



✓ Taller: Redes Neuronales


En este taller podrán poner en práctica sus conocimientos sobre la construcción e implementación de redes neuronales de una capa y multicapa. El taller está constituido por 4 puntos, en los cuales deberán seguir las instrucciones de cada numeral para su desarrollo.

✓ Datos predicción probabilidad de admisión a la universidad

En este taller se usará el conjunto de datos de admisiones a una universidad proveniente de la base de datos de Kaggle. Cada observación contiene la probabilidad de que un estudiante sea admitido por la universidad, dadas distintas variables predictoras como el puntaje del examen TOEFL y GRE, el promedio (GPA), entre otras. El objetivo es predecir la probabilidad de admisión de cada estudiante. Para más detalles pueden visitar el siguiente enlace: [datos](#).

```
import warnings
warnings.filterwarnings('ignore')
```

```
# Importación librerías
import numpy as np
import keras
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
!pip install livelossplot
from sklearn.model_selection import train_test_split
from keras import initializers
from livelossplot import PlotLossesKeras
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.callbacks import History
```

 Requirement already satisfied: livelossplot in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: bokeh in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: contourpy>=1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: numpy>=1.16 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: packaging>=16.8 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: tornado>=5.1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: xyzservices>=2021.09.1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
 Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages

```
# Carga de datos de archivo .csv
data = pd.read_csv('https://raw.githubusercontent.com/albahnsen/MIAD_ML_and_NLP/main/data.csv')
data.head()
```



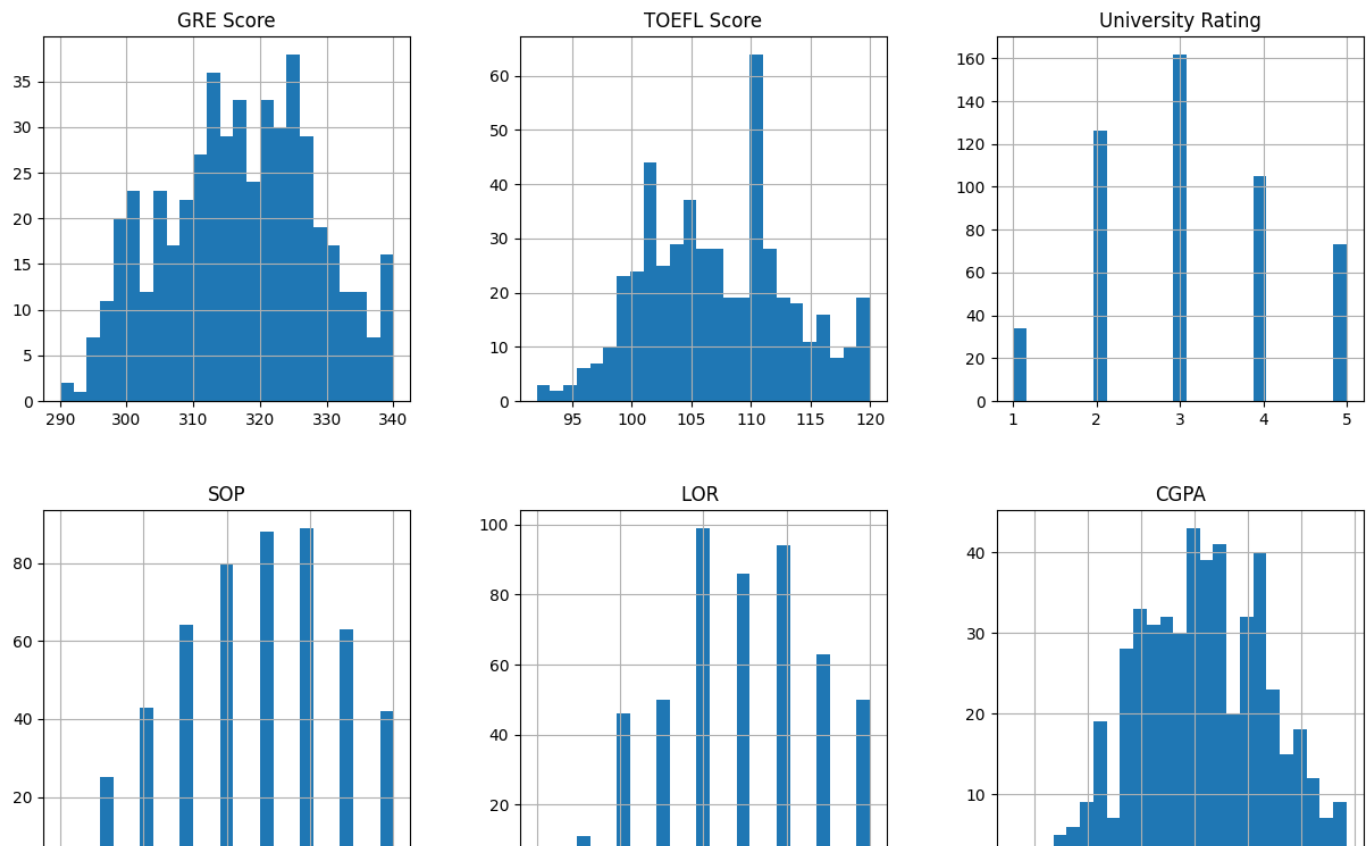
Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87	1	0.76
3	316	104	3	3.0	3.5	8.00	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.80

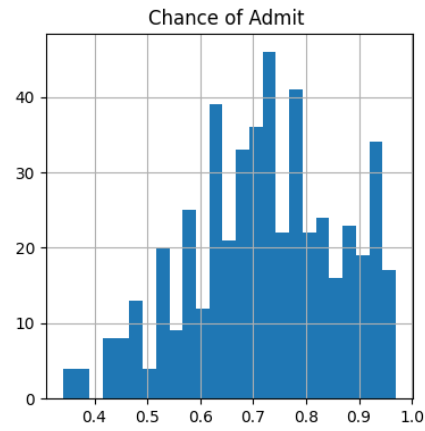
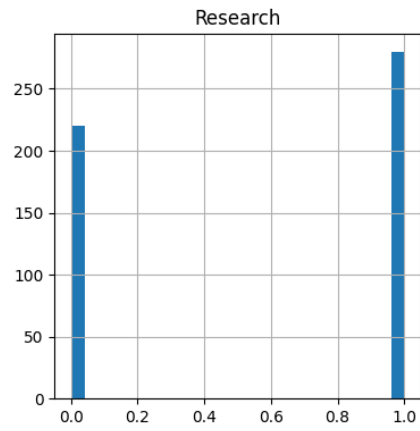
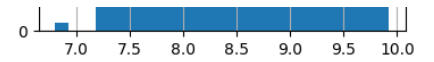
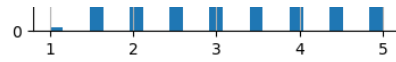
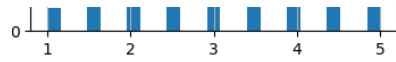


Next steps:

[Generate code with data](#)
☒ [View recommended plots](#)

```
# Histogramas de las variables del dataset
data.hist(bins=25, figsize=(15,15))
plt.show()
```





```
# Definición de variables predictoras (X)
X = data.drop(data.columns[-1], axis=1)
# Definición de variable de interés (y)
Y = data[data.columns[-1]]
```

```
# Separación de variables predictoras (X) y variable de interés (y) en set de ent
xTrain, xTest, yTrain, yTest = train_test_split(X,Y,test_size=0.3, random_state=2)
```

```
# Estandarizamos/normalizamos las variables predictoras X con StandardScaler
from sklearn.preprocessing import StandardScaler
```

```
# Definición de función
scaler = StandardScaler()
scaler.fit(X)
```

```
# Transformación del set de train y test
xTrain = pd.DataFrame(data=scaler.transform(xTrain), columns=xTrain.columns, index=xTrain.index)
xTest = pd.DataFrame(data=scaler.transform(xTest), columns=xTest.columns, index=xTest.index)
```

```
xTrain = np.array(xTrain)
yTrain = np.array(yTrain)
```

```
xTest = np.array(xTest)
yTest = np.array(yTest)
```

```
output_var = 1
print(output_var, 'output variables')
```

➡ 1 output variables

```
dims = xTrain.shape[1]
print(dims, 'input variables')
```

➡ 7 input variables

✓ Punto 1 - Red Neuronal de una capa

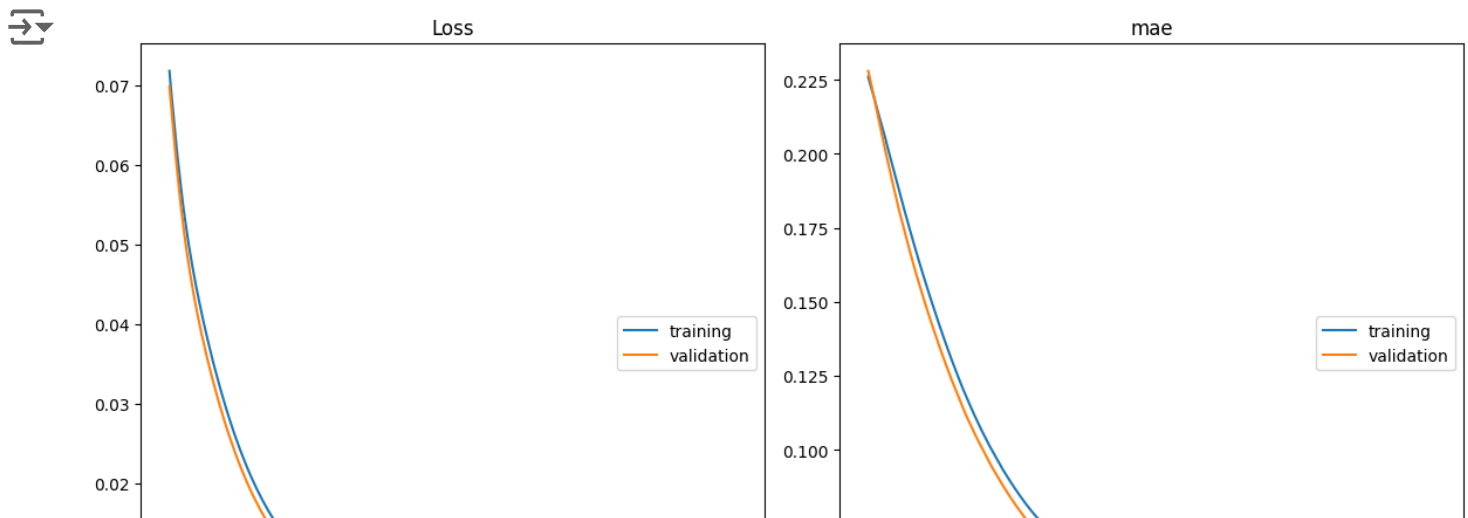
En la celda 1 creen una **red neuronal de una capa** con la librería Keras, que prediga la probabilidad de admisión de los estudiantes, usando los sets de entrenamiento y test definidos anteriormente. Pueden usar la función de pérdida, el optimizador y el número de épocas que consideren pertinentes para el modelo, justificando su selección. Finalmente, grafiquen la pérdida del modelo vs el número de épocas en el set de entrenamiento y validación, y presenten el desempeño del modelo con las métricas error absoluto medio (MAE) y error cuadrático medio (MSE).

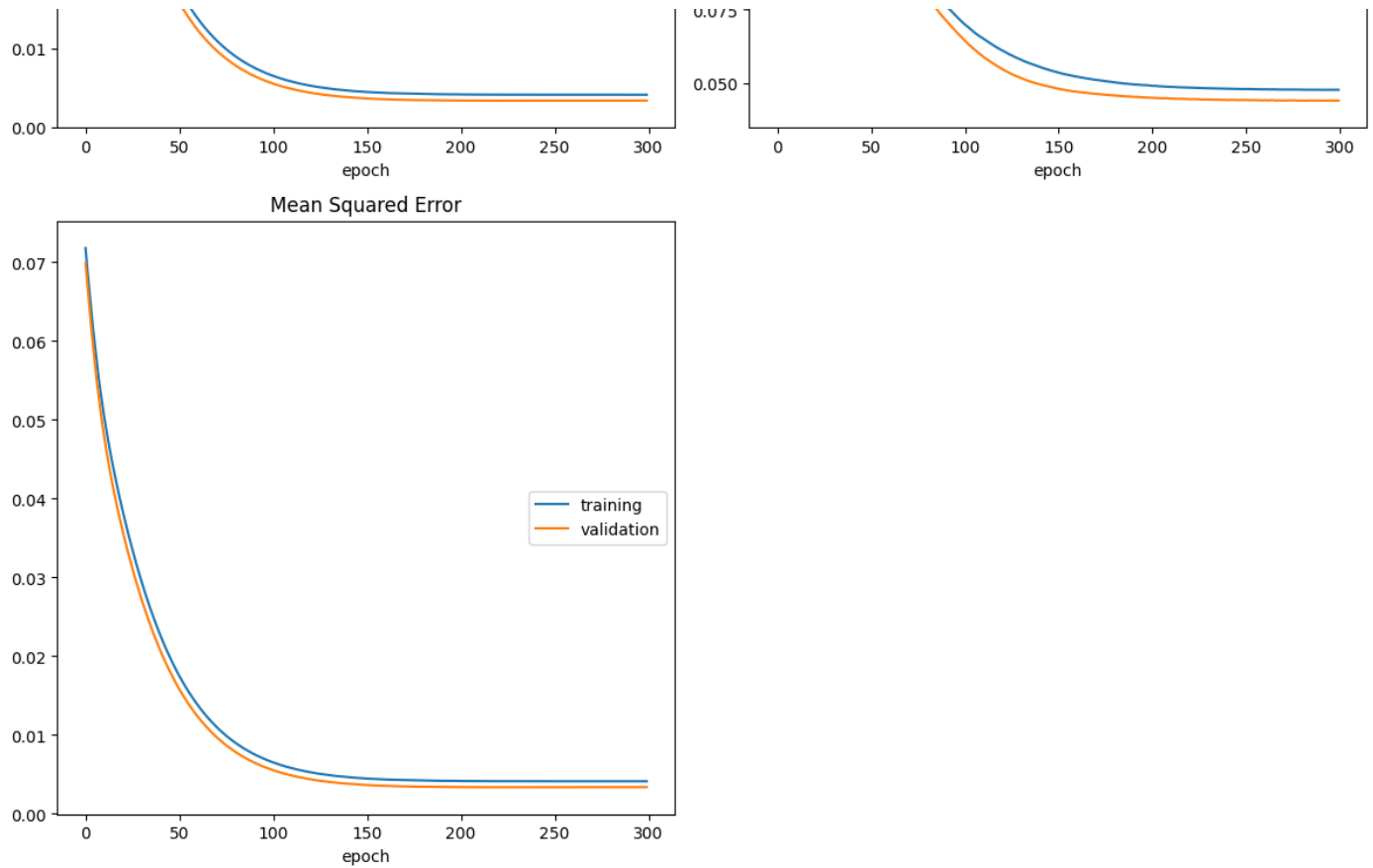
```
# Celda 1
from keras import backend as K

K.clear_session()
model = Sequential()
model.add(Dense(1, input_dim=xTrain.shape[1], activation='sigmoid'))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae', 'mse'])

model.fit(xTrain, yTrain,
          verbose=1,
          epochs=300,
          validation_data=(xTest, yTest),
          callbacks=[PlotLossesKeras()])

loss, mae_one, mse_one = model.evaluate(xTest, yTest, verbose=1)
print("Error absoluto medio (MAE):", mae_one)
print("Error cuadrático medio (MSE):", mse_one)
```





Loss

training	(min: 0.004, max: 0.072, cur: 0.004)
validation	(min: 0.003, max: 0.070, cur: 0.003)

mae

training	(min: 0.048, max: 0.226, cur: 0.048)
validation	(min: 0.044, max: 0.228, cur: 0.044)

Mean Squared Error

training	(min: 0.004, max: 0.072, cur: 0.004)
validation	(min: 0.003, max: 0.070, cur: 0.003)

11/11 [=====] - 1s 89ms/step - loss: 0.0041 - mae: 0.043
 5/5 [=====] - 0s 5ms/step - loss: 0.0034 - mae: 0.043

Error absoluto medio (MAE): 0.043934933841228485

Error cuadrático medio (MSE): 0.003389120800420642

Las pérdidas en el training y validation disminuyeron, lo cual nos da a entender que el modelo está aprendiendo de manera efectiva. Los valores de pérdida de validación son parecidos a los de entrenamiento, lo cual es un buen indicador de que no hay sobreajuste muy alto.

Adicionalmente, las curvas de pérdidas de entrenamiento y validación convergen en valores cercanos, lo cual indica que el número de épocas utilizado es el adecuado. El MAE y el MSE en los datos de entrenamiento son bajos, lo cual nos da a comprender que las predicciones obtenidas están cercanas a los valores reales.

✓ Punto 2 - Red Neuronal multicapa

En la celda 2 creen una **red neuronal con dos capas** con la librería Keras, que prediga la probabilidad de admisión de los estudiantes usando los sets de entrenamiento y test definidos anteriormente. Pueden usar la función de pérdida, el optimizador, el número de épocas y el número de neuronas que consideren pertinentes para el modelo, justificando su selección. Finalmente, grafiquen la pérdida del modelo vs el número de épocas en el set de entrenamiento y validación, y presenten el desempeño del modelo con las métricas error absoluto medio (MAE) y error cuadrático medio (MSE).

```
# Celda 2
from keras.layers import Dense, Activation
```



```

from keras import backend as K
#Limpiar la session
K.clear_session()

# Definición red neuronal con la función Sequential()
model = Sequential()

# Definición de las dimensiones de entrada y salida
input_var = xTrain.shape[1]
output_var = yTrain.shape[0]

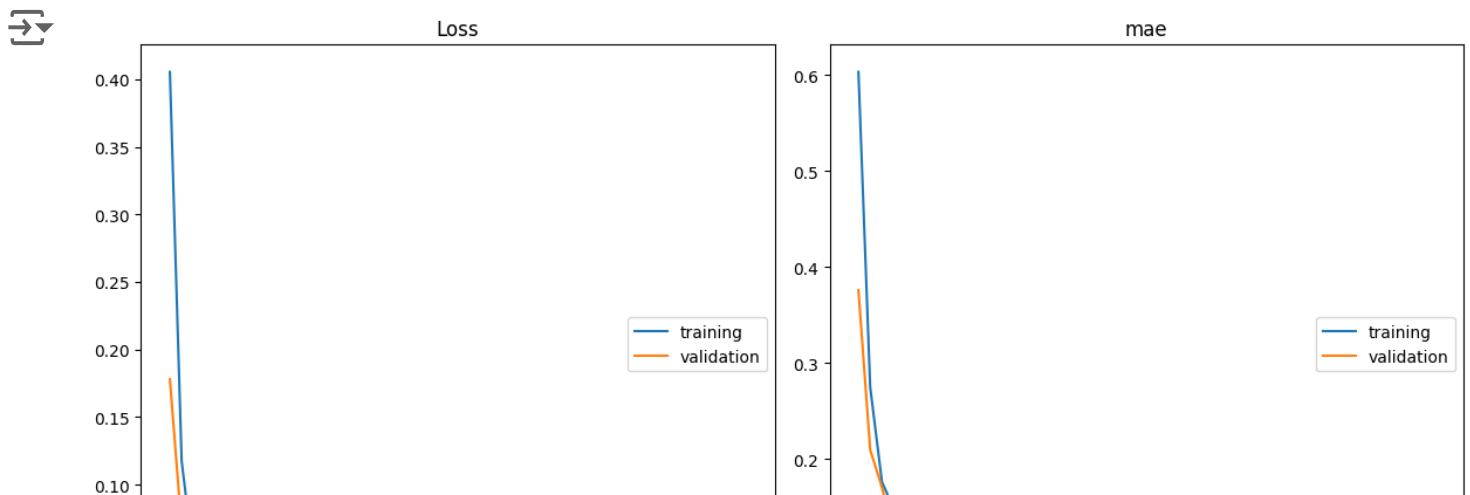
# 2 capas y una de salida
model.add(Dense(250, input_shape=(input_var,),activation='relu'))
model.add(Dense(250,activation='relu'))
model.add(Dense(output_var))

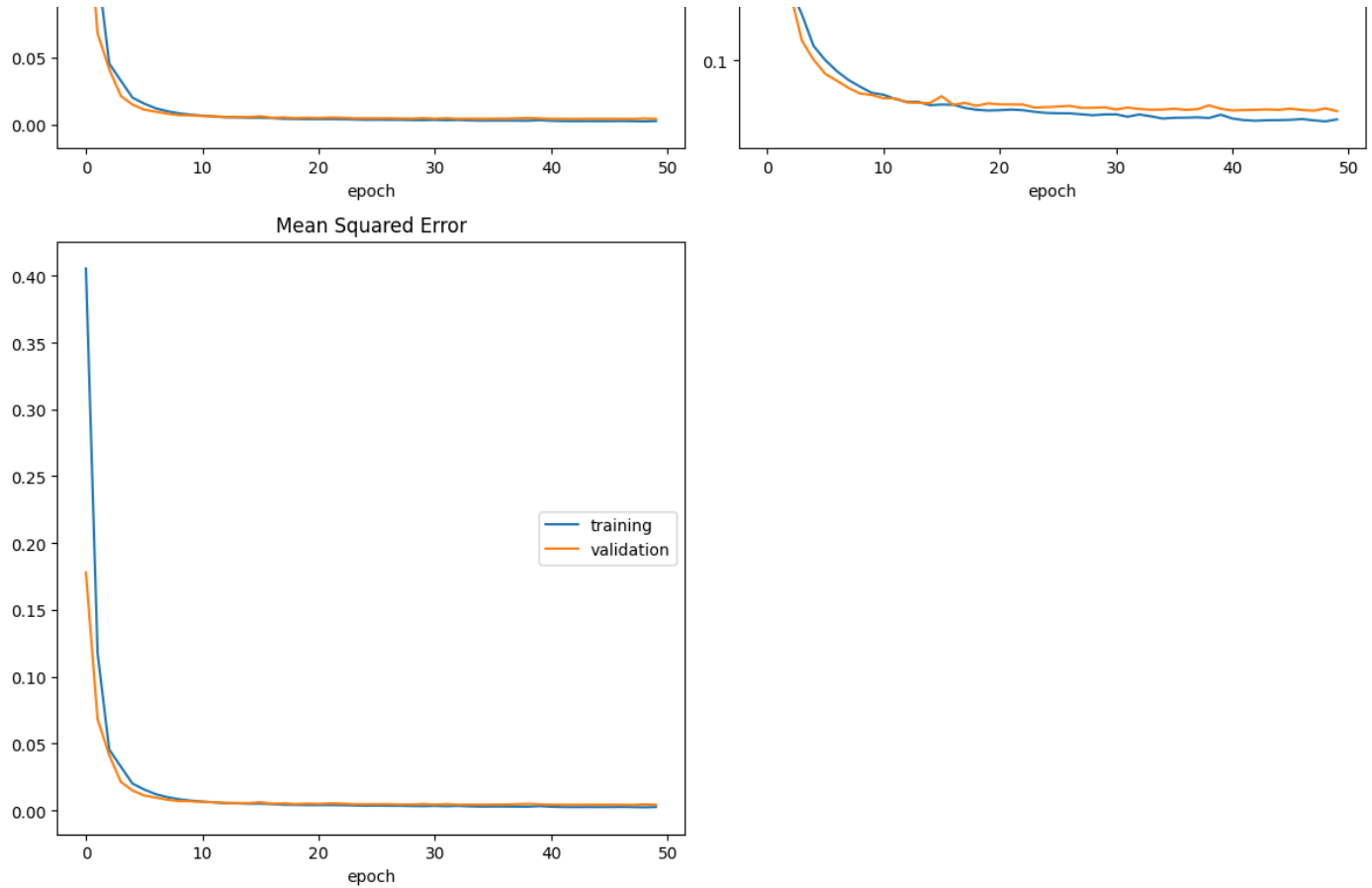
# Definición de función de perdida. Se usa mean_squared_error dado que es un ejer
#model.compile(optimizer='sgd', loss='mean_squared_error')
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae', 'mse'])
# Separación de datos de entrenamiento para considerar un set de validación duran
X_train, X_val, Y_train, Y_val = train_test_split(xTrain, yTrain, test_size=0.15,
# Entrenamiento de la red neuronal con 50 épocas
model.fit(X_train, Y_train,
          validation_data = (X_val, Y_val),
          epochs=50,
          callbacks=[PlotLossesKeras()])

#métricas error absoluto medio (MAE) y error cuadrático medio (MSE)
loss, mae_mul, mse_mul = model.evaluate(xTest, yTest, verbose=1)

print("Error Absoluto Medio (MAE):", mae_mul)
print("Error Cuadrático Medio (MSE):", mse_mul)

```





Loss

training	(min: 0.002, max: 0.405, cur: 0.003)
validation	(min: 0.004, max: 0.178, cur: 0.004)

mae

training	(min: 0.036, max: 0.603, cur: 0.038)
validation	(min: 0.046, max: 0.376, cur: 0.046)

Mean Squared Error

training	(min: 0.002, max: 0.405, cur: 0.003)
validation	(min: 0.004, max: 0.178, cur: 0.004)

10/10 [=====] - 1s 96ms/step - loss: 0.0025 - mae: 0.

5/5 [=====] - 0s 3ms/step - loss: 0.0048 - mae: 0.051

Error Absoluto Medio (MAE): 0.05114066228270531

Error Cuadrático Medio (MSE): 0.004827346187084913

Los resultados de la grafica de perdida indican un buen aprendizaje inicial y una adecuada generalización. Ambas pérdidas disminuyen muy rápidamente en las primeras épocas, y la pérdida de validación se estabiliza después de una caída inicial, lo cual puede sugerir que el modelo ha alcanzado su capacidad de aprendizaje con la configuración actual.

Capas del Modelo: Se añaden tres capas Dense. Las primeras dos capas tienen 250 neuronas cada una y utilizan la función de activación ReLU. La tercera capa (capa de salida) tiene un número de neuronas igual a `output_var`

Para elegir el numero de neuronas se corrio el modelo con varios valores, y se decide utilizar 250 neuronas ya que genera un buen resultado con un buen costo computacional. Estos son algunos de los valores obtenidos.

250 neuronas MAE 0,1038 MSE 0,0179

100 Neuronas MAE: 0,11740 MSE: 0,02376

50 Neuronas MAE 0.1266154 MSE 0.02766

Para el modelo se utilizo el optimizador adam y la función de pérdida mean_squared_error, por ser un problema de regresión. Además, se monitorean las métricas de mae (error absoluto medio) y mse (error cuadrático medio). El modelo se entrena usando los datos de entrenamiento y validación durante 50 épocas y segun los resultados se evidencia que es un numero adecuado ya que aunque se alcanzo convergencia en etapas tempranas, las 50 etapas elegidas no representan un costo computacional alto y permiten asegurar que el modelo continua en convergencia para etapas posteriores.

✓ Punto 3 - Red Neuronal multicapa

En la celda 3 creen **una red neuronal con más de una capa con la librería Keras, usando early stopping y dropout**, que prediga la probabilidad de admisión de los estudiantes con los sets de entrenamiento y test definidos anteriormente. Pueden usar la función de perdida, el optimizador, el número de épocas y el número de neuronas que consideren pertinentes para el modelo, justificando su selección. Finalmente, grafiquen la pérdida del modelo vs el número de épocas en el set de entrenamiento y validación, y presenten el desempeño del modelo con las métricas error absoluto medio (MAE) y error cuadrático medio (MSE).

```
# Celda 3
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import backend as K
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt # Asegúrate de tener esta importación

# Limpiar la sesión anterior de Keras para asegurarnos de que el modelo se inicie
K.clear_session()

# Definición red neuronal con la función Sequential()
model = Sequential()
input_var = xTrain.shape[1] # Número de características de entrada
output_var = 1              # Probabilidad de admisión (una salida)

# Añadir capas al modelo
model.add(Dense(128, input_shape=(input_var,), activation='relu'))
model.add(Dropout(0.5)) # Dropout para reducir el sobreajuste
```

```

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5)) # Segundo dropout
model.add(Dense(output_var, activation='sigmoid')) # Capa de salida con activaci

# Compilación del modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae', 'mse'])

# Callbacks para el entrenamiento
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
fBestModel = 'best_model.keras' # Nombre del archivo para guardar el mejor modelo
best_model = ModelCheckpoint(fBestModel, verbose=0, save_best_only=True)

# Entrenamiento del modelo
history = model.fit(X_train, Y_train,
                    batch_size=128,
                    epochs=50,
                    validation_data=(xTest, yTest),
                    callbacks=[best_model, early_stop],
                    verbose=1)

# Evaluación del modelo
loss, mae_mul2, mse_mul2 = model.evaluate(xTest, yTest, verbose=1)
print("Error Absoluto Medio (MAE):", mae_mul2)
print("Error Cuadrático Medio (MSE):", mse_mul2)

# Gráficas de la pérdida del modelo vs el número de épocas en el set de entrenami
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

```

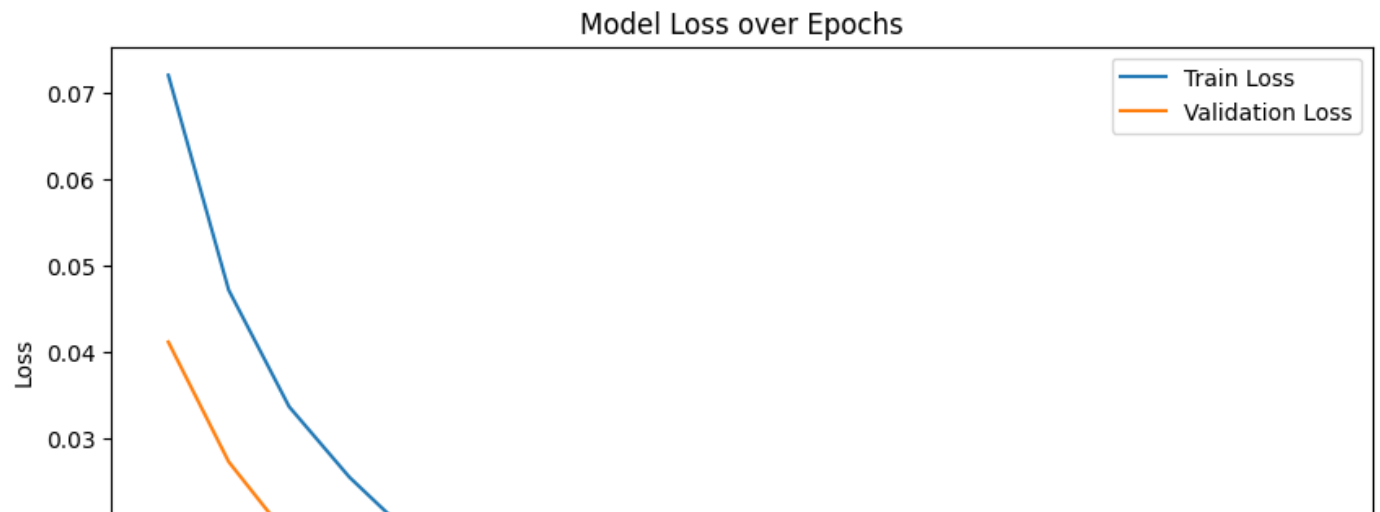
↔ Epoch 1/50
3/3 [=====] - 1s 155ms/step - loss: 0.0721 - mae: 0.2
Epoch 2/50
3/3 [=====] - 0s 58ms/step - loss: 0.0471 - mae: 0.18
Epoch 3/50
3/3 [=====] - 0s 52ms/step - loss: 0.0336 - mae: 0.15
Epoch 4/50
3/3 [=====] - 0s 52ms/step - loss: 0.0254 - mae: 0.13

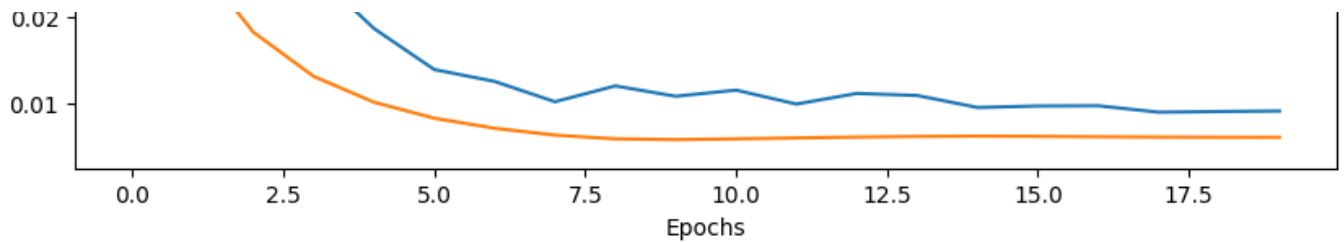
```

```

Epoch 5/50
3/3 [=====] - 0s 44ms/step - loss: 0.0188 - mae: 0.11
Epoch 6/50
3/3 [=====] - 0s 44ms/step - loss: 0.0140 - mae: 0.09
Epoch 7/50
3/3 [=====] - 0s 58ms/step - loss: 0.0126 - mae: 0.08
Epoch 8/50
3/3 [=====] - 0s 46ms/step - loss: 0.0102 - mae: 0.07
Epoch 9/50
3/3 [=====] - 0s 48ms/step - loss: 0.0121 - mae: 0.08
Epoch 10/50
3/3 [=====] - 0s 55ms/step - loss: 0.0109 - mae: 0.08
Epoch 11/50
3/3 [=====] - 0s 28ms/step - loss: 0.0116 - mae: 0.08
Epoch 12/50
3/3 [=====] - 0s 35ms/step - loss: 0.0100 - mae: 0.08
Epoch 13/50
3/3 [=====] - 0s 36ms/step - loss: 0.0112 - mae: 0.08
Epoch 14/50
3/3 [=====] - 0s 35ms/step - loss: 0.0110 - mae: 0.08
Epoch 15/50
3/3 [=====] - 0s 34ms/step - loss: 0.0096 - mae: 0.08
Epoch 16/50
3/3 [=====] - 0s 37ms/step - loss: 0.0097 - mae: 0.07
Epoch 17/50
3/3 [=====] - 0s 27ms/step - loss: 0.0098 - mae: 0.08
Epoch 18/50
3/3 [=====] - 0s 32ms/step - loss: 0.0090 - mae: 0.07
Epoch 19/50
3/3 [=====] - 0s 24ms/step - loss: 0.0091 - mae: 0.07
Epoch 20/50
3/3 [=====] - 0s 28ms/step - loss: 0.0092 - mae: 0.07
Epoch 20: early stopping
5/5 [=====] - 0s 4ms/step - loss: 0.0061 - mae: 0.063
Error Absoluto Medio (MAE): 0.06311830878257751
Error Cuadrático Medio (MSE): 0.006107145454734564

```





Se realizó un modelo con un total de tres capas densas (Dense layers) y dos capas intermedias de Dropout.

Grafica de perdida: La grafica de perdida nos muestra que al inicio del entrenamiento, tanto la pérdida de entrenamiento como la de validación disminuyen rápidamente, lo que indica que el modelo está aprendiendo efectivamente de los datos. Después de las primeras épocas, las pérdidas se estabilizan y convergen lo que muestra un resultado positivo. El modelo ha aprendido de manera efectiva, minimizando tanto la pérdida de entrenamiento como la de validación, y no muestra signos claros de sobreajuste, dado que la pérdida de validación y de entrenamiento convergen.

Estructura del Modelo Se usa el modelo Sequential de Keras. Primera Capa Densa: Tiene 128 neuronas y utiliza ReLU (Rectified Linear Unit) como función de activación. Esta capa recibe directamente los datos de entrada, por lo que su `input_shape` debe coincidir con el número de características de los datos (definido por `input_var`). Segunda Capa Densa: Consiste en 64 neuronas, también con activación ReLU. Esta configuración permite al modelo procesar más profundamente la información extraída en la primera capa. Capa de Salida: Tiene una sola neurona con una función de activación sigmoid, adecuada para predecir un valor continuo entre 0 y 1, (probabilidad.) Capas de Dropout: Se añaden después de cada capa densa (excepto la de salida) con una tasa de 0.5, lo que significa que el 50% de las neuronas se "apagan" aleatoriamente durante cada paso de entrenamiento. Esto ayuda a prevenir el sobreajuste asegurando que el modelo no dependa demasiado de cualquier característica particular.

Hiperparámetros del Modelo

Optimizador: Se usa adam, un optimizador basado en la gradiente descendente estocástica que ajusta la tasa de aprendizaje de manera adaptativa para diferentes parámetros, facilitando la convergencia más rápida y estable del entrenamiento.

Función de Pérdida: mean_squared_error (MSE) ya que es un problema de regresión.

EarlyStopping: Monitoriza val_loss y detiene el entrenamiento si no hay mejora después de patience=10 épocas, evitando así el sobreentrenamiento y ahorro de recursos.

ModelCheckpoint: Guarda el mejor modelo encontrado durante el entrenamiento, basado en val_loss, asegurando que se conserva el modelo con el mejor desempeño en los datos de validación.

✓ Punto 4 - Comparación y análisis de resultados

En la celda 4 comparen los resultados obtenidos de las diferentes redes y comenten las ventajas del mejor modelo y las desventajas del modelo con el menor desempeño.

```
# Celda 4
model_names = ['Modelo de una capa', 'Modelo Multicapa', 'Modelo Multicapa con Ea']
mae_values = [mae_one, mae_mul, mae_mul2]
mse_values = [mse_one, mse_mul, mse_mul2]

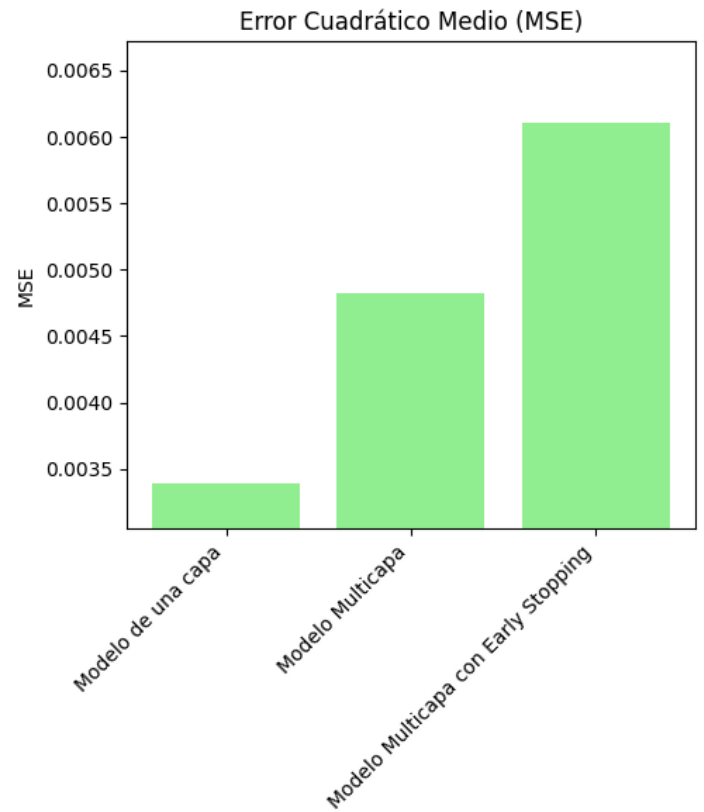
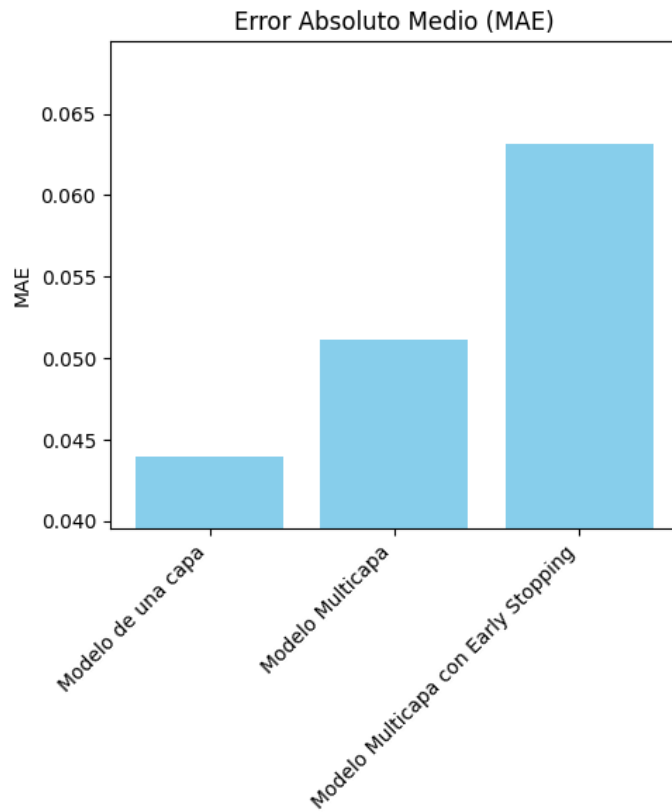
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
plt.bar(model_names, mae_values, color='skyblue')
plt.title('Error Absoluto Medio (MAE)')
plt.ylabel('MAE')
plt.ylim(min(mae_values) * 0.9, max(mae_values) * 1.1)
plt.xticks(rotation=45, ha='right')

plt.subplot(1, 2, 2)
plt.bar(model_names, mse_values, color='lightgreen')
plt.title('Error Cuadrático Medio (MSE)')
plt.ylabel('MSE')
plt.ylim(min(mse_values) * 0.9, max(mse_values) * 1.1)
plt.xticks(rotation=45, ha='right')
```



```
plt.tight_layout()  
plt.show()
```



El modelo de una sola capa es el más adecuado en cuanto a rendimiento y simplicidad, mientras que el modelo multicapa muestra signos de sobreajuste.

VENTAJAS DEL MEJOR MODELO (PUNTO 1: RED NEURONAL DE UNA CAPA):

- Tiene una pérdida muy baja en el conjunto de entrenamiento y en el de validación, lo que indica un buen ajuste del modelo a los datos.
 - El MAE y MSE son los más bajos entre los tres modelos.
 - Debido a que tiene una sola capa, resulta más simple y rápido de entrenar y ejecutar.
-

DESVENTAJAS DEL PEOR MODELO (PUNTO 3: RED NEURONAL MULTICAPA):

- Tiene niveles de MSE y MAE más altos, lo cual sugieren que es un modelo con predicciones menos precisas que los modelos 1 y 2.
- Muestra tener el peor rendimiento en comparación con los otros 2 modelos. Agregarle early stopping puede ayudar con temas de sobreajuste, sin embargo, vimos que esto puede introducir complejidad al modelo.
- El número de épocas puede variar ya que no es fijo.

Start coding or [generate](#) with AI.

