JANUARY 2018

UNIVERSIDAD DE GRANADA

STOCK MARKET PREDICTION

IBEX-35 TensorFlow

PREPARED BY:

Juan Manuel López Torralba

INTELIGENCIA ARTIFICAL EN TELECOMUNICACIONES

Instructor: García Castellano, Francisco Javier

1. Objetivos

El siguiente proyecto consiste en la realización de un sistema basado en *Deep Learning* que realice predicciones sobre el IBEX-35. Para ello se desarrollará un programa en Python 3.6 y TensorFlow.

2. Introducción

2.1. *Machine Learning*

Se define como el conjunto de técnicas de inteligencia artificial donde los ordenadores son capaces de aprender de forma autónoma sin intervención humana directa. Para ello la máquina debe ser entrenada de diferentes maneras según el propósito del sistema. El programa debe aprender combinaciones de características de este set de entrenamiento para así construir un modelo.

A continuación se debe probar la eficacia del sistema mediante un conjunto de test, conjunto que a su vez servirá para mejorar dicho modelo mediante la aplicación de variables de ajuste, pesos, que se van recalculando durante todo el proceso. El entrenamiento supervisado es rápido y n hace uso de muchos. Sin embargo necesita de intervención humana para etiquetar la información del entrenamiento, lo que en general es lento y costoso [1].

2.2. *Deep Learning*

Se define como un conjunto de algoritmos que pretenden el modelaje de datos a alto nivel, para lo cual se apoyan en el uso de arquitecturas compuestas de muchas capas en las que se realizan operaciones no lineales. La metodología más común es la red neuronal artificial, siendo la que mejor resultados obtiene en visión por ordenador o reconocimiento de imágenes por ejemplo. En *Deep Learning* se obtienen tasas de éxito elevadas con entrenamiento no supervisado.

2.3. TensorFlow

TensorFlow es una biblioteca de software de código abierto empelada para el cálculo numérico utilizando gráficos de flujo de datos. Los nodos en el gráfico representan operaciones matemáticas, mientras que los bordes del gráfico representan los conjuntos de datos multidimensionales (tensores) comunicados entre ellos. Esta arquitectura flexible le permite implementar cálculos en una o más CPU o GPU en cualquier ordenador de escritorio, servidor o dispositivo móvil con una sola API. TensorFlow fue desarrollado originalmente por investigadores e ingenieros que trabajan en el equipo Brain de Google dentro de la organización de

investigación de Inteligencia Artificial de Google para realizar investigaciones de aprendizaje automático y redes neuronales profundas [2].

3. Desarrollo del sistema

En primer lugar debemos encontrar los datos de mercado del Ibex-35, para lo cual se hará uso de la herramienta *Yahoo Finances*, la cual guarda un registro de los valores en bolsa del Ibex desde su salida a bolsa allá por el año 2000. Se descargan los valores diarios de cierre, apertura y máximo valor del día, entre otros. Para la realización del programa se cuenta tanto con datos del IBEX-35 de forma global como de empresas asociadas individuales como MAPRE, BBVA y ENDESA, que serán usados de forma independiente, dando así la posibilidad de también predecir el porvenir de estas empresas de forma individual.

A continuación, se procede a la descripción y explicación del código desarrollado:

```
# Import modules
import tensorflow as tf                          # Tensorflow
import numpy as np                               # Numpy arrays
import os
import math
import pandas as pd                              # Read data from CSV file
from datetime import datetime, timedelta
import matplotlib.pyplot as plt                  # plot
from sklearn.preprocessing import MinMaxScaler   # Estimator
```

En el código anterior se importan los módulos necesarios para la realización del sistema. *Tensorflow* es la biblioteca de Google utilizada en este programa para construir y entrenar redes neuronales. *NumPy* es una biblioteca de *Python* que contiene funciones matemáticas de alto nivel para operar con vectores y matrices. *Os* es un módulo de *Python* que te facilita la interacción con el sistema operativo. *Math* es otro modulo para operaciones matemáticas. La librería de *Pandas* te permite trabajar con estructuras datos como archivos .csv, que es el formato elegido como contenedor de los datos originales de entrada del programa. Con *datetime* se gestionarán las marcas temporales de los datos. La librería *matplotlib* es para representar gráficos y MinMaxScaler se usa para escalar las entradas.

El programa continúa con la definición de funciones que ayudan tanto a la limpieza como a la comprensión del código. La primera de ella es *clean_dataset(df)* que se encargará de eliminar variables de valor infinito y NaN, además de vacíos [3]. La siguiente función es *weight_variable(shape)* con la que construiremos la matriz de pesos de las distintas capas de la red neuronal. Con la función *bias_variable(shape)*

se introduce un pequeño ruido. Finalmente, se define la función de activación de la red neuronal *rectified linear unit reLU(x, weight, bias)*. Tal y como se puede apreciar a continuación:

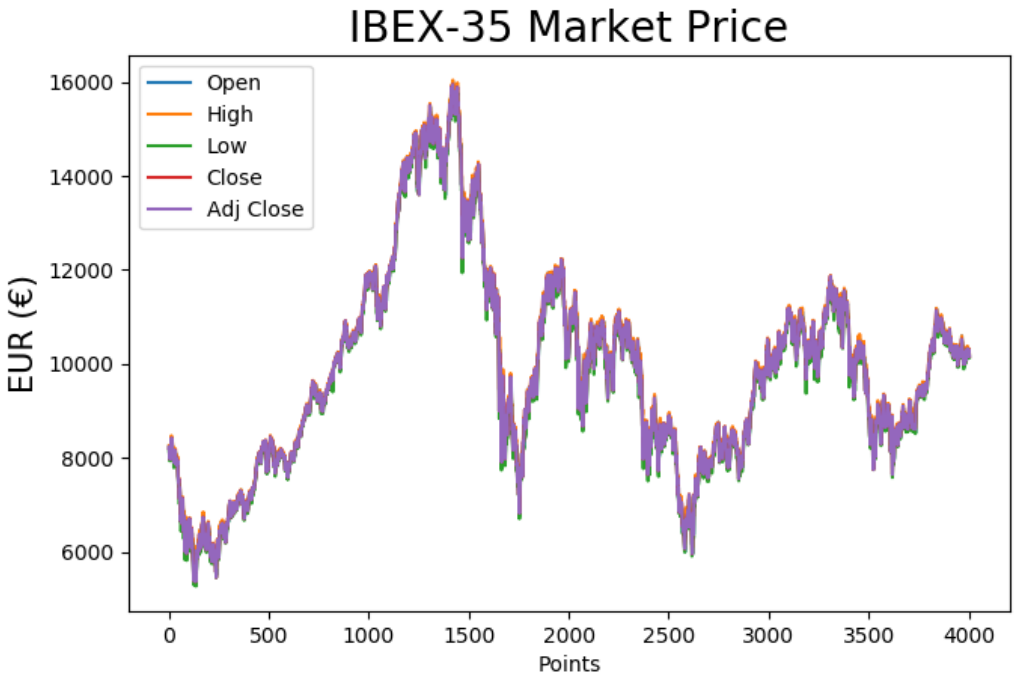
```
#Function def

# Remove NaN,inf values

def clean_dataset(df):

        assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"

        df.dropna(inplace=True)

        indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)

        return df[indices_to_keep].astype(np.float64)


# weight def

def weight_variable(shape):

        weight_initializer=tf.variance_scaling_initializer(mode="fan_avg",distribution="uniform"
, scale=1)

        initial = weight_initializer(shape)

        return tf.Variable(initial)


# bias def

def bias_variable(shape):

        bias_initializer = tf.zeros_initializer()

        initial = bias_initializer(shape)

        return tf.Variable(initial)# reLU def

def reLU(x, weight, bias):

 aux = tf.add(tf.matmul(x, weight), bias)

 return tf.nn.relu(aux)
```

Posteriormente se procede al pre-procesado de los datos. En primer lugar se carga el fichero ".csv" que se desee. Como se dijo anteriormente, se disponen de los valores en bolsa de ENDESA, BBVA, MAPRE e IBEX-35 desde su inicio en la bolsa de Madrid. Seguidamente, se hace una conversión de *string* a valor numérico de todas las fechas contenidas en el fichero de datos, que hacen referencia al día concreto donde la cotización en bolsa del sujeto bajo estudio alcanzó un valor específico. Esto se hace

para lograr un manejo más sencillo de ese campo de datos por si hiciera falta su tratamiento en futuras especificaciones. El código continúa con la eliminación, por columnas, de los campos indeseados, como pudieran ser las fechas o el volumen contratos/contratos que son comerciados en un día concreto. Finalmente, se chequea que no se disponen de valores numéricos incorrectos como -∞, +∞ o *NaN*. A continuación se muestra la sección del código a la que re hace referencia:

```
# Csv data
dataRaw = pd.read_csv('..\csvfiles/ibex35/ibex35.csv')
#dataRaw = pd.read_csv('..\csvfiles/ibex35/endesa.csv')
#dataRaw = pd.read_csv('..\csvfiles/ibex35/bbva.csv')
#dataRaw = pd.read_csv('..\csvfiles/ibex35/mapfre.csv')


# Parse data


dateRawAux = dataRaw['Date'].values
dateTimeAux = np.zeros(dateRawAux.shape)


        #date strings to numeric value
for i, j in enumerate(dateRawAux):


        dateTimeAux[i] = datetime.strptime(j, '%Y-%m-%d').timestamp()
   #Add the newly parsed column to the dataframe
        dataRaw['Timestamp'] = dateTimeAux
        Timestamp = dateTimeAux


# Remove undesired columns by columns
dataFrame_redux = dataRaw.drop(['Date','Timestamp','Volume'],axis=1)
data = clean_dataset(dataFrame_redux)
data = dataFrame_redux.values            # te lo da como numpy


print(np.any(np.isnan(data)))
```

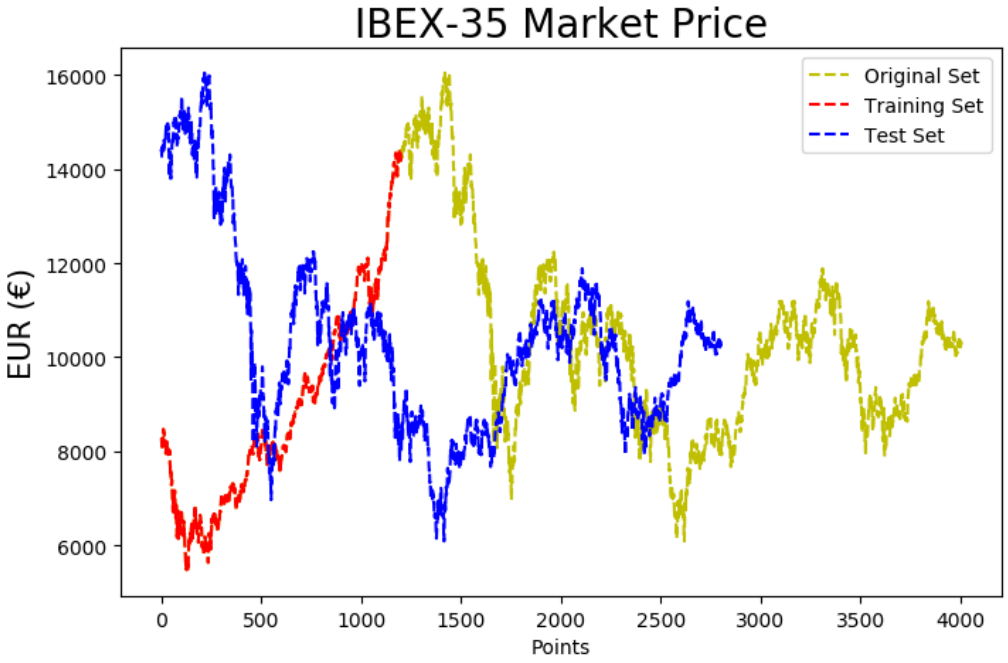
A continuación se muestran las cotizaciones históricas del IBEX-35 en Euros desde el año 2000:



El siguiente paso es el de seleccionar el set de datos de entrenamiento y test:

```
#Set the training data

train_start = 0

train_end = int(np.floor(0.3*n))

test_start = train_end + 1

test_end = n

data_train = data[np.arange(train_start, train_end), :]

data_test = data[np.arange(test_start, test_end), :]
```

Como podemos observar, se ha seleccionado como set de entrenamiento al 30 % de la cantidad total de datos originales, quedando el restante 70 % para test. En la siguiente figura puede observarse:



Posteriormente se escalan los datos de entrada con el fin de poder aplicar la función de activación reLU, que se define en el rango [-1, 1]. Esto se hace de forma sencilla en Python con la librería de sklearn MinMaxScaler, tal y como se muestra en el siguiente bloque de código:

```
scaler = MinMaxScaler(feature_range=(-1, 1))

scaler.fit(data_train)

data_train = scaler.transform(data_train)

data_test = scaler.transform(data_test)
```

Finalmente, antes de proceder con el diseño de la red neuronal, se definen las señales de entrada y salida de los set de entrenamiento y test:

```
# Build x and y test & training set

X_train = data_train[:, 1:]

y_train = data_train[:, 0]

X_test = data_test[:, 1:]

y_test = data_test[:, 0]

# Number of stocks in training data

n_stocks = X_train.shape[1]
```

Para la construcción de la red neuronal se hace uso de las librerías de TensorFlow para Python. En primer lugar se debe iniciar una sesión de *TensorFlow*, en mi caso la hago interactiva para ahorrarnos la necesidad de hacer constante referencia al objeto *session* [4]. Se definen los *placeholders* que son unas variables "simbólicas", que pueden ser manipuladas durante la ejecución del programa [5]. Se necesitan dos placeholders para ajustar nuestro modelo. "x" contiene las entradas; esto es, las cotizaciones en bolsa del IBEX-35, en una matriz bidimensional. Por otro lado "y" contiene la red de salida en una matriz mono dimensional. Posteriormente se define las capas ocultas (4) que forman la red y el número de neuronas de cada una. Esto puede verse en la siguiente sección de código:

```
## Building the Artifitial Neural Network (ANN)


# Neurons

n_neurons_1 = 1024

n_neurons_2 = int(n_neurons_1/2)  #512

n_neurons_3 = int(n_neurons_2/2)  #256

n_neurons_4 = int(n_neurons_3/2)  #128


# Session

ann = tf.InteractiveSession()


# Placeholder

x = tf.placeholder(dtype=tf.float32, shape=[None, n_stocks])

y = tf.placeholder(dtype=tf.float32, shape=[None])
```

El siguiente paso es definir las distintas capas ocultas de la red neuronal y las relaciones entre las mismas, tal que así:

$$h = Weight \cdot x + bias$$

Posteriormente, se aplica la función de activación reLU a las capas de la red y se obtiene la capa de salida:

```
# Hidden layers weights

W_hidden_1 = weight_variable([n_stocks, n_neurons_1])

W_hidden_2 = weight_variable([n_neurons_1, n_neurons_2])

W_hidden_3 = weight_variable([n_neurons_2, n_neurons_3])

W_hidden_4 = weight_variable([n_neurons_3, n_neurons_4])


# Hidden layers biases

bias_hidden_1 = bias_variable([n_neurons_1])

bias_hidden_2 = bias_variable([n_neurons_2])

bias_hidden_3 = bias_variable([n_neurons_3])

bias_hidden_4 = bias_variable([n_neurons_4])


# Output layer weights & biases

W_out = weight_variable([n_neurons_4, 1])

bias_out = bias_variable([1])


# Hidden layer activation function application

hidden_1 = reLU(x, W_hidden_1, bias_hidden_1)

hidden_2 = reLU(hidden_1, W_hidden_2, bias_hidden_2)

hidden_3 = reLU(hidden_2, W_hidden_3, bias_hidden_3)

hidden_4 = reLU(hidden_3, W_hidden_4, bias_hidden_4)

# Output layer (transpose!)

out = tf.transpose(tf.add(tf.matmul(hidden_4, W_out), bias_out))
```

Finalmente, nos queda definir la función de coste y la función de optimización. La función de coste genera medidas de la desviación entre las predicciones de la red y las observaciones de los sets de entrenamiento. Se usará el *mean squared error (MSE)*. El optimizador se ocupa de los cálculos necesarios que se utilizan para adaptar el peso de la red y las variables de sesgo durante el entrenamiento. Estos cálculos invocan el cálculo de los llamados gradientes, que indican la dirección en la que se deben cambiar los pesos y sesgos durante el entrenamiento para así minimizar la función de coste [6]. Una de las funciones de optimización más comunes en *Deep Learning* es *Adam* que significa "*Adaptive Moment Estimation*" y es una combinación de AdaGrad and RMSProp.

```
# Cost function
mse = tf.reduce_mean(tf.squared_difference(out, y))
# Optimizer
opt = tf.train.AdamOptimizer(1e-3).minimize(mse)
# Init
ann.run(tf.global_variables_initializer())
```

Para acabar ya solo queda alimentar la red neuronal con lotes de datos del set de entrenamiento y hacer un test final con el set de pruebas. Para ello se realizan varios bucles *for* y una serie de *plots* con el transcurso del test:

```
# Run
epochs = 20
for e in range(epochs):

    # Shuffle training data
    shuffle_indices = np.random.permutation(np.arange(len(y_train)))
    X_train = X_train[shuffle_indices]
    y_train = y_train[shuffle_indices]

    # Minibatch training
    for i in range(0, len(y_train) // batch_size):
        start = i * batch_size
        batch_x = X_train[start:start + batch_size]
        batch_y = y_train[start:start + batch_size]
        # Run optimizer with batch
        ann.run(opt, feed_dict={x: batch_x, y: batch_y})

        # Show progress
        if np.mod(i, 10) == 0:
            # MSE train and test
            mse_train.append(ann.run(mse, feed_dict={x: X_train, y: y_train}))
```

```
mse_test.append(ann.run(mse, feed_dict={x: X_test, y: y_test}))


mse_train_aux = mse_train[-1]

mse_test_aux = mse_test[-1]


print('MSE Train: ', mse_train_aux)

print('MSE Test: ', mse_test_aux)


# Prediction

pred = ann.run(out, feed_dict={x: X_test})

line2.set_ydata(pred)

plt.title('Test Performance: '+'Epoch ' + str(e) + ', Batch ' + str(i))

plt.xlabel('points')

textvar = plt.text(x = 1000, y = 1.0, s = 'MSE Train: ' + str(mse_train_aux), fontsize = 8)

textvar2 = plt.text(x = 1000, y = 0.9, s = 'MSE Test: '  + str(mse_test_aux), fontsize = 8)

plt.legend( ('Original Set', 'Prediction'), loc = 'upper right')

plt.pause(0.01)

textvar.remove()

textvar2.remove()
```

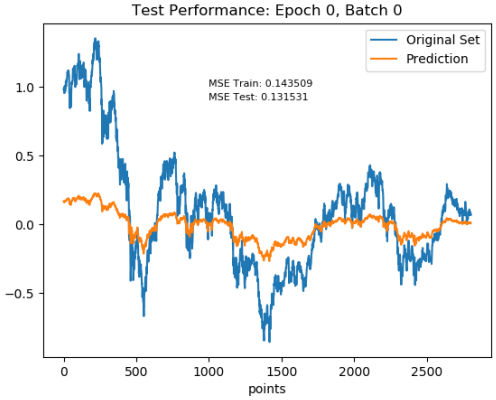
4. Resultados

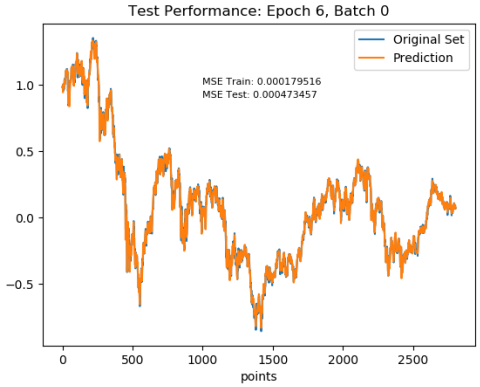
Tras la realización del programa se observa que la red se adapta muy bien al problema descrito, en concreto se obtiene, en términos de MSE, lo siguiente:

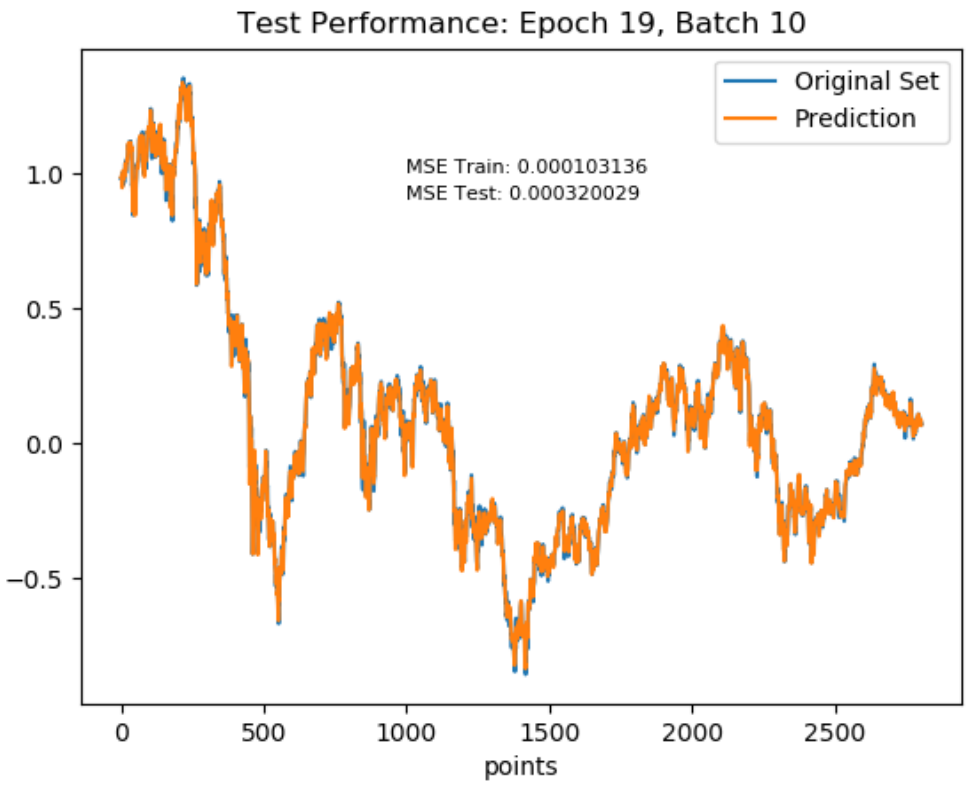
$$MSE\ Train\colon\ 0.000112042$$

$$MSE\ Test\colon\ 0.00032189$$

En las siguientes figuras puede observarse el transcurso de la prueba:







5. Conclusiones

Se ha conseguido realizar una red neuronal con *Python* y *TensorFlow* que cosigue hacer una buena predicción de la cotización del IBEX-35.

La realización de este proyecto ha servido como una primera, y muy útil, toma de contacto con las librerías de *TensorFlow* para crear redes neuronales y observar sus resultados, problemas y soluciones.

En el futuro se podía mejorar el diseño de este sistema mediante la implementación de predicciones futuras de la cotización en bolsa basadas quizá en movimientos de mercado de compañías rivales, noticias de compañías de relevancia como *Bloomberg*, o incluso comentarios en Twitter.

6. Referencias

[1] C. G. Moreno, «IndraCompany,» [En línea]. Available: https://www.indracompany.com/es/blogneo/deep-learning-sirve.

[2] Google, "About TensorFlow," [Online]. Available: https://www.tensorflow.org/.

[3] N. T. Smith, "Stock Market Prediction Using Multi-Layer Perceptrons With TensorFlow," 20 April 2016. [Online]. Available: https://nicholastsmith.wordpress.com/2016/04/20/stock-market-prediction-using-multi-layer-perceptrons-with-tensorflow/. [Accessed 26 01 2018].

[4] L. TensorFlow, "Interative Sessions," [Online]. Available: https://learningtensorflow.com/lesson5/. [Accessed 26 01 2018].

[5] J. Torres, «“Hello World” en TensorFlow,» 04 05 2017. [En línea]. Available: http://jorditorres.org/research-teaching/tensorflow/libro-hello-world-en-tensorflow/. [Último acceso: 26 01 2018].

[6] S. Heinz, "Medium," 9 2017. [Online]. Available: https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877?token=oLoZbeiwJj95kZbW. [Accessed 26 01 2018].