Dropout and Batch Normalization

向練習 4 中的 Spotify 模型新增 Dropout，並了解批次歸一化如何幫助您在複雜的資料集上成功訓練模型。

設定環境

```
# 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 *
```

```
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） 將 Dropout 加入 Spotify 模型

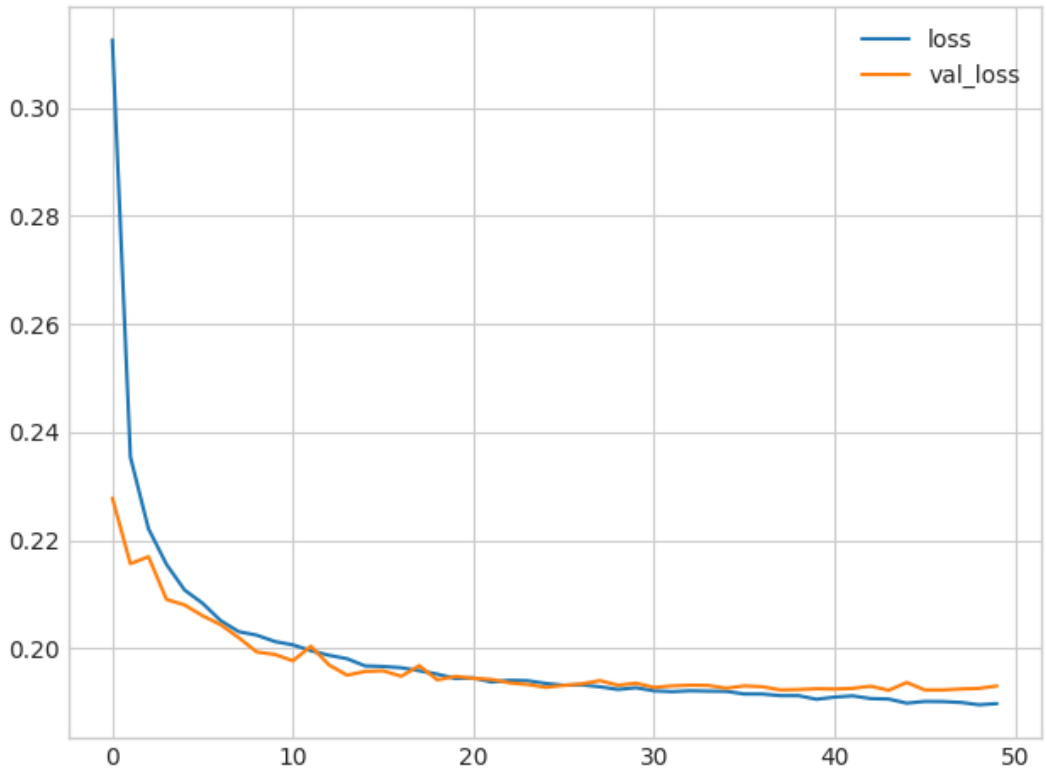
這是練習 4 的最後一個模型。增加兩個 dropout 層，一個位於 Dense 層之後，包含 128 個單元，另一個位於 Dense 層之後，包含 64 個單元。將兩個 dropout 率均設定為 0.3。

```
# 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(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(1)
])

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

訓練模型，查看新增 dropout 的效果。

```
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()))
```

```
Minimum Validation Loss: 0.1923
```



2) 評估 Dropout

回想一下練習 4，這個模型傾向於在第 5 個時期(epoch)左右過度擬合資料。這次添加 dropout 似乎有助於防止過度擬合嗎？

從學習曲線可以看出，即使訓練損失持續下降，驗證損失仍然接近恆定的最小值。因此，我們可以看出，這次添加 dropout 確實防止了過擬合。此外，透過增加網路擬合虛假模式的難度，dropout 可能鼓勵網路尋找更多真實模式，這可能也在一定程度上改善了驗證損失。

探討批量歸一化(batch normalization)如何解決訓練中的問題。

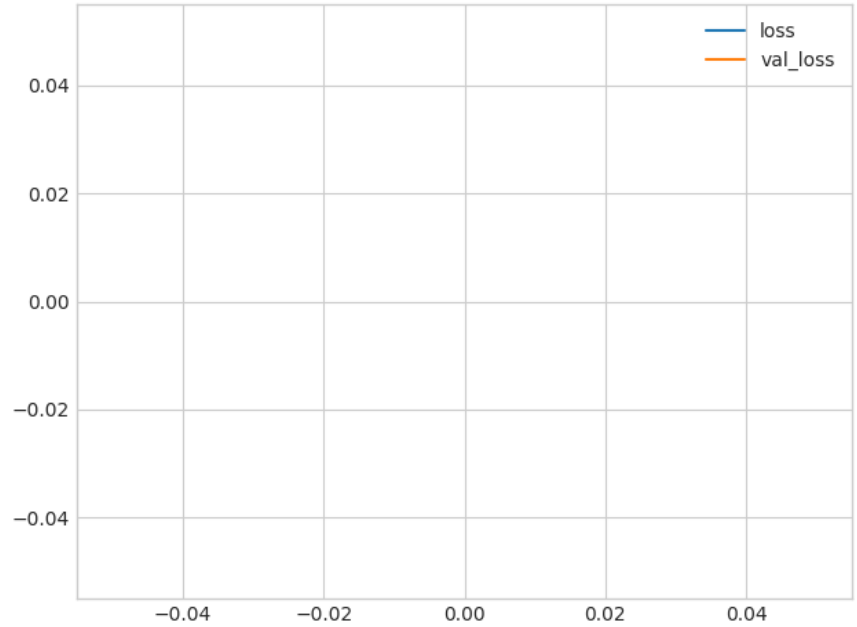
載入 Concrete 資料集。這次不進行任何標準化。這將使批量歸一化的效果更加明顯。

```
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]]
```

在非標準化的(unstandardized)具體資料上訓練網路。

```
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
```



最終得到的是一張空白圖嗎？嘗試在這個資料集上訓練這個網路通常會失敗。即使它收斂了（由於權重初始化的幸運），它也傾向於收斂到一個非常大的數字。

3) Add Batch Normalization Layers(添加批量標準化層)

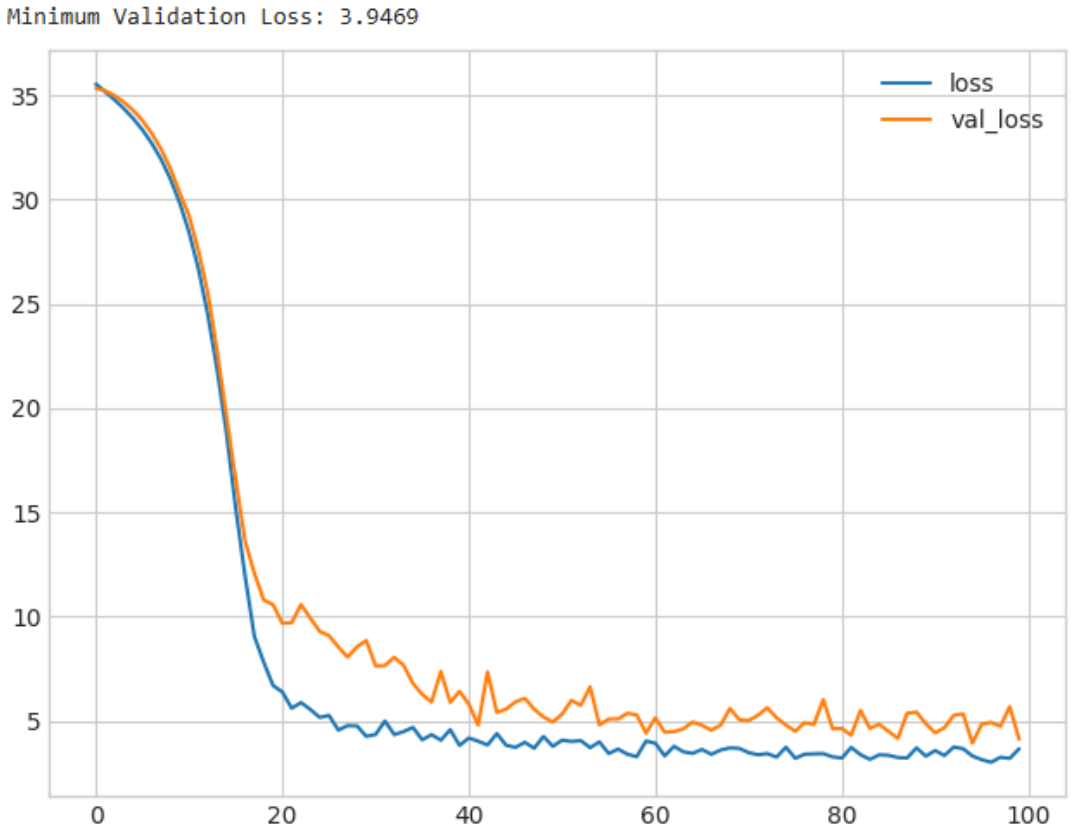
批量歸一化可以幫助糾正此類問題。

加入四個 BatchNormalization layers，分別位於每個密集層之前。（記得將 input_shape 參數移到新的第一層。）

```
# 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()
```

查看批量標準化是否可以讓我們訓練模型。

```
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()))
```



4）評估批量標準化(Batch Normalization)

可以看到，添加 batch normalization 後，首次嘗試就取得了巨大的進步！透過在資料通過網路時自適應地縮放數據，批量歸一化可以讓你在複雜的資料集上訓練模型。