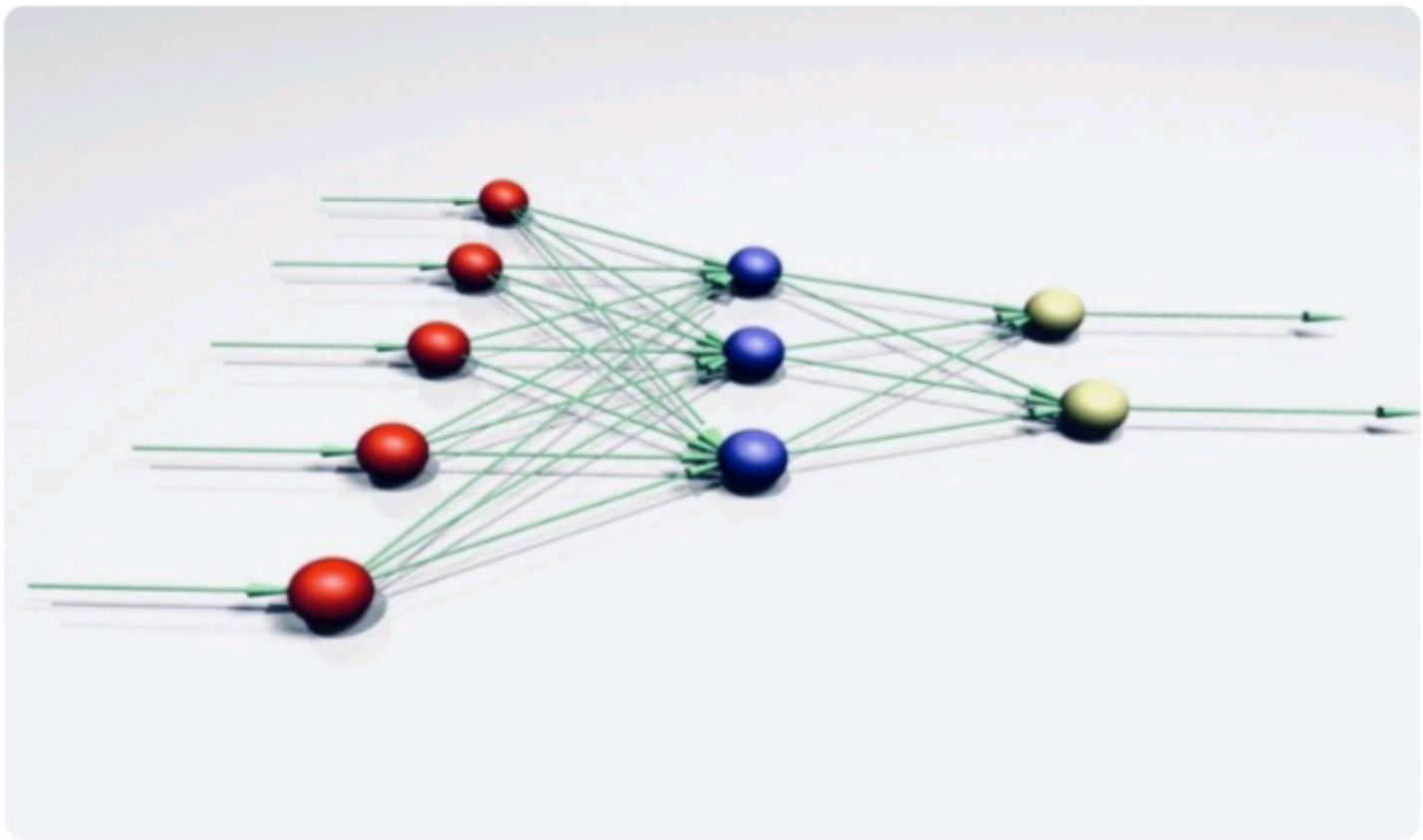

Keras is one of the most popular deep learning libraries of the day and has made a big contribution to the commoditization of artificial intelligence. It is simple to use and can build powerful neural networks in just a few lines of code.

In this post, we'll walk through how to build a neural network with Keras that predicts the sentiment of user reviews by categorizing them into two categories: positive or negative. This is called sentiment analysis and we will do it with the famous IMDB review dataset. The model we'll build can also be applied to other machine learning problems with just a few changes.

Note that we will not go into the details of Keras or deep learning. This post is intended to provide a blueprint of a Keras neural network and to make you familiar with its implementation.

TABLE OF CONTENTS

What is Keras?

Keras is an open source Python library for easily building neural networks. The library is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano and MXNet. Tensorflow and Theano are the most used numerical platforms in Python when building deep learning algorithms, but they can be quite complex and difficult to use.
By comparison, Keras provides an easy and convenient way to build deep learning models.

What is Sentiment Analysis?

Sentiment analysis aims to determine the attitude, or sentiment. For example, a speaker or writer with respect to a document, interaction, or event. It is a natural language processing problem in which text needs to be understood to predict the underlying intent.

The sentiment is mostly categorized into positive, negative and neutral categories. Through sentiment analysis we might want to predict, for example, a customer's opinion and attitude about a product based on a review they wrote. This technique is widely applied to things like reviews, surveys, documents and much more.

The imdb Dataset

The IMDB sentiment classification dataset consists of 50,000 movie reviews from IMDB users that are labeled as either positive (1) or negative (0). The reviews are preprocessed and each one is encoded as a sequence of word indexes in the form of integers. The words within the reviews are indexed by their overall frequency within the dataset. For example, the integer “2” encodes the second most frequent word in the data. The 50,000 reviews are split into 25,000 for training and 25,000 for testing.

Importing Dependencies and getting the Data

We start by importing the required dependencies to preprocess our data and build our model.

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt


import numpy as np
from keras.utils import to_categorical
from keras import models
from keras import layers
```

Continue downloading the IMDB dataset, which is, fortunately, already built into Keras. Since we want to avoid a 50/50 train test split, we will immediately merge the data into data and targets after downloading so we can do an 80/20 split later on.

```
from keras.datasets import imdb
(training_data, training_targets), (test
data = np.concatenate((training_data, te
targets = np.concatenate((training_targe
```

Exploring the Data[10]

Now we can start exploring the dataset:

```
print("Categories:", np.unique(targets))
print("Number of unique words:", len(np.u

Categories: [0 1]
Number of unique words: 9998
length = [len(i) for i in data]
print("Average Review length:", np.mean(
print("Standard Deviation:", round(np.st

Average Review length: 234.75892
Standard Deviation: 173.0
```

Let's look at a single training example:

```
print("Label:", targets[0])


Label: 1
print(data[0])


[1, 14, 22, 16, 43, 530, 973, 1622, 1385
```

Above you can see the first review of the dataset, which is labeled as positive (1). The code below retrieves the dictionary mapping word indices back into the original words so that we can read them. It replaces every unknown word with a “#”. It does this by using
the get_word_index() function.

```
index = imdb.get_word_index()
reverse_index = dict([(value, key) for (
decoded = " ".join( [reverse_index.get(i
print(decoded)

# this film was just brilliant casting l
```

Data Preparation

Now it's time to prepare our data. We will vectorize every review and fill it with zeros so it contains exactly 10,000 numbers. That means we fill every review that is shorter than 10,000 with zeros. We need to do this because the biggest review is nearly that long and every input for our neural network needs to have the same size. We will also transform the targets into floats.

```
def vectorize(sequences, dimension = 100
results = np.zeros((len(sequences), dime
for i, sequence in enumerate(sequences):
results[i, sequence] = 1
return results
```

```
def vectorize(sequences, dimension = 100
results = np.zeros((len(sequences), dime
for i, sequence in enumerate(sequences):
results[i, sequence] = 1
return results

data = vectorize(data)
targets = np.array(targets).astype("floa
```

Now we split our data into a training and a testing set. The training set will contain 40,000 reviews and the testing set 10,000.

```
test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]
```

Building and Training the Model

Now we're ready to build our simple neural network. We'll start by defining the type of model we want to build. There are two types of models available in Keras: the sequential model and the model class used with functional API.

Next we simply add the input-, hidden- and output-layers. Between them, we are using dropout to prevent overfitting. Please note you should always use a dropout rate between 20% and 50%.

```
# Input - Layer
model.add(layers.Dense(50, activation =
# Hidden - Layers
model.add(layers.Dropout(0.3, noise_shap
model.add(layers.Dense(50, activation =
model.add(layers.Dropout(0.2, noise_shap
model.add(layers.Dense(50, activation =
# Output- Layer
model.add(layers.Dense(1, activation = "
model.summary()
```

```
_________________________________________
Layer (type)                 Output Shap
=========================================
dense_1 (Dense)              (None, 50)
_________________________________________
dropout_1 (Dropout)          (None, 50)
_________________________________________
dense_2 (Dense)              (None, 50)
_________________________________________
```

```
_________________________________________________________________
Layer (type)                 Output Shap
=================================================================
dense_1 (Dense)              (None, 50)
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)
_________________________________________________________________
dense_2 (Dense)              (None, 50)
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)
_________________________________________________________________
dense_3 (Dense)              (None, 50)
_________________________________________________________________
dense_4 (Dense)              (None, 1)
=================================================================
Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
_________________________________________________________________
```