Universidad de los Andes
Colombia | Facultad de Ingeniería

Universidad de los Andes
Maestría en Inteligencia Analítica para la Toma de Decisiones.
Modelos Avanzados para el Análisis De Datos 2.
Prof. Martín Andrade Restrepo.

Taller 6. Redes neuronales recurrentes

Observaciones:

En este taller usaremos redes neuronales recurrentes y convolucionales para hacer predicciones temporales de temperaturas usando un dataset de datos metereológicos. Los datos fueron tomados cada 10 minutos.
En el siguiente vínculo descargue los datos: https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip

Pasos previos

1. Descomprima la carpeta y guárdela en la ubicación que desee.
2. Cree un nuevo archivo "`main`" en su editor de preferencia de Python (en lo posible, utilice Spyder o Pycharm). Recuerde utilizar el environment en el que usted ya instaló tensorflow.
3. Para cargar los datos puede utilizar las siguientes líneas:

```
import os
data_dir = El directorio donde guardo el data set
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')
f = open(fname)
data = f.read()
f.close()
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[1:]
print(header)
print(len(lines))
```

   Las últimas dos lineas imprimen las variables y el número de datos a usar.

4. Convertiremos los datos en "`arrays`" de Numpy para acelerar las operaciones:

```
import numpy as np
float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
        values = [float(x) for x in line.split(',')[1:]]
        float_data[i, :] = values
```

Ejercicio 1

Describa preliminarmente los datos que usará en su modelo (utilice estadísticas descriptivas, histogramas, etc.).

Para preprocesar los datos y extraer las secuencias con las que se entrenará la red puede usar la siguiente función:

```
def generator(data, lookback, delay, min_index, max_index, step=6):
if max_index is None:
        max_index = len(data) - delay - 1
i = min_index + lookback
rows = np.arange(i, max_index)
samples = np.zeros((len(rows),
        lookback // step,
        data.shape[-1]))
targets = np.zeros((len(rows),))
for j, row in enumerate(rows):
        indices = range(rows[j] - lookback, rows[j], step)
        samples[j] = data[indices]
        targets[j] = data[rows[j] + delay][1]
return samples, targets
```

La función genera los inputs (variable "`samples`") junto con los respectivos valores de la variable respuesta, la temperatura (variable "`targets`").

Los parámetros de la función están descritos de la siguiente forma (*Chollet, Francois. Deep learning with Python. Simon and Schuster, 2021*):

- `data` — The original array of floating-point data.
- `lookback` — How many timesteps back the input data should go.
- `delay` — How many timesteps in the future the target should be.
- `min_index` and `max_index` — Indices in the data array that delimit which timesteps to draw from. This is useful for keeping a segment of the data for validation and another for testing.
- `step` — The period, in timesteps, at which you sample data. You'll set it to 6 in order to draw one data point every hour.

Ejercicio 2

Usando la función genere los datos de entrenamiento, validación y prueba para un valor del `lookback` deseado. Se recomienda entrenar con mínimo $20,000$ elementos de la base.

Ejercicio 3

Normalice sus datos utilizando la media y la desviación estándar del conjunto de entrenamiento de cada una de sus variables.

Construcción y entrenamiento de la red

Para entrenar la red cargue las librerías usando:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Para construir la primera red puede usar la clase `layers.SimpleRNN` (las capas recurrentes más simples).

Puede tomar la siguiente red como ejemplo (la red utiliza el API secuencial, pero recuerde que también puede definir la red capa por capa utilizando la función "`layers`" incluyendo al final como parámetro la capa anterior):

```
model = keras.Sequential()
model.add(keras.Input(shape = (None, float_data.shape[-1])))
model.add(layers.SimpleRNN(32, return_sequences = True, activation = 'relu'))
model.add(layers.SimpleRNN(32, activation = 'relu'))
model.add(layers.Dense(1))
print(model.summary())
model.compile(
    loss = 'mae',
    optimizer = keras.optimizers.Adam(learning_rate = 0.001),
    metrics = ["mse"]
    )
model.fit(x_train, y_train, batch_size = 64, epochs = 10, verbose = 2,
    validation_data=(x_val, y_val))
model.evaluate(x_test, y_test, batch_size = 64, verbose = 2)
```

Note la ausencia de función de activación en la última capa densa. Esto es típico en problemas de regresión.

Además, en el momento de añadir la primera capa en el "`layers.SimpleRNN`", el primer parámetro es la cantidad de unidades y el segundo se usa para poder utilizar todos los outputs de cada secuencia en lugar de solo el último (necesario si se van a incluir varias capas recurrentes).

Ejercicio 4

1. Analice cuidadosamente el código de arriba e implementelo en su "`main`" incorporando lo que haga falta para que pueda correrse de forma correcta.
2. Experimente con las activaciones. La activación por defecto de `SimpleRNN` es "`tanh`".

Ejercicio 5

1. Investigue sobre las unidades GRU (Gated recurrent unit).
2. Para implementar las unidades GRU o LSTM puede usar: "`layers.GRU`" o "`layers.LSTM`" en lugar de "`layers.SimpleRNN`".

3. Experimente con dichas unidades y describa sus resultados. Compárelos con aquellos obtenidos con las capas `layers.SimpleRNN`.

Ejercicio 6

Para utilizar redes neuronales recurrentes bidireccionales puede utilizar:

```
model.add(
layers.Bidirectional(
layers.LSTM(256,return_sequences = True)
)
)
```

en lugar de las líneas del código original en donde se insertaba la primera capa.

1. Implemente esta capa bidireccional y compare el desempeño con sus redes anteriores.

2. Realice varios entrenamientos de sus redes variando los hiper parámetros y las arquitecturas. ¿Cuál red ofrece los mejores resultados? (Recuerde utilizar sus conjuntos de validación y prueba).