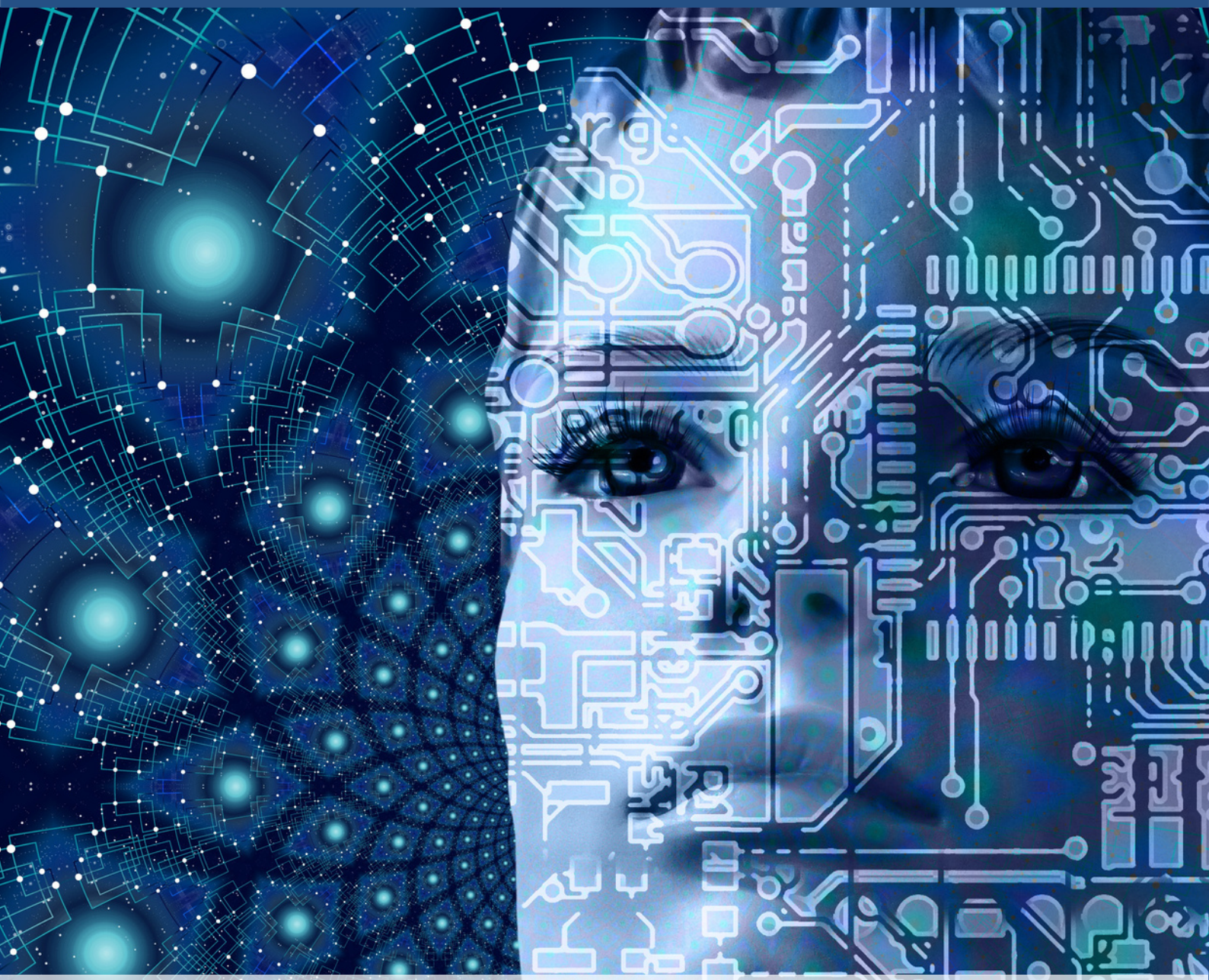
COLOMBIA POTENCIA DE LA VIDA

TIC



Construcción Redes Neuronales Recurrente en Series Temporales

TALENTO TECH | AZ PROYECTOS EDUCATIVOS | UTP Universidad Tecnológica de Pereira

Construcción Redes Neuronales Recurrente en Series Temporales

Predicción de valores en una serie de tiempo utilizando una red neuronal recurrente (RNN).

Objetivo: Comprender el proceso de construcción, entrenamiento y evaluación de una red neuronal recurrente para realizar predicciones en una serie de tiempo.

Pasos del ejercicio

1 Preprocesamiento de datos:

Se proporciona una serie de tiempo de prueba y se divide en conjuntos de entrenamiento y prueba.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN

# Paso 1: Preprocesamiento de datos (ejemplo básico)
# Supongamos que tenemos la siguiente serie de tiempo prueba
time_series = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
print("time_series: ",time_series)
train_size = int(len(time_series) * 0.6)  # Tamaño del conjunto de entrenamiento
train_data, test_data = time_series[:train_size], time_series[train_size:]
print("train_data:", train_data)
print("test_data",test_data)
```

```
time_series:  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
train_data: [1 2 3 4 5 6 7 8 9]
test_data [10 11 12 13 14 15]
```

2 Preparación de datos para LSTM:

Se implementa una función para transformar los datos en secuencias y crear conjuntos de entrenamiento y prueba en formato de secuencias.

```
# Paso 2: Preparación de datos (ejemplo básico)
# Por ejemplo, transformamos nuestros datos en secuencias
def create_dataset(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

# Definimos el número de pasos de tiempo para la serie temporal
seq_length = 4
# Creamos conjuntos de entrenamiento y prueba en formato 3D
X_train, y_train = create_dataset(train_data, seq_length)
X_test, y_test = create_dataset(test_data, seq_length)

# Imprimimos los conjuntos de datos en formato 3D
print("Conjunto de secuencias de entrenamiento:\n", X_train)
print("Etiquetas de entrenamiento:\n", y_train)
print("Conjunto de secuencias de prueba:\n", X_test)
print("Etiquetas de prueba:\n", y_test)
```

```
Conjunto de secuencias de entrenamiento:
 [[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]
 [5 6 7 8]]
Etiquetas de entrenamiento:
 [5 6 7 8 9]
Conjunto de secuencias de prueba:
 [[10 11 12 13]
 [11 12 13 14]]
Etiquetas de prueba:
 [14 15]
```

3 Construcción y Compilación del modelo:

Se construye un modelo de red neuronal recurrente (RNN) utilizando una capa SimpleRNN y una capa Dense para la predicción. Después, se compila el modelo especificando el optimizador y la función de pérdida.

```
# Paso 3: Construcción del modelo
model = Sequential()
model.add(SimpleRNN(16, activation='relu', input_shape=(seq_length, 1)))
model.add(Dense(1))

# Paso 4: Compilacion del modelo
model.compile(optimizer='adam', loss='mse')  # Compilación del modelo
```

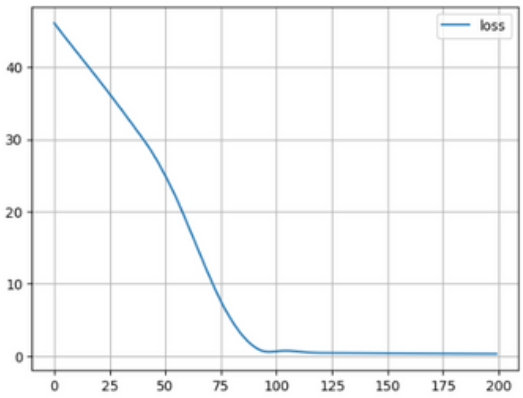
4 Entrenamiento del modelo:

Se entrena el modelo utilizando los datos de entrenamiento y se grafica el historial del entrenamiento.

```
# Paso 5: Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=200)

#  Paso 6: Graficar el historial de entrenamiento:
pd.DataFrame(history.history).plot(grid=True)
```

```
Epoch 195/200
1/1 [==============================] - 0s 12ms/step - loss: 0.3253
Epoch 196/200
1/1 [==============================] - 0s 8ms/step - loss: 0.3240
Epoch 197/200
1/1 [==============================] - 0s 7ms/step - loss: 0.3227
Epoch 198/200
1/1 [==============================] - 0s 11ms/step - loss: 0.3214
Epoch 199/200
1/1 [==============================] - 0s 8ms/step - loss: 0.3202
Epoch 200/200
1/1 [==============================] - 0s 11ms/step - loss: 0.3189
```



5 Evaluación del modelo:

Se evalúa el rendimiento del modelo utilizando los datos de prueba.

```
# Paso 7: Evaluación del modelo
Evaluación = model.evaluate(X_test, y_test)
print(Evaluación)
```

```
1/1 [==============================] - 0s 203ms/step - loss: 7.3801
7.380114555358887
```

6 Predicción:

Se realiza la predicción de valores utilizando el modelo entrenado.

```
# Paso 8: Predicción
print("X_test: \n",X_test)
print("y_test: \n",y_test)
y_pred = model.predict(X_test)
print("y_pred: \n",y_pred)
```

```
X_test:
 [[10 11 12 13]
 [11 12 13 14]]
y_test:
 [14 15]
1/1 [==============================] - 0s 142ms/step
y_pred:
 [[16.513128]
 [17.905928]]
```

Actividad

para los estudiantes

Ejercicio: Análisis de Descomposición Estacional

Instrucciones:

- Analizar el proceso de preprocesamiento de datos y preparación de datos para LSTM.
- Estudiar la arquitectura del modelo y comprender cómo se configuran las capas.
- Observar el proceso de compilación del modelo y comprender los parámetros utilizados.
- Realizar el entrenamiento del modelo y analizar el historial de entrenamiento obtenido.
- Evaluar el rendimiento del modelo utilizando los datos de prueba y analizar los resultados.
- Realizar predicciones utilizando el modelo entrenado y compararlas con los valores reales.

Entregable del ejercicio

- Análisis del proceso de preprocesamiento de datos y preparación de datos.
- Descripción de la arquitectura del modelo y explicación de los parámetros utilizados en la compilación del modelo.
- Reporte de la evaluación del modelo, incluyendo la función de pérdida y cualquier métrica adicional utilizada.
- Comparación entre las predicciones realizadas por el modelo y los valores reales, con gráficos que muestren esta comparación.

Este ejercicio proporcionará a los estudiantes una comprensión práctica del uso de redes neuronales recurrentes (RNN), en particular de la arquitectura del modelado y la predicción de series de tiempo. Además, les permitirá familiarizarse con el proceso completo de construcción, entrenamiento y evaluación de modelos de aprendizaje automático en este contexto.