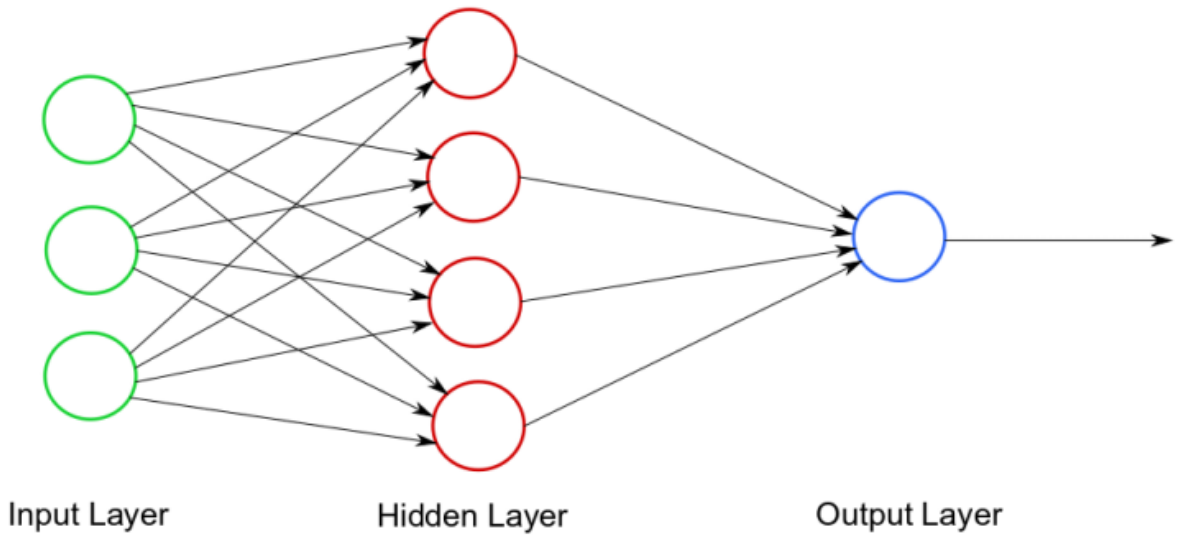


Figure 1: Neural Network Diagram

How It Works

1. **Inputs**:
    - Data is fed into the input layer. In NLP, the input could be word embeddings representing words in a sentence.
2. **Weights**:
    - Each connection between neurons has a weight that determines the importance of the input values.
    - Weights are adjusted during training to improve the network's performance.
3. **Bias**:
    - Each neuron has a bias term added to the weighted sum of inputs before passing through the activation function. This helps the model better fit the data.
4. **Linear Combination and Activation Functions**:
    - **Linear Combination**: Each neuron takes the inputs, multiplies them by their weights, adds the bias, and then sums them up. This is a linear combination. Mathematically, it's like calculating `weighted_sum = weight1 * input1 + weight2 * input2 + ... + weightN * inputN + bias`.
    - **Activation Function**: After the linear combination, the result is passed through an activation function. This function decides whether the neuron should be activated or not. It introduces non-linearity to the model, allowing it to learn complex patterns. Common examples are the sigmoid function, which squashes values between 0 and 1, and the ReLU (Rectified Linear Unit) function, which outputs the input directly if it is positive, else it outputs zero.
5. **Forward Propagation**:
    - This is the process of moving the inputs through the network, from the input layer through the hidden layers to the output layer, using the linear combinations and activation functions.
6. **Loss Function**:
    - After forward propagation, the network makes a prediction. The loss function measures how far this prediction is from the actual value. It's a way of quantifying how wrong the model's predictions are.
7. **Backpropagation and Gradient Descent**:
    - **Backpropagation**: This is the process of going back through the network to adjust the weights. It uses the derivative of the loss function to understand how each weight affects the final error.
    - **Gradient Descent**: This is an optimization algorithm used to minimize the loss by adjusting the weights. It calculates the gradient (or slope) of the loss function and changes the weights in the direction that reduces the loss.

Implementation: Sentiment Analysis with `quanteda` and `neuralnet`

We'll create a neural network to predict the sentiment of movie reviews using the `quanteda` package for text preprocessing.

1. Prepare the Dataset We will create a small dataset of movie reviews and their sentiments (0 for negative, 1 for positive).

```
set.seed(12345)

# Sample dataset
reviews <- c("I loved the movie", "It was terrible", "Great story and acting",
             "Worst film ever", "Fantastic!", "Not good", "Amazing movie",
             "Boring plot", "Superb!", "I did not like it")
sentiments <- c(1, 0, 1, 0, 1, 0, 1, 0, 1, 0)  # 1: Positive, 0: Negative

# Combine into a data frame
data <- data.frame(reviews, sentiments)
```

2. Text Preprocessing with `quanteda` We will preprocess the text reviews using `quanteda` and convert them into numerical representations using the TF-IDF method.

```
# Install and load necessary packages
# install.packages("quanteda")
# install.packages("quanteda.textmodels")
# install.packages("caret")
# install.packages("neuralnet")
library(quanteda)
library(quanteda.textmodels)
library(caret)
library(neuralnet)

# Create a corpus from the reviews
corpus <- corpus(data$reviews)

# Tokenize the corpus
tokens <- tokens(corpus, remove_punct = TRUE)

# Perform further preprocessing: convert to lower case, remove stopwords
tokens <- tokens_tolower(tokens)
tokens <- tokens_remove(tokens, stopwords("english"))

# Create a document-feature matrix (DFM) with TF-IDF weighting
dfm <- dfm(tokens)
dfm_tfidf <- dfm_tfidf(dfm)

# Convert the DFM to a matrix and then to a data frame
dfm_matrix <- as.matrix(dfm_tfidf)
```

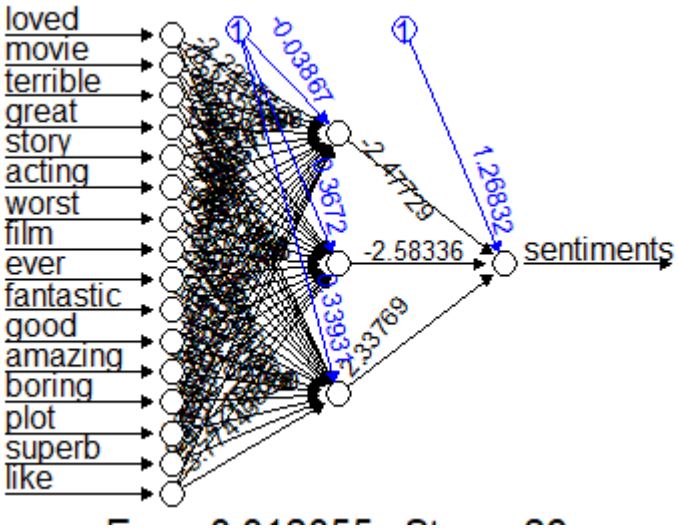
```r
dfm_df <- as.data.frame(dfm_matrix)

# Combine the DFM with the sentiments
data_preprocessed <- cbind(dfm_df, sentiments = data$sentiments)
```

3. Define and Train the Neural Network We will define a neural network with one hidden layer and train it using the preprocessed data.

```r
# Define the neural network
set.seed(12345)
net <- neuralnet(sentiments ~ ., data_preprocessed, hidden = 3, linear.output = FALSE)

# Plot the neural network
plot(net)
```



So, how does the hidden layer work, and can we have more hidden layers?

In our data set of movie reviews, we want to classify each review as either positive or negative.

Neural Network Structure:

1. **Input Layer**: Each review is represented as a vector, typically using techniques like Bag of Words, TF-IDF (as we have done it here), or word embeddings (e.g., Word2Vec, GloVe).
2. **Hidden Layers**: Let's say we have two hidden layers, each with 3 neurons.
3. **Output Layer**: A single neuron with a sigmoid activation function to output a probability that the review is positive.

Role of Hidden Layers:

1. **First Hidden Layer**:
   - **Feature Extraction**: This layer might learn to detect simple sentiment-related features in the text, such as the presence of certain keywords (e.g., "good", "bad", "excellent", "terrible").
   - **Non-linear Transformation**: Each neuron in this layer applies a non-linear activation function (e.g., ReLU, tanh) to the weighted sum of inputs, allowing the model to capture non-linear relationships between the words.
2. **Second Hidden Layer**:
   - **Higher-Order Features**: This layer can learn to combine the simple features detected by the first hidden layer into more complex patterns. For example, it might recognize combinations of words that indicate sentiment more reliably than individual words (e.g., "not good", "very bad").
   - **Abstraction**: This layer creates a more abstract representation of the input text, which is less sensitive to specific word choices and more focused on overall sentiment.

Example Workflow:

1. **Input Layer**:
   - Review: "The movie was not good at all."
   - Vector Representation (simplified): [0.1, 0.0, 0.2, 0.7, 0.0, 0.0, 0.3, …]
2. **First Hidden Layer**:
   - Neuron 1 might respond strongly to the word "good".
   - Neuron 2 might respond to the word "bad".
   - Neuron 3 might respond to the phrase "excellent".
   - Outputs (after activation): [0.8, 0.3, 0.9]
3. **Second Hidden Layer**:
   - Neuron 1 might detect patterns like negations followed by positive words.
   - Neuron 2 might focus on sequences of words that indicate strong sentiment.
   - Neuron 3 might combine these patterns into an overall sentiment score.
   - Outputs (after activation): [0.6, 0.7, 0.4]
4. **Output Layer**:
   - Sigmoid activation function is applied to the weighted sum of the second hidden layer's outputs to produce a probability.
   - Output: 0.2 (indicating a low probability of a positive review, hence classified as negative).

Code Example in R:

Here's how you might set up this neural network using the `neuralnet` package in R:

```r
# Define the neural network with two hidden layers, each with 3 neurons
set.seed(12345)
net1 <- neuralnet(sentiments ~ ., data_preprocessed, hidden = c(3, 3), linear.output = FALSE)

# Plot the neural network
#plot(net1)
```

In this example, the hidden layers transform the input text features through a series of non-linear transformations, enabling the neural network to learn and recognize complex patterns in the text that correlate with the sentiment of the reviews. I haven't plotted the output of 'net1' as I want to keep the example simple but it is done pretty much the same way.

4. Predict Sentiment for New Reviews We will preprocess new reviews using `quanteda` and use the trained neural network `net` with single hidden layer to predict their sentiments.

```r
# Sample new reviews for prediction
new_reviews <- c("I enjoyed the movie", "It was not good", "The movie was superb")

# Preprocess the new reviews
new_corpus <- corpus(new_reviews)
new_tokens <- tokens(new_corpus, remove_punct = TRUE)
new_tokens <- tokens_tolower(new_tokens)
new_tokens <- tokens_remove(new_tokens, stopwords("english"))
new_dfm <- dfm(new_tokens)

# Match the features of the new DFM to the original DFM
new_dfm <- dfm_match(new_dfm, features = featnames(dfm))

# Apply TF-IDF weighting
new_dfm_tfidf <- dfm_tfidf(new_dfm)

# Convert the new DFM to a matrix and then to a data frame
new_dfm_matrix <- as.matrix(new_dfm_tfidf)
new_dfm_df <- as.data.frame(new_dfm_matrix)

# Predict using the neural network
predictions <- compute(net, new_dfm_df)

# View predictions
predictions$net.result
```

```
##             [,1]
## text1 0.76700166
```

```
## text2 0.04372611
## text3 0.92958267
```

```
# Apply a threshold to classify the sentiments
threshold <- 0.5
predicted_classes <- ifelse(predictions$net.result > threshold, "Positive", "Negative")
predicted_classes
```

```
##       [,1]
## text1 "Positive"
## text2 "Negative"
## text3 "Positive"
```

Summary

1. **Prepare the Dataset**: We created a small dataset of movie reviews and their sentiments.
2. **Text Preprocessing with `quanteda`**: Converted the text data into numerical representations using TF-IDF.
3. **Define and Train the Neural Network**: Trained the neural network on the numerical representations of the reviews.
4. **Predict Sentiment for New Reviews**: Preprocessed new reviews, matched features with the original DFM, and used the trained neural network to predict sentiments.

Detailed Explanation

Inputs and Weights

- **Inputs**: These are the pieces of data you feed into the network. In our case, the input is the TF-IDF weighted terms from the reviews.
- **Weights**: Weights are the importance given to each input. The network learns the optimal weights during training.

Linear Combination and Activation Functions

- **Linear Combination**: Each neuron takes the inputs, multiplies them by their weights, adds the bias, and then sums them up.
- **Activation Function**: The result of the linear combination is passed through an activation function, introducing non-linearity. Common activation functions include Sigmoid and ReLU.

Forward Propagation

- Forward propagation involves moving the inputs through the network, from the input layer through the hidden layers to the output layer.

Loss Function

- The loss function measures how far the network's predictions are from the actual values.

Backpropagation and Gradient Descent

- **Backpropagation**: Adjusts the weights based on the error calculated by the loss function.
- **Gradient Descent**: Minimizes the loss by adjusting the weights in the direction that reduces the error.