Stochastic Gradient Descent

在本練習中，您將基於 ***Fuel Economy*** 資料集訓練一個神經網絡，然後探索學習率和批次大小對隨機梯度下降 (SGD) 的影響。

設定環境

```
# 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 *
```

在燃油經濟性資料集中，您的任務是根據汽車的引擎類型或生產年份等特徵預測其燃油經濟性。

First,載入資料集

```
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]
```

我們的目標是「FE」列，其餘列是 features。

```
# Uncomment to see original data
fuel.head()
# Uncomment to see processed features
pd.DataFrame(X[:10,:]).head()
```

[3]:

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	44	45	46	47	48	49
0	0.913643	1.068005	0.524148	0.685653	-0.226455	0.391659	0.43492	0.463841	-0.447941	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.913643	1.068005	0.524148	0.685653	-0.226455	0.391659	0.43492	0.463841	-0.447941	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.530594	1.068005	0.524148	0.685653	-0.226455	0.391659	0.43492	0.463841	-0.447941	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.530594	1.068005	0.524148	0.685653	-0.226455	0.391659	0.43492	0.463841	-0.447941	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.296693	2.120794	0.524148	-1.458464	-0.226455	0.391659	0.43492	0.463841	-0.447941	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 50 columns

Define the network we'll use for this task.

```
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),
])
```

1) Add Loss and Optimizer(添加損失和優化器)

在訓練網路之前，我們需要定義要使用的損失函數和最佳化器。使用模型的編譯方法，加入 Adam 最佳化器和 MAE 損失函數(Adam optimizer and MAE loss)。

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

2) Train Model

Once you've defined the model and compiled it with a loss and optimizer you're ready for training. Train the network for 200 epochs with a batch size of 128. The input data is X with target y.

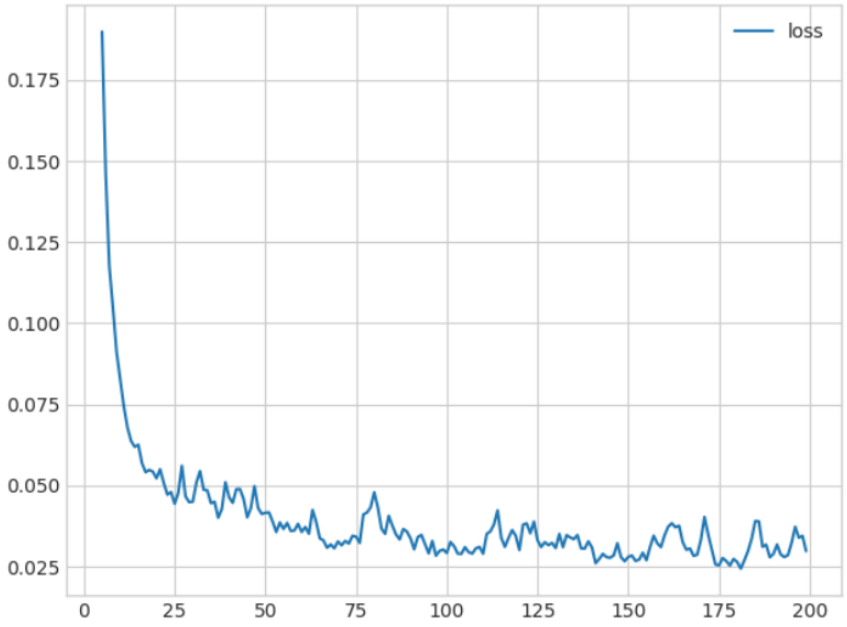
```
history = model.fit(
    X,y, # training data
    batch_size=128, # batch size
    epochs=200 # epochs
)
```

```
Epoch 185/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0337
Epoch 186/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0391
Epoch 187/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0390
Epoch 188/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0312
Epoch 189/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0318
Epoch 190/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0279
Epoch 191/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0290
Epoch 192/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0319
Epoch 193/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0288
Epoch 194/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0280
Epoch 195/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0286
Epoch 196/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0324
Epoch 197/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0373
Epoch 198/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0340
Epoch 199/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0345
Epoch 200/200
9/9 [==============================] - 0s 3ms/step - loss: 0.0298
```

最後一步是查看損失曲線並評估訓練效果。獲取訓練損失的圖表。

```python
import pandas as pd

history_df = pd.DataFrame(history.history)
# Start the plot at epoch 5. You can change this to get a different view.
history_df.loc[5:, ['loss']].plot();
```



3) 評估訓練效果

這取決於訓練過程中損失函數的演變：如果學習曲線已經趨於平穩，那麼繼續訓練通常不會帶來任何優勢。相反，如果損失函數似乎仍在下降，那麼繼續訓練更長時間可能會帶來優勢。

透過設定學習率和批次大小，你可以控制以下因素：
1.訓練模型需要多長時間
2.學習曲線的噪音程度
3.損失函數的數值會變得多小

為了更好地理解這兩個參數，我們將研究線性模型，也就是最簡單的神經網路。由於它只有一個權重和一個偏差，因此更容易看出參數變化的影響。

下一個單元將產生一個類似教學課程中的動畫。變更 learning_rate、batch_size 和 num_examples（資料點數量）的值，然後執行該單元。（這可能需要一兩分鐘。）嘗試以下組合，或嘗試一些您自己的組合：

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

```
# Experiment with different values for the learning rate, batch size, and number of examples
learning_rate = 0.05
batch_size = 32
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
)
```

影片

4）Learning Rate and Batch Size (學習率和批次大小)

改變這些參數會產生什麼影響？

較小的批次大小會導致權重更新和損失曲線雜訊較大。這是因為每個批次都是一個較小的資料樣本，而較小的樣本往往會產生雜訊較大的估計值。較小的批次可以產生“平均”效應，這可能是有益的。

較小的學習率會使更新量較小，訓練需要更長時間才能收斂。較大的學習率可以加快訓練速度，但也無法「穩定」到最小值。當學習率過大時，訓練可能會完全失敗。（嘗試將學習率設定為較大的值，例如 0.99，以查看此效果。）