GUIÓN DE PRÁCTICAS DE REDES NEURONALES ARTIFICIALES (TSCAO): Parte IV

MªCarmen Pegalajar Jiménez

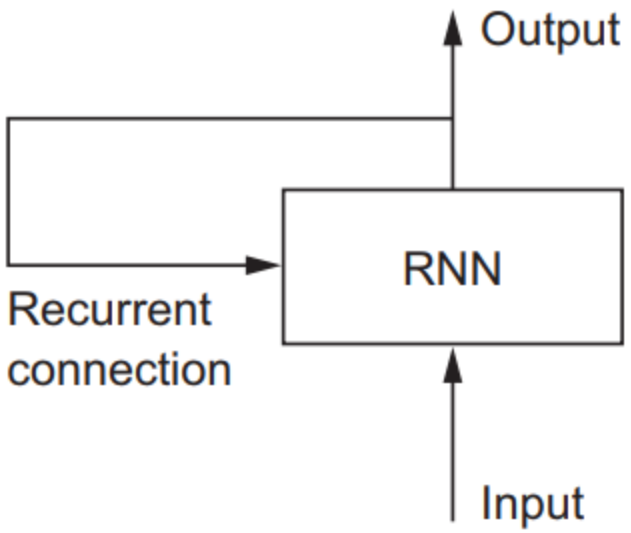
29/01/2024

Dpto Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

4. Redes Neuronales Recurrentes.

Es útil comprender al menos algunos de los conceptos básicos antes de comenzar con la implementación. Una red neuronal recurrente (RNN) procesa secuencias, ya sea precios diarios de acciones, frases o mediciones de sensores, se procesará por ejemplo un elemento por unidad de tiempo (cada vez) y la RNN mientras tanto retendrá en una memoria (llamada estado) lo que ha sucedido previamente en la secuencia.

Recurrente significa que la salida en el paso de tiempo actual se convierte en la entrada al siguiente paso de tiempo. En cada elemento de la secuencia, el modelo considera no solo la entrada actual, sino lo que recuerda acerca de los elementos anteriores.



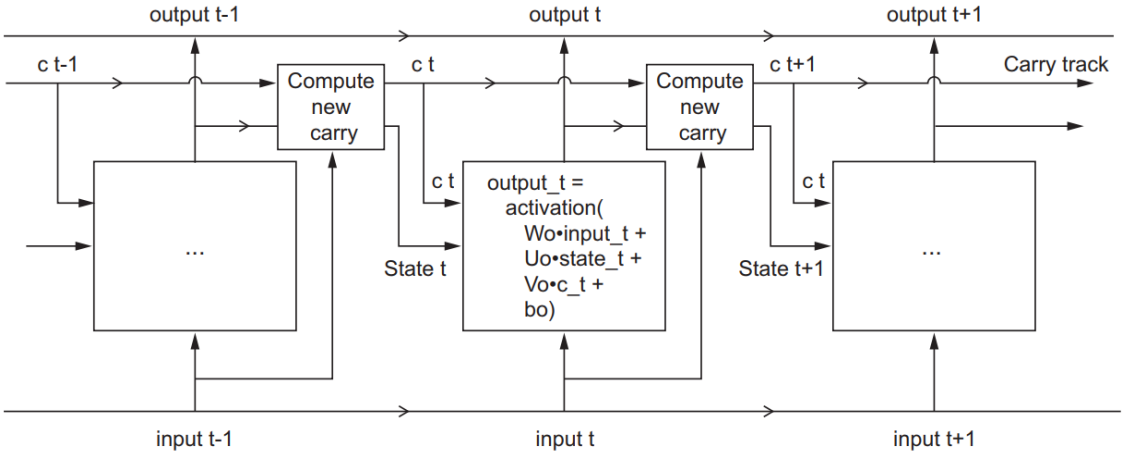
Esta memoria permite a la red aprender dependencias a largo plazo en una secuencia, lo que significa que puede tener en cuenta todo el contexto al hacer una predicción, ya sea la siguiente palabra en una frase, una clasificación de sentimientos o la próxima medición de temperatura. Una RNN está diseñada para imitar la forma humana de procesar secuencias: consideramos la oración completa al formar una respuesta en lugar de palabras por sí mismas. Por ejemplo, considera la siguiente frase:

"El concierto fue aburrido durante los primeros 15 minutos mientras la banda entraba en calor…,

pero luego fue terriblemente emocionante".

Un modelo de aprendizaje automático que considera las palabras de forma aislada, como un modelo de “bolsa” de palabras, probablemente concluiría que esta oración es negativa. Una RNN, por el contrario, debería poder ver las palabras "pero" y "terriblemente emocionante" y darse cuenta de que la oración cambia de negativa a positiva porque ha examinado toda la secuencia. Leer una secuencia completa nos da un contexto para procesar su significado, un concepto codificado en redes neuronales recurrentes.

Hoy por hoy las redes más utilizadas y populares son las redes LSTM (Long Short-Term Memory). En el corazón de una red recurrente LSTM hay una capa de celdas de memoria. Estas RNA mantienen un estado de sus celdas tal que la transmisión de la señal entre las neuronas (información en forma de gradiente) no se pierda a medida que se procesa la secuencia. En cada paso de tiempo, la LSTM considera la entrada actual, el “acarreo” y el estado de la celda.



Las celdas de memoria (unidades neuronales) asociadas a la red LSTM tienen 3 puertas y vectores de peso diferentes: hay una puerta para "olvidar" y descartar información irrelevante; una puerta de "entrada" para manejar la entrada actual, y una puerta de "salida" para producir predicciones en cada paso de tiempo. Sin embargo, como señala Chollet, es infructuoso tratar de asignar significados específicos a cada uno de los elementos de la celda.

La función de cada elemento celular de la LSTM se decide en última instancia por los parámetros (pesos) que se aprenden durante el entrenamiento. Debemos de recordar que el beneficio de una red neuronal recurrente para el aprendizaje de secuencias es que mantiene una memoria de toda la secuencia evitando que se pierda información previa.

A continuación, veremos un ejemplo de predicción de bolsa en las acciones de Google.

Lo primero que haremos será cargar el fichero de datos. Usamos pandas para importar el conjunto de datos y la función *iloc* para crear una matriz que tenga las dimensiones especificadas [:, 1:2] (haremos la predicción sobre la variable Open)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importamos el conjunto de entrenamiento

training_set = pd.read_csv('Google_Stock_Price_Train.csv', sep=',')

# Trabajaremos con la columna "Open"

training_set = training_set.iloc[:,1:2].values
```

Normalizaremos los datos y obtendremos el conjunto de entrada y salida para entrenar nuestra red:

```
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler() # por defecto es 0,1

# Normalizamos el conjunto de entrenamiento

training_set = sc.fit_transform(training_set)

# Obtenemos el conjunto de entrada a la red y el conjunto de salidas,
# x_train es la entrada, y_train es la salida

X_train = training_set[0:1257]
y_train = training_set[1:1258]
X_train = np.reshape(X_train, (1257, 1, 1))
```

A continuación, vamos a construir nuestra red LSTM. Para ello lo primero que haremos será importar las librerías y paquetes necesarios como *keras*, *Sequential*, *Dense y LSTM.*

```
# Parte 2 - Construyendo la Red Neural Recurrente
# Importando las librerias y paquetes

from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.layers import LSTM

# Inicializando la RNN
# utilizaremos un modelo continuo, modelo de regresión

regressor = Sequential()

# Añadimos una capa de entrada y la capa LSTM

regressor.add(LSTM(units = 4, activation = 'sigmoid', input_shape = (None, 1)))

# Añadimos la capa de salida con una única neurona

regressor.add(Dense(units = 1))

# Compilamos la RNN
# usamos el error cuadrático medio
# MSE para regresión


regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Ajustamos la RNN al conjunto de entrenamiento

regressor.fit(X_train, y_train, batch_size = 32, epochs = 200)
```

Ahora que hemos entrenado el modelo debemos hacer la predicción y visualización de los resultados:

```
# Parte 3 – Hacer las predicciones y visualizar los resultados

# Obtener el precio real de las acciones de 2017
test_set = pd.read_csv('Google_Stock_Price_Test.csv', sep=',')

real_stock_price = test_set.iloc[:,1:2].values

# Obtener el precio de las acciones previsto para 2017

inputs = real_stock_price
inputs = sc.transform(inputs)
inputs = np.reshape(inputs, (20, 1, 1))
predicted_stock_price = regressor.predict(inputs)
predicted_stock_price = sc.inverse_transform(predicted_stock_p
predicted_stock_price = sc.inverse_transform(predicted_stock_price)

# Visualizando los resultados

plt.plot(real_stock_price, color = 'red', label = 'Real GoogleStock Price')
```

```
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock
Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Time Price')
plt.legend()
plt.show()
```

Actividad: A continuación, realizad diferentes pruebas con topologías variadas y con número de entradas distintas, manipulando el conjunto de entrenamiento tanto de salida como de entrada con distintos pasos temporales. Construir una tabla con diferentes configuraciones tanto de aprendizaje como de topologías. Tomar conclusiones de dicha tabla, seleccionar la mejor opción y explicar por qué.

A continuación, se muestra un ejemplo para ver la variedad de arquitecturas tanto de RN como de la configuración de la entrada que pueden tomarse.

Actividad: Explicad que pretende el código y modificarlo para que pueda obtener resultados más ajustados. Realizar varias ejecuciones con diferentes configuraciones y parámetros, al igual que hicimos con las redes feedforward, variando capas, neuronas, algoritmos de entrenamiento, etc. y sacar conclusiones.

```
#Creando la RNN
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout


# Inicializamos el modelo

regresor = Sequential()

# Agregando la primera capa LSTM y regularizando "Deserción"

regresor.add(LSTM(units=50, return_sequences=True,

input_shape=(X_regresor.add(Dropout(0.2))

# Agregando segunda capa LSTM

regresor.add(LSTM(units=50, return_sequences=True))
regresor.add(Dropout(0.2))
```

```
# Agregando tercera capa LSTM

regresor.add(LSTM(units=50, return_sequences=True))
regresor.add(Dropout(0.2))

# Agregando cuarta capa LSTM

regresor.add(LSTM(units=50))
regresor.add(Dropout(0.2))
#Agregando capa de salida LSTM

regresor.add(Dense(units=1))
regresor.add(Dropout(0.2))

# Compilamos el modelo indicándole el optimizador y la función de

regresor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regresor.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

Finalmente, realizamos la predicción:

```
# PARTE 3: PREDICCION Y VISUALIZACION
# Leyendo Datos para Testear

dataset_train = pd.read_csv('Google_Stock_Price_Train.csv', sep=',')
train_set = dataset_train.iloc[:,1:2].values
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv', sep=',')
test_set=dataset_test.iloc[:,1:2].values

#print ("longitud(dataset_total): ", len(dataset_total))

print ("longitud(dataset_test): ", len(dataset_test))


# Prediciendo Accion Google 2017

dataset_total=pd.concat((dataset_train['Open'],dataset_test['Open']),axis=0)
inputs = dataset_total[len(dataset_total)-len(dataset_test)-60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)

#Agregando Dimensión Extra

X_test = []
for i in range(60,80):
```

```
X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0], X_test.shape[1],1))


# Prediccion
prediccion_precio = regresor.predict(X_test)
prediccion_precio = sc.inverse_transform(prediccion_precio)
```