



LABORATORIO 3

Ejercicio

En este ejercicio, entrenará una red neuronal en el *Fuel Economy* dataset y luego explorará el efecto de la tasa de aprendizaje y el tamaño del lote en SGD.

Cuando esté listo, ejecute la siguiente celda para configurar todo.

```
# Setup plotting
import matplotlib.pyplot as plt
from learntools.deep_learning_intro.dltools import animate_sgd
plt.style.use('seaborn-whitegrid')
# Set Matplotlib defaults
plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsize='large',
       titleweight='bold', titlesize=18, titlepad=10)
plt.rc('animation', html='html5')

# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.deep_learning_intro.ex3 import *
```

Ejercicio

En el *Fuel Economy* dataset, su tarea consiste en predecir la economía de combustible de un automóvil determinadas características como el tipo de motor o el año en que se fabricó.

Primero cargue el conjunto de datos ejecutando la celda a continuación.



```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer, make_column_selector
from sklearn.model_selection import train_test_split

fuel = pd.read_csv('../input/dl-course-data/fuel.csv')

X = fuel.copy()
# Remove target
y = X.pop('FE')

preprocessor = make_column_transformer(
    (StandardScaler(),
     make_column_selector(dtype_include=np.number)),
    (OneHotEncoder(sparse=False),
     make_column_selector(dtype_include=object)),
)

X = preprocessor.fit_transform(X)
y = np.log(y) # log transform target instead of standardizing

input_shape = [X.shape[1]]
print("Input shape: {}".format(input_shape))
```

Input shape: [50]

Ejercicio

Eche un vistazo a los datos si lo desea. Nuestro objetivo en este caso es la columna '**FE**' y las columnas restantes son las características.



```
# Uncomment to see original data  
fuel.head()
```

	EngDispl	NumCyl	Transmission	FE	AirAspirationMethod	NumGears	TransLockup	TransCreeperGear	DriveDesc	IntakeValvePerCyl	ExhaustValvesPerCyl	CarlineClassDesc	VarValveTiming
0	4.7	8	AM6	28.0198	NaturallyAspirated	6	1	0	TwoWheelDriveRear	2	2	2Seaters	1
1	4.7	8	M6	25.6094	NaturallyAspirated	6	1	0	TwoWheelDriveRear	2	2	2Seaters	1
2	4.2	8	M6	26.8000	NaturallyAspirated	6	1	0	AllWheelDrive	2	2	2Seaters	1
3	4.2	8	AM6	25.0451	NaturallyAspirated	6	1	0	AllWheelDrive	2	2	2Seaters	1
4	5.2	10	AM6	24.8000	NaturallyAspirated	6	0	0	AllWheelDrive	2	2	2Seaters	1

Ejecute la siguiente celda para definir la red que usaremos para esta tarea.

[4]:

```
from tensorflow import keras  
from tensorflow.keras import layers  
  
model = keras.Sequential([  
    layers.Dense(128, activation='relu', input_shape=input_shape),  
    layers.Dense(128, activation='relu'),  
    layers.Dense(64, activation='relu'),  
    layers.Dense(1),  
)
```

Ejercicios

Paso1: Add Loss and Optimizer

Antes de entrenar la red, debemos definir la pérdida y el optimizador que usaremos. Usando el método de compilación del modelo, agregue el optimizador Adam y la pérdida MAE.



```
# YOUR CODE HERE
model.compile(
    optimizer='adam',
    loss='mae',
)

# Check your answer
q_1.check()
```

Correct

Ejercicios

Paso2: Train model

Una vez que haya definido el modelo y lo haya compilado con una pérdida y un optimizador, estará listo para el entrenamiento. Entrene la red durante 200 épocas con un tamaño de lote de 128. Los datos de entrada son **X** con destino **y**



```
# YOUR CODE HERE
history = model.fit(
    X, y,
    batch_size=128,
    epochs=200,
)

# Check your answer
q_2.check()
```

```
Epoch 1/200
9/9 [=====] - 1s 3ms/step - loss: 3.1927
Epoch 2/200
9/9 [=====] - 0s 2ms/step - loss: 1.1906
Epoch 3/200
9/9 [=====] - 0s 2ms/step - loss: 0.7161
Epoch 4/200
9/9 [=====] - 0s 2ms/step - loss: 0.4400
Epoch 5/200
9/9 [=====] - 0s 2ms/step - loss: 0.2911
Epoch 6/200
9/9 [=====] - 0s 2ms/step - loss: 0.2011
```

```
Epoch 187/200
9/9 [=====] - 0s 3ms/step - loss: 0.0268
Epoch 188/200
9/9 [=====] - 0s 4ms/step - loss: 0.0276
Epoch 189/200
9/9 [=====] - 0s 3ms/step - loss: 0.0274
Epoch 190/200
9/9 [=====] - 0s 3ms/step - loss: 0.0233
Epoch 191/200
9/9 [=====] - 0s 2ms/step - loss: 0.0262
Epoch 192/200
9/9 [=====] - 0s 3ms/step - loss: 0.0279
Epoch 193/200
9/9 [=====] - 0s 2ms/step - loss: 0.0289
Epoch 194/200
9/9 [=====] - 0s 2ms/step - loss: 0.0414
Epoch 195/200
9/9 [=====] - 0s 2ms/step - loss: 0.0366
Epoch 196/200
9/9 [=====] - 0s 3ms/step - loss: 0.0322
Epoch 197/200
9/9 [=====] - 0s 3ms/step - loss: 0.0265
Epoch 198/200
9/9 [=====] - 0s 3ms/step - loss: 0.0285
Epoch 199/200
9/9 [=====] - 0s 3ms/step - loss: 0.0280
Epoch 200/200
9/9 [=====] - 0s 3ms/step - loss: 0.0357
Epoch 201/200
9/9 [=====] - 0s 3ms/step - loss: 0.0373
Epoch 202/200
9/9 [=====] - 0s 3ms/step - loss: 0.0388
```

Correct

Ejercicios

Paso3: Evaluate training

Si entrenara el modelo por más tiempo, ¿esperaría que la pérdida disminuya aún más?

➤

```
# View the solution (Run this cell to receive credit!)  
q_3.check()
```

Esto depende de cómo haya evolucionado la pérdida durante el entrenamiento: si las curvas de aprendizaje se han estabilizado, normalmente no habrá ninguna ventaja para entrenar para épocas adicionales. Por el contrario, si la pérdida parece seguir disminuyendo, entonces entrenar durante más tiempo podría ser ventajoso.

Con la tasa de aprendizaje y el tamaño del lote, tiene cierto control sobre:

- Cuánto tiempo se tarda en entrenar a un modelo
- Qué ruidosas son las curvas de aprendizaje
- Qué pequeña se vuelve la pérdida

Para comprender mejor estos dos parámetros, veremos el modelo lineal, nuestra red neuronal más simple. Teniendo solo un peso y un sesgo, es más fácil ver qué efecto tiene un cambio de parámetro.

La siguiente celda generará una animación como la del tutorial. Cambie los valores de **learning_rate**, **batch_size** y **num_examples** (cuántos puntos de datos) y luego ejecute la celda. (Puede que tarde uno o dos minutos). Pruebe las siguientes combinaciones o pruebe algunas propias:

learning_rate	batch_size	num_examples
0.05	32	256
0.05	2	256
0.05	128	256
0.02	32	256
0.2	32	256
1.0	32	256
0.9	4096	8192
0.99	4096	8192

Ejercicios

Paso3: Evaluate training

```
# YOUR CODE HERE: Experiment with different values for the learning rate, batch size, and number of examples
learning_rate = 0.05
batch_size = 128
num_examples = 256

animate_sgd(
    learning_rate=learning_rate,
    batch_size=batch_size,
    num_examples=num_examples,
    # You can also change these, if you like
    steps=50, # total training steps (batches seen)
    true_w=3.0, # the slope of the data
    true_b=2.0, # the bias of the data
)
```



```
# YOUR CODE HERE: Experiment with different values for the learning rate, batch size, and number of examples
learning_rate = 0.9
batch_size = 4096
num_examples = 8192

animate_sgd(
    learning_rate=learning_rate,
    batch_size=batch_size,
    num_examples=num_examples,
    # You can also change these, if you like
    steps=50, # total training steps (batches seen)
    true_w=3.0, # the slope of the data
    true_b=2.0, # the bias of the data
)
```



Ejercicios

Paso4: Learning Rate and Batch Size

¿Qué efecto tuvo el cambio de estos parámetros? Una vez que lo haya pensado, ejecute la celda a continuación para discutir un poco.



```
# View the solution (Run this cell to receive credit!)  
q_4.check()
```

Correct:

Probablemente haya visto que los tamaños de lote más pequeños daban actualizaciones de peso más ruidosas y curvas de pérdida. Esto se debe a que cada lote es una pequeña muestra de datos y las muestras más pequeñas tienden a dar estimaciones más ruidosas. Sin embargo, los lotes más pequeños pueden tener un efecto de "promediado" que puede ser beneficioso.

Las tasas de aprendizaje más pequeñas hacen que las actualizaciones sean más pequeñas y la formación tarda más en converger. Las tasas de aprendizaje elevadas pueden acelerar la capacitación, pero no "se adaptan" al mínimo también. Cuando la tasa de aprendizaje es demasiado grande, la formación puede fallar por completo. (Intente establecer la tasa de aprendizaje en un valor grande como 0,99 para ver esto).

A photograph of a workspace featuring a silver laptop on the left, a wooden desk, and an open notebook with a black pen resting on it. A semi-transparent green rectangle is centered over the image, containing the text 'LABORATORIO 4' in white, bold, sans-serif capital letters.

LABORATORIO 4

Ejercicios

En este ejercicio, aprenderá a mejorar los resultados del entrenamiento al incluir **Early Stopping Callback** para evitar el **overfitting** (sobreajuste).

Cuando esté listo, ejecute la siguiente celda para configurar todo.

```
[1]: # Setup plotting
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
# Set Matplotlib defaults
plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsize='large',
       titleweight='bold', titlesize=18, titlepad=10)
plt.rc('animation', html='html5')

# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.deep_learning_intro.ex4 import *
```

Ejercicios

Primero cargue el conjunto de datos de Spotify. Su tarea será predecir la popularidad de una canción en función de varias características de audio, como "tempo", "danceability" y "mode".



```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import GroupShuffleSplit
```

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks

spotify = pd.read_csv('../input/dl-course-data/spotify.csv')

X = spotify.copy().dropna()
y = X.pop('track_popularity')
artists = X['track_artist']

features_num = ['danceability', 'energy', 'key', 'loudness', 'mode',
                'speechiness', 'acousticness', 'instrumentalness',
                'liveness', 'valence', 'tempo', 'duration_ms']
features_cat = ['playlist_genre']

preprocessor = make_column_transformer(
    (StandardScaler(), features_num),
    (OneHotEncoder(), features_cat),
)

# We'll do a "grouped" split to keep all of an artist's songs in one
# split or the other. This is to help prevent signal leakage.
def group_split(X, y, group, train_size=0.75):
    splitter = GroupShuffleSplit(train_size=train_size)
    train, test = next(splitter.split(X, y, groups=group))
    return (X.iloc[train], X.iloc[test], y.iloc[train], y.iloc[test])

X_train, X_valid, y_train, y_valid = group_split(X, y, artists)

X_train = preprocessor.fit_transform(X_train)
X_valid = preprocessor.transform(X_valid)
y_train = y_train / 100 # popularity is on a scale 0-100, so this rescales to 0-1.
y_valid = y_valid / 100

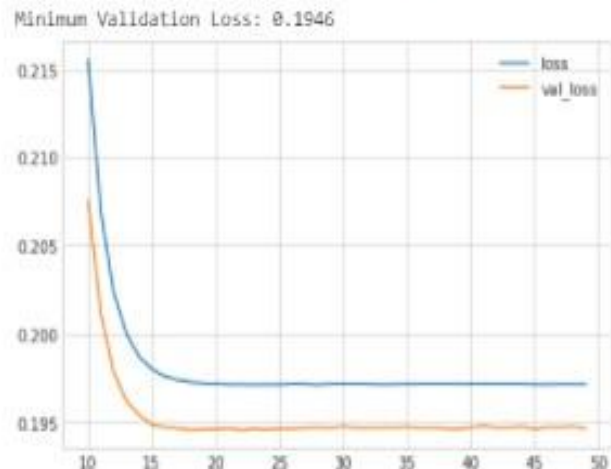
input_shape = [X_train.shape[1]]
print("Input shape: {}".format(input_shape))
```

Input shape: [18]

Ejercicios

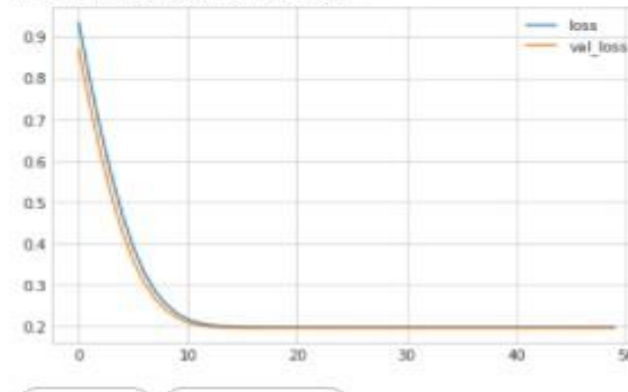
Comencemos con la red más simple, un modelo lineal. Este modelo tiene poca capacidad. Ejecute esta siguiente celda sin ningún cambio para entrenar un modelo lineal en el conjunto de datos de Spotify.

```
# Start the plot at epoch 10
history_df.loc[10:, ['loss', 'val_loss']].plot()
print("Minimum Validation Loss: {:.4f}".format(history_df['val_loss'].min()));
```



```
model = keras.Sequential([
    layers.Dense(1, input_shape=input_shape),
])
model.compile(
    optimizer='adam',
    loss='mae',
)
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=512,
    epochs=50,
    verbose=0, # suppress output since we'll plot the curves
)
history_df = pd.DataFrame(history.history)
history_df.loc[0:, ['loss', 'val_loss']].plot()
print("Minimum Validation Loss: {:.4f}".format(history_df['val_loss'].min()));
```

Minimum Validation Loss: 0.1946



No es raro que las curvas sigan un patrón de "palo de hockey" como se ve aquí. Esto hace que la parte final del entrenamiento sea difícil de ver, así que comencemos en la época 10:

Ejercicios

Paso1: Evaluate Baseline

¿Qué piensas? ¿Diría que este modelo está **underfitting** (desajustado), **overfitting** (sobreajustado), perfecto?

```
# View the solution (Run this cell to receive credit!)  
q_1.check()
```

Correct:

La brecha entre estas curvas es bastante pequeña y la pérdida de validación nunca aumenta, por lo que es más probable que la red esté **underfitting** que **overfitting**. Valdría la pena experimentar con más capacidad para ver si ese es el caso.

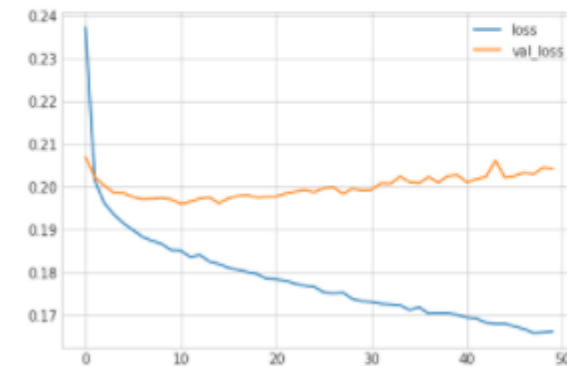
Ejercicios

Paso1: Evaluate Baseline

Ahora agreguemos algo de capacidad a nuestra red. Agregaremos tres capas ocultas con 128 unidades cada una. Ejecute la siguiente celda para entrenar la red y ver las curvas de aprendizaje.

```
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=input_shape),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
model.compile(
    optimizer='adam',
    loss='mae',
)
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=512,
    epochs=50,
)
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
print("Minimum Validation Loss: {:.4f}".format(history_df['val_loss'].min()));
```

```
Epoch 48/50: 0s 4ms/step - loss: 0.1665 - val_loss: 0.2025
Epoch 47/50: 0s 4ms/step - loss: 0.1655 - val_loss: 0.2033
Epoch 48/50: 0s 4ms/step - loss: 0.1649 - val_loss: 0.2029
Epoch 49/50: 0s 3ms/step - loss: 0.1657 - val_loss: 0.2045
Epoch 50/50: 0s 4ms/step - loss: 0.1649 - val_loss: 0.2042
Minimum Validation Loss: 0.1968
```



Ejercicios

Paso2: Add Capacity

¿Cuál es su evaluación de estas curvas? ¿**underfitting** (Mal ajuste), **overfitting** (ajuste excesivo), correcto?

▶

```
# View the solution (Run this cell to receive credit!)  
q_2.check()
```

Correct:

Ahora, la pérdida de validación comienza a aumentar muy pronto, mientras que la pérdida de entrenamiento continúa disminuyendo. Esto indica que la red ha comenzado a adaptarse. En este punto, tendríamos que intentar algo para prevenirlo, ya sea reduciendo el número de unidades o mediante un método como la detención anticipada. (¡Veremos otro en la próxima lección!)

Ejercicios

Paso3: Define Early Stopping Callback

Ahora defina **Early Stopping Callback** que espera 5 épocas ('patients') para un cambio en la pérdida de validación de al menos **0,001 (min_delta)** y mantiene los pesos con la mejor pérdida (**restore_best_weights**).



```
from tensorflow.keras import callbacks

# YOUR CODE HERE: define an early stopping callback
early_stopping = callbacks.EarlyStopping (
    min_delta=0.001,
    patience=5,
    restore_best_weights=True,
)

# Check your answer
q_3.check()
```

Correct

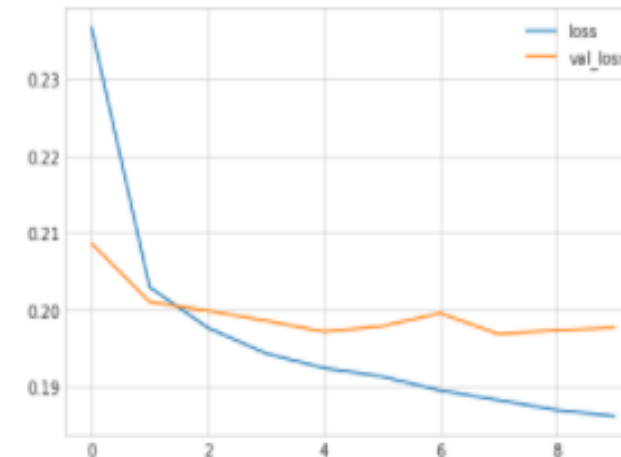
Ejercicios

Paso3: Define Early Stopping Callback

Ahora ejecute esta celda para entrenar el modelo y obtener las curvas de aprendizaje. Observe el argumento de **callbacks** (devoluciones de llamada) en **model.fit**.

```
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=input_shape),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
model.compile(
    optimizer='adam',
    loss='mae',
)
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=512,
    epochs=50,
    callbacks=[early_stopping]
)
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
print("Minimum Validation Loss: {:.4f}".format(history_df['val_loss'].min()));
```

```
Epoch 7/50
48/48 [=====] - 0s 4ms/step - loss: 0.1902 - val_loss: 0.1995
Epoch 8/50
48/48 [=====] - 0s 4ms/step - loss: 0.1867 - val_loss: 0.1968
Epoch 9/50
48/48 [=====] - 0s 4ms/step - loss: 0.1876 - val_loss: 0.1973
Epoch 10/50
48/48 [=====] - 0s 4ms/step - loss: 0.1839 - val_loss: 0.1977
Minimum Validation Loss: 0.1968
```



Ejercicios

Paso4: Train and Interpret

¿Fue esto una mejora en comparación con el entrenamiento sin detenerse temprano?

Was this an improvement compared to training without early stopping?

+ Code

+ Markdown

```
# View the solution (Run this cell to receive credit!)
q_4.check()
```

Correct:

Callback (devolución de llamada) de **Early Stopping** (detención temprana) detuvo el entrenamiento una vez que la red comenzó a sobreajustarse. Además, al incluir **restore_best_weights** todavía podemos mantener el modelo donde la pérdida de validación fue más baja.

A photograph of a workspace featuring a silver laptop on the left, a wooden desk, and an open notebook with a black pen resting on it. A semi-transparent green rectangle is centered over the image, containing the text 'LABORATORIO 5' in white, bold, sans-serif capital letters.

LABORATORIO 5

Ejercicios

En este ejercicio, agregará la omisión al modelo de Spotify del Ejercicio 4 y verá cómo la normalización por lotes puede permitirle entrenar modelos con éxito en conjuntos de datos difíciles. ¡Ejecute la siguiente celda para comenzar!

Primero cargue el conjunto de datos de Spotify.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import GroupShuffleSplit

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks

spotify = pd.read_csv('../input/dl-course-data/spotify.csv')

X = spotify.copy().dropna()
y = X.pop('track_popularity')
artists = X['track_artist']

features_num = ['danceability', 'energy', 'key', 'loudness', 'mode',
               'speechiness', 'acousticness', 'instrumentalness',
               'liveness', 'valence', 'tempo', 'duration_ms']
features_cat = ['playlist_genre']

preprocessor = make_column_transformer(
    (StandardScaler(), features_num),
    (OneHotEncoder(), features_cat),
)
```

```
# Setup plotting
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
# Set Matplotlib defaults
plt.rc('figure', autolayout=True)
plt.rc('axes', labelweight='bold', labelsizelarge',
       titleweight='bold', titlesize=18, titlepad=10)
plt.rc('animation', html='html5')

# Setup feedback system
from learntools.core import binder
binder.bind(globals())
from learntools.deep_learning_intro.ex5 import *
```

```
def group_split(X, y, group, train_size=0.75):
    splitter = GroupShuffleSplit(train_size=train_size)
    train, test = next(splitter.split(X, y, groups=group))
    return (X.iloc[train], X.iloc[test], y.iloc[train], y.iloc[test])

X_train, X_valid, y_train, y_valid = group_split(X, y, artists)

X_train = preprocessor.fit_transform(X_train)
X_valid = preprocessor.transform(X_valid)
y_train = y_train / 100
y_valid = y_valid / 100

input_shape = [X_train.shape[1]]
print("Input shape: {}".format(input_shape))
```

Input shape: [18]

Ejercicios

Paso1: Add Dropout to Spotify Model

Aquí está el último modelo del ejercicio 4. Agregue dos **dropout layers** (capas de abandono u omisión), una después de la capa Densa con 128 unidades y otra después de la capa Densa con 64 unidades. Establezca la tasa de abandono en ambos en 0,3.



```
# YOUR CODE HERE: Add two 30% dropout layers, one after 128 and one after 64
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=input_shape),
    layers.Dropout(rate=0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(rate=0.3),
    layers.Dense(1)
])

# Check your answer
q_1.check()
```

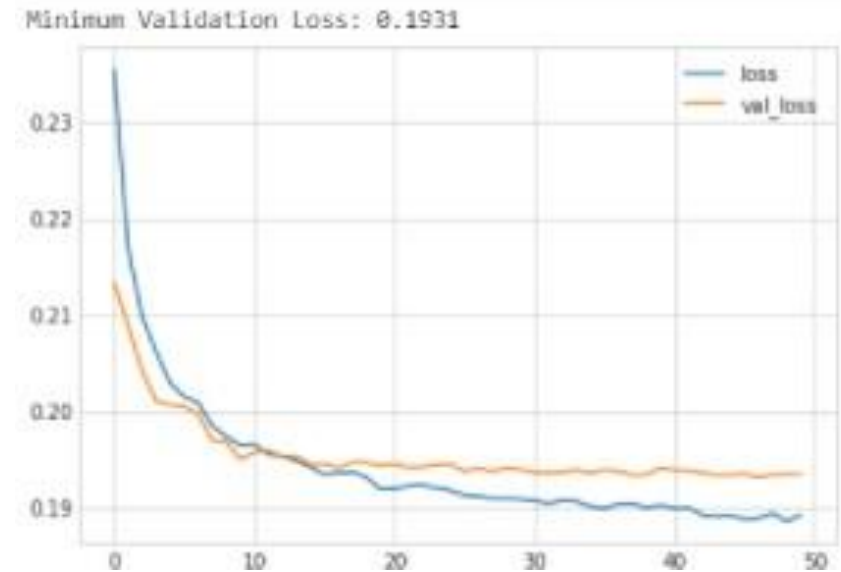
Correct

Ejercicios

Paso1: Add Dropout to Spotify Model

Ahora ejecute la siguiente celda para entrenar el modelo y vea el efecto de agregar abandonos.

```
model.compile(
    optimizer='adam',
    loss='mae',
)
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=512,
    epochs=50,
    verbose=0,
)
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
print("Minimum Validation Loss: {:.4f}".format(history_df['val_loss'].min()))
```



Ejercicios

Paso2: Evaluate Dropout

Recuerde del ejercicio 4 que este modelo tendía a sobreajustar los datos alrededor de la época 5. ¿La adición de abandonos pareció ayudar a prevenir el sobreajuste esta vez?

▶ `# View the solution (Run this cell to receive credit!)`
`q_2.check()`

Correct

A partir de las curvas de aprendizaje, puede ver que la pérdida de validación permanece cerca de un mínimo constante aunque la pérdida de entrenamiento continúa disminuyendo. Entonces, podemos ver que agregar abandonos evitó el sobreajuste esta vez. Además, al dificultar que la red se ajuste a patrones espurios, la deserción puede haber alentado a la red a buscar más patrones verdaderos, posiblemente mejorando también la pérdida de validación).

Ahora, cambiaremos de tema para explorar cómo la normalización por lotes puede solucionar problemas en el entrenamiento.

Cargue el conjunto de datos Concrete. Esta vez no haremos ninguna estandarización. Esto hará que el efecto de la normalización de lotes sea mucho más evidente.

▶

```
import pandas as pd

concrete = pd.read_csv('../input/dl-course-data/concrete.csv')
df = concrete.copy()

df_train = df.sample(frac=0.7, random_state=0)
df_valid = df.drop(df_train.index)

X_train = df_train.drop('CompressiveStrength', axis=1)
X_valid = df_valid.drop('CompressiveStrength', axis=1)
y_train = df_train['CompressiveStrength']
y_valid = df_valid['CompressiveStrength']

input_shape = [X_train.shape[1]]
```

Ejercicios

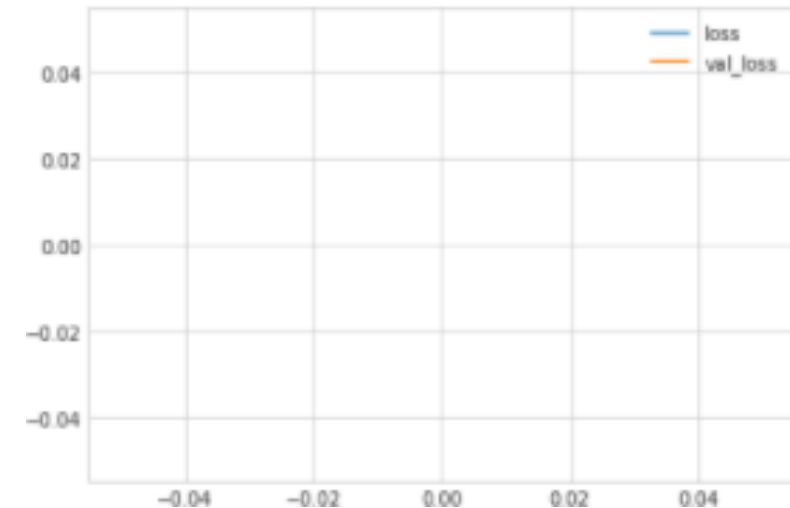
Paso2: Evaluate Dropout

Ejecute la siguiente celda para entrenar a la red con los datos de hormigón no estandarizados.

```
11]: model = keras.Sequential([
    layers.Dense(512, activation='relu', input_shape=input_shape),
    layers.Dense(512, activation='relu'),
    layers.Dense(512, activation='relu'),
    layers.Dense(1),
])
model.compile(
    optimizer='sgd', # SGD is more sensitive to differences of scale
    loss='mae',
    metrics=['mae'],
)
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=64,
    epochs=100,
    verbose=0,
)

history_df = pd.DataFrame(history.history)
history_df.loc[0:, ['loss', 'val_loss']].plot()
print(("Minimum Validation Loss: {:.4f}").format(history_df['val_loss'].min()))
```

Minimum Validation Loss: nan



¿Terminaste con un gráfico en blanco? Intentar entrenar esta red en este conjunto de datos generalmente fallará. Incluso cuando converge (debido a una inicialización de peso afortunada), tiende a converger a un número muy grande.

Ejercicios

Paso3: Add batch Normalization Layers

La normalización por lotes puede ayudar a corregir problemas como este.

Agregue cuatro capas de BatchNormalization, una antes de cada una de las capas densas. (Recuerde mover el argumento `input_shape` a la nueva primera capa).



```
# YOUR CODE HERE: Add a BatchNormalization layer before each Dense layer
model = keras.Sequential([
    layers.BatchNormalization(input_shape=input_shape),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(1),
])

# Check your answer
q_3.check()
```

Correct

Ejercicios

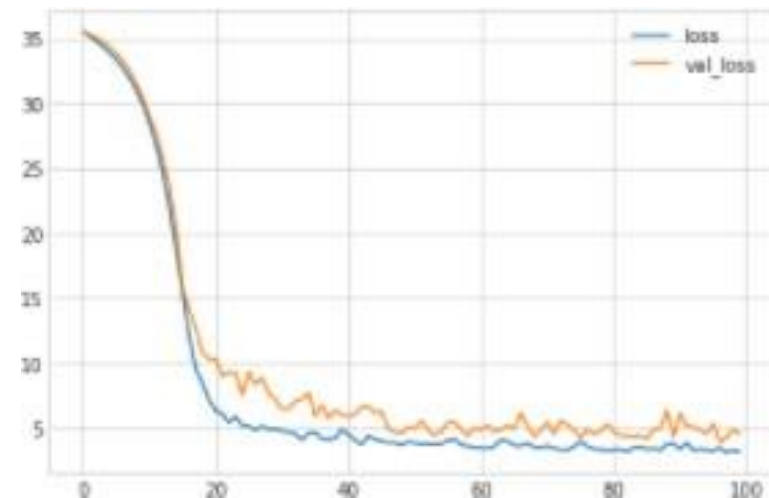
Paso3: Add batch Normalization Layers

Ejecute la siguiente celda para ver si la normalización por lotes nos permitirá entrenar el modelo.

```
model.compile(
    optimizer='sgd',
    loss='mae',
    metrics=['mae'],
)
EPOCHS = 100
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=64,
    epochs=EPOCHS,
    verbose=0,
)

history_df = pd.DataFrame(history.history)
history_df.loc[0:, ['loss', 'val_loss']].plot()
print(("Minimum Validation Loss: {:.4f}").format(history_df['val_loss'].min()))
```

Minimum Validation Loss: 3.8706



Ejercicios

Paso4: Evaluate batch Normalization

¿Le ayudó agregar la normalización por lotes?



```
# View the solution (Run this cell to receive credit!)  
q_4.check()
```

Correct

¡Puede ver que agregar la normalización por lotes fue una gran mejora en el primer intento! Al escalar de forma adaptativa los datos a medida que pasan a través de la red, la normalización por lotes puede permitirle entrenar modelos en conjuntos de datos difíciles.