This notebook is an exercise in the Intro to Deep Learning course. You can reference the tutorial at this link.

Introduction

In this exercise, you'll add dropout to the *Spotify* model from Exercise 4 and see how batch normalization can let you successfully train models on difficult datasets.

Run the next cell to get started!

In [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.ex5 import *
```

First load the *Spotify* dataset.

In [2]:
```
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),
)

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]
```

1) Add Dropout to Spotify Model

Here is the last model from Exercise 4. Add two dropout layers, one after the `Dense` layer with 128 units, and one after the `Dense` layer with 64 units. Set the dropout rate on both to `0.3`.

In [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()
```

```
2022-12-18 05:40:31.343389: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:31.434552: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:31.435439: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:31.437708: I tensorflow/core/platform/cpu_feature_guard.cc:142] This Te
nsorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler fla
gs.
2022-12-18 05:40:31.438073: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:31.439020: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:31.439990: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:33.490851: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
```

```
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:33.491762: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:33.492508: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937]
successful NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero
2022-12-18 05:40:33.493100: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Cre
ated device /job:localhost/replica:0/task:0/device:GPU:0 with 15401 MB memory:  -> devic
e: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
```

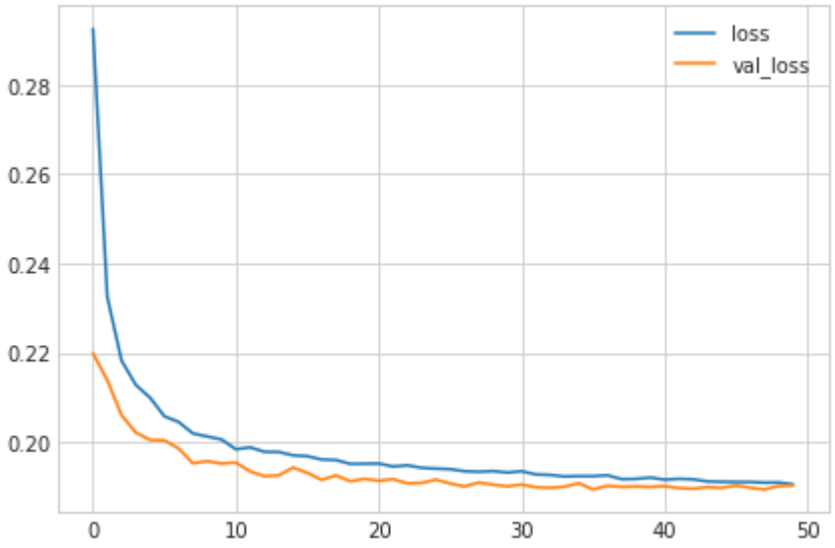
Correct

In [4]:
```
# Lines below will give you a hint or solution code
#q_1.hint()
#q_1.solution()
```

Now run this next cell to train the model see the effect of adding dropout.

In [5]:
```
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: {:0.4f}".format(history_df['val_loss'].min()))
```

```
2022-12-18 05:40:34.020769: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:1
85] None of the MLIR Optimization Passes are enabled (registered 2)
Minimum Validation Loss: 0.1894
```



2) Evaluate Dropout

Recall from Exercise 4 that this model tended to overfit the data around epoch 5. Did adding dropout seem to help prevent overfitting this time?

In [6]:
```
# View the solution (Run this cell to receive credit!)
q_2.check()
```

Correct:

From the learning curves, you can see that the validation loss remains near a constant minimum even though the training loss continues to decrease. So we can see that adding dropout did prevent overfitting this time. Moreover, by making it harder for the network to fit spurious patterns, dropout may have encouraged the network to seek out more of the true patterns, possibly improving the validation loss some as well).

Now, we'll switch topics to explore how batch normalization can fix problems in training.

Load the *Concrete* dataset. We won't do any standardization this time. This will make the effect of batch normalization much more apparent.

In [7]:
```
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]]
```

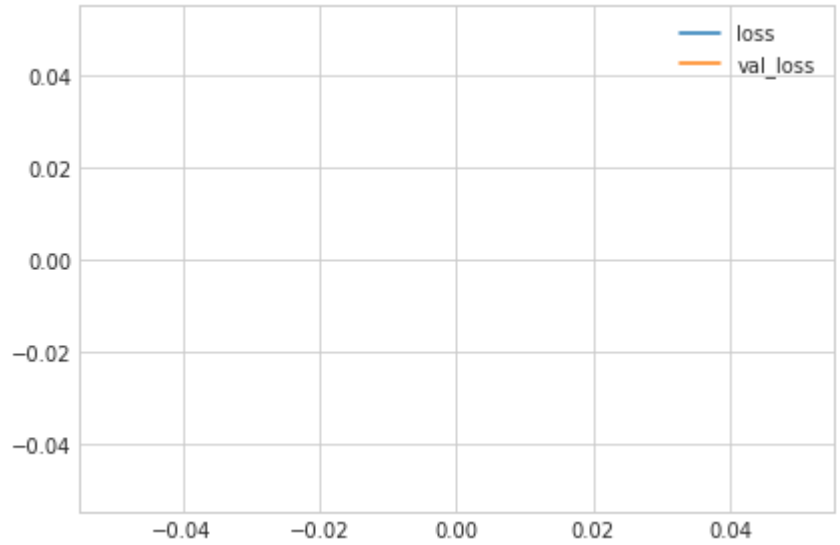
Run the following cell to train the network on the unstandardized *Concrete* data.

In [8]:
```
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: {:0.4f}").format(history_df['val_loss'].min()))
```

```
Minimum Validation Loss: nan
```



Did you end up with a blank graph? Trying to train this network on this dataset will usually fail. Even when it does converge (due to a lucky weight initialization), it tends to converge to a very large number.

3) Add Batch Normalization Layers

Batch normalization can help correct problems like this.

Add four `BatchNormalization` layers, one before each of the dense layers. (Remember to move the `input_shape` argument to the new first layer.)

In [9]:
```
# YOUR CODE HERE: Add a BatchNormalization layer before each Dense layer
model = keras.Sequential([
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu', input_shape=input_shape),
    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

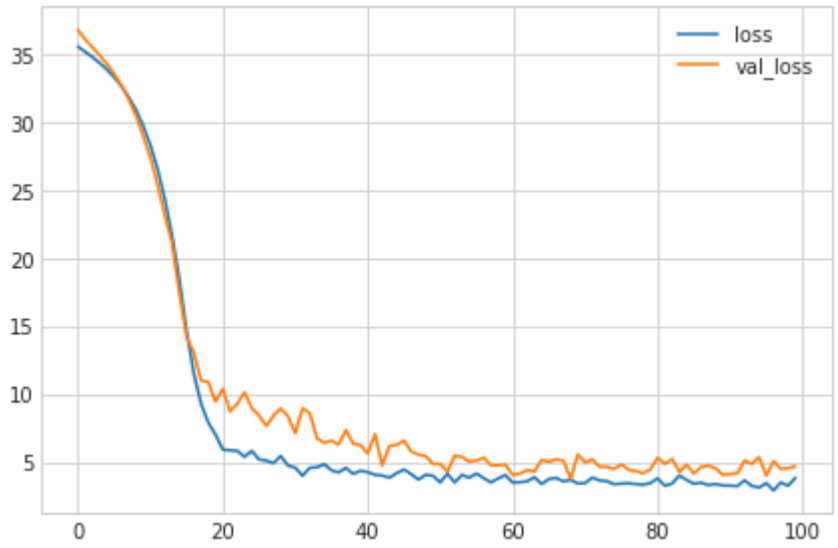
In [10]:
```
# Lines below will give you a hint or solution code
#q_3.hint()
#q_3.solution()
```

Run the next cell to see if batch normalization will let us train the model.

In [11]:
```
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: {:0.4f}").format(history_df['val_loss'].min()))
```

```
Minimum Validation Loss: 3.7695
```



4) Evaluate Batch Normalization

Did adding batch normalization help?

In [12]:
```
# View the solution (Run this cell to receive credit!)
q_4.check()
```

Correct:

You can see that adding batch normalization was a big improvement on the first attempt! By adaptively scaling the data as it passes through the network, batch normalization can let you train models on difficult datasets.

Keep Going

Create neural networks for binary classification.

Have questions or comments? Visit the course discussion forum to chat with other learners.