

8. Dense + Dropout + Batch Normalization

Cargamos la base de datos

```
from google.colab import files
uploaded = files.upload()
```

Elegir archivos

Student_perf...ce_data_.csv

- **Student_performance_data_.csv**(text/csv) - 166901 bytes, last modified: 4/10/2024 - 100% done

Save Student_performance_data_.csv to Student_performance_data_.csv

importamos librerías necesarias

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.callbacks import EarlyStopping
```

Leemos los datos

```
data = pd.read_csv("Student_performance_data_.csv")
data.head()
```

E

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences	Tutoring	ParentalSupport	Extracurricular
0	1001	17	1	0	2	19.833723	7	1	2	0
1	1002	18	0	0	1	15.408756	0	0	1	0
2	1003	15	0	2	3	4.210570	26	0	2	0
3	1004	17	1	0	3	10.028829	14	0	3	0
4	1005	17	1	0	2	4.672495	17	1	3	0

Pasos siguientes:

Generar código con data

Ver gráficos recomendados

New interactive sheet

```
data.info()
```

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2392 entries, 0 to 2391

Data columns (total 15 columns):

#	Column	Non-Null Count	Dtype
0	StudentID	2392 non-null	int64
1	Age	2392 non-null	int64
2	Gender	2392 non-null	int64
3	Ethnicity	2392 non-null	int64
4	ParentalEducation	2392 non-null	int64
5	StudyTimeWeekly	2392 non-null	float64
6	Absences	2392 non-null	int64
7	Tutoring	2392 non-null	int64
8	ParentalSupport	2392 non-null	int64
9	Extracurricular	2392 non-null	int64

```

10 Sports          2392 non-null  int64
11 Music           2392 non-null  int64
12 Volunteering    2392 non-null  int64
13 GPA             2392 non-null  float64
14 GradeClass      2392 non-null  float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB

```

```
data = data.drop(["StudentID", "Ethnicity", "Volunteering"], axis=1)
```

Seleccionamos nuestra variable objetivo y dividimos los datos en entrenamiento y prueba y los normalizamos.

```

X = data.drop(columns="GPA")
y = data["GPA"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

El early stopping es una técnica que nos ayuda a prevenir el sobreajuste (overfitting).

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```


✓ Experiment 1: A single Dense Hidden Layer

```

model1 = Sequential()
model1.add(Dense(64, activation="relu", input_shape=(X_train.shape[1],)))
model1.add(Dense(1)) # Output layer

model1.compile(optimizer="adam", loss="mean_squared_error", metrics=["mae"])

```

 /usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape` argument to the `Dense` layer.
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

✓ Experiment 2: A set of three Dense Hidden Layers

```

model2 = Sequential()
model2.add(Dense(64, activation="relu", input_shape=(X_train.shape[1],)))
model2.add(Dense(32, activation="relu"))
model2.add(Dense(16, activation="relu"))
model2.add(Dense(1))

model2.compile(optimizer="adam", loss="mean_squared_error", metrics=["mae"])

```

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

```

model3 = Sequential()
model3.add(Dense(64, activation="relu", input_shape=(X_train.shape[1],)))
model3.add(Dropout(0.3)) # Dropout layer with 30% dropout rate

```

```

model3.add(Dense(32, activation="relu"))
model3.add(Dropout(0.3))
model3.add(Dense(16, activation="relu"))
model3.add(Dropout(0.3))
model3.add(Dense(1))

model3.compile(optimizer="adam", loss="mean_squared_error", metrics=["mae"])

```

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

```

model4 = Sequential()
model4.add(Dense(64, activation="relu", input_shape=(X_train.shape[1],)))
model4.add(Dropout(0.3))
model4.add(BatchNormalization())
model4.add(Dense(32, activation="relu"))
model4.add(Dropout(0.3))
model4.add(BatchNormalization())
model4.add(Dense(16, activation="relu"))
model4.add(Dropout(0.3))
model4.add(BatchNormalization())
model4.add(Dense(1))

model4.compile(optimizer="adam", loss="mean_squared_error", metrics=["mae"])

```

✎ Comparative table

```

# Train and evaluate each model
models = {"Experiment 1": model1, "Experiment 2": model2, "Experiment 3": model3, "Experiment 4": model4}
results = []

```

```

for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train, y_train, epochs=100, validation_split=0.2, batch_size=32,
              callbacks=[early_stopping], verbose=0)

```

```

# Evaluate model
y_pred = model.predict(X_test).flatten()
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

```

```

# Results
results.append({
    "Experiment": name,
    "MSE": mse,
    "MAE": mae,
    "R^2 Score": r2
})

```

```

↗ Training Experiment 1...
15/15 ————— 0s 4ms/step
Training Experiment 2...
15/15 ————— 0s 12ms/step
Training Experiment 3...
15/15 ————— 0s 4ms/step
Training Experiment 4...
15/15 ————— 0s 8ms/step
[{'Experiment': 'Experiment 1',
  'MSE': 0.04359728704558086,

```

```
'MAE': 0.1675882205717592,
'R^2 Score': 0.9472783074390577},
{'Experiment': 'Experiment 2',
'MSE': 0.3021403070778817,
'MAE': 0.4202616071432048,
'R^2 Score': 0.6346252379563282},
{'Experiment': 'Experiment 3',
'MSE': 0.5896985304513076,
'MAE': 0.6610443984791654,
'R^2 Score': 0.2868844202716364},
{'Experiment': 'Experiment 4',
'MSE': 3.1045403152464957,
'MAE': 1.5977685158016397,
'R^2 Score': -2.7542845239969385}]
```

```
results_df = pd.DataFrame(results)
print(results_df)
```

	Experiment	MSE	MAE	R^2 Score
0	Experiment 1	0.043597	0.167588	0.947278
1	Experiment 2	0.302140	0.420262	0.634625
2	Experiment 3	0.589699	0.661044	0.286884
3	Experiment 4	3.104540	1.597769	-2.754285

El primer modelo (Experimento 1) logró los mejores resultados, esto nos indica que un modelo más simple con una sola capa oculta fue capaz de aprender bien la relación que existe entre las características y el GPA dando una predicción precisa. En el segundo modelo aumentó su valor en MSE y MAE y disminuyó en R², lo que nos indica que un modelo más complejo no resultó ser mejor, agregar profundidad a la red parece haber introducido un mayor riesgo de sobreajuste o complejidad innecesaria sin mejorar el desempeño en comparación con el modelo más sencillo. El tercer modelo a pesar de incluir capas de Dropout para reducir el sobreajuste presentó un peor desempeño, la introducción de dropout podría haber reducido la capacidad del modelo para aprender adecuadamente de los datos y finalmente el último modelo resulta ser el peor, demostrando un desempeño deficiente, no fue capaz de generalizar correctamente los datos, la combinación de dropout y normalización parece haber sobreajustado aún más el modelo.

En conclusión el modelo más simple mostró el mejor desempeño, lo que sugiere que la complejidad adicional en los otros modelos no proporcionó mejoras significativas.