Modulo 2 - 8. Dense + Dropout + Batch Normalization

Por Juan Pablo Bernal Lafarga - A01742342

Use the Student GPA dataset to predict student GPA.

Use previous concepts to create different Neural Network Architectures and compare your results. (Python Notebook)

El propósito del siguiente bloque es cargar y preprocesar un conjunto de datos de rendimiento estudiantil seleccionando variables relevantes dividiendo en conjuntos de entrenamiento y prueba estandarizando las variables predictoras y verificando el tamaño del conjunto de datos de entrenamiento para preparar el entrenamiento de una red neuronal que prediga el GPA de los estudiantes

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,BatchNormalization
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("Student_performance_data _.csv")
dataset = data[['Age','ParentalEducation','StudyTimeWeekly','Absences','Tutc

X = dataset.drop('GPA', axis = 1)
y = dataset['GPA']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.fit_transform(X_test)

np.shape(X_train)
```

Out[ ]:
```
(1913, 11)
```

Experiment 1: A single Dense Hidden Layer

In [ ]:
```
model = Sequential()
model.add(Dense(64, input_dim = 11, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse', metrics=['mean_absolute_error'])

history = model.fit(X_train_std, y_train, epochs = 25, batch_size=5, validat
```

Experiment 2: A set of three Dense Hidden Layers

In [ ]:
```
model2 = Sequential()
model2.add(Dense(64, input_dim = 11, activation='relu'))
model2.add(Dense(32))
model2.add(Dense(16))
model2.add(Dense(1))

model2.compile(optimizer='adam', loss='mse', metrics=['mean_absolute_error']

history2 = model2.fit(X_train_std, y_train, epochs = 25, batch_size=5, valic
```

Experiment 3: Add a dropout layer after each Dense Hidden Layer

In [ ]:
```
model3 = Sequential()
model3.add(Dense(64, input_dim = 11, activation='relu'))
model3.add(Dropout(0.15))
model3.add(Dense(32))
model3.add(Dropout(0.15))
model3.add(Dense(16))
model3.add(Dropout(0.15))
model3.add(Dense(1))

model3.compile(optimizer='adam', loss='mse', metrics=['mean_absolute_error']

history3 = model3.fit(X_train_std, y_train, epochs = 25, batch_size=5, valic
```

Experiment 4: Add a Batch Normalization Layer after each Dropout Layer.

In [ ]:
```
model4 = Sequential()
model4.add(Dense(64, input_dim = 11, activation='relu'))
model4.add(Dropout(0.2)) # Tira el 20%
model4.add(BatchNormalization())
```

```
model4.add(Dense(32))
model4.add(Dropout(0.2))
model4.add(BatchNormalization())
model4.add(Dense(16))
model4.add(Dropout(0.2))
model4.add(BatchNormalization())
model4.add(Dense(1))

model4.compile(optimizer='adam', loss='mse', metrics=['mean_absolute_error']

history4 = model4.fit(X_train_std, y_train, epochs = 25, batch_size=5, valic
```

Create a comparative table

In [20]:
```
losses = [history.history['loss'][-1], history2.history['loss'][-1], history
maes = [history.history['mean_absolute_error'][-1], history2.history['mean_a

for i in range(len(losses)):
    print(f'Modelo {i+1}. Loss: {losses[i]}, MAE: {maes[i]} \n')
```

```
Modelo 1. Loss: 0.029858337715268135, MAE: 0.13903826475143433

Modelo 2. Loss: 0.03181667998433113, MAE: 0.14311639964580536

Modelo 3. Loss: 0.07811465859413147, MAE: 0.21685998141765594

Modelo 4. Loss: 0.29006633162498474, MAE: 0.42743393778800964
```

Como podemos observar, el modelo con menor MSE (pérdida) y menor MAE es el del primer experimento con una simple hidden layer. Significando que las predicciones del modelo están, en promedio, bastante cerca de los valores reales. Mientras que el modelo con mayor MSE y MAE es el experimento 4 con dropout y batch normalization, seguido por el experimento 3 con dropout. Esto sugiere que el problema de predicción de calificaciones no requiere una arquitectura compleja y podría estar beneficiándose de la simplicidad, evitando así el riesgo de sobreajuste que puede ocurrir con capas adicionales o técnicas como el dropout y batch normalization.