Introduction to TensorFlow

Linear Regression with TensorFlow

郭耀仁

大綱

- 取得資料
- Benchmark
- 建構 TensorFlow 計算圖形
- 訓練
- 檢視 TensorBoard
- 加入 Name Scopes
- 隨堂練習

取得資料

簡單、作為測試目的即可

Scikit-Learn Boston 房價資料集

In [1]:
```
from sklearn.datasets import load_boston

boston = load_boston()
print(boston.feature_names)
print(boston.DESCR)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (at
tribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 s
q.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by t
own
```

- LSTAT   % lower status of the population
- MEDV   Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

This dataset was taken from the StatLib library which is maintained at Carnegi
e Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential D
ata and Sources of Collinearity', Wiley, 1980. 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.

In [2]:

```
X_arr = boston.data[:, -1].reshape(-1, 1)
y_arr = boston.target
print(X_arr.shape)
print(y_arr.shape)
```

```
(506, 1)
(506,)
```

In [3]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size=0.3, r
andom_state=123)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(354, 1)
(152, 1)
(354,)
(152,)
```

Benchmark

以 Scikit-Learn 的 LinearRegression 作對照

In [4]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
```

In [5]:

```
print(lr.coef_)
print(lr.intercept_)
print(mse)
```

```
[-0.95406451]
34.776016063907036
38.68755662089717
```

Benchmark 完整程式碼

In [6]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

boston = load_boston()
X_arr = boston.data[:, -1].reshape(-1, 1)
y_arr = boston.target
X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size=0.3, r
andom_state=123)
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
```

建構 TensorFlow 計算圖形

準備 Placeholders 供訓練時輸入 X_train、y_train

In [7]:
```
import tensorflow as tf

X_train_shape = X_train.shape
y_train_shape = y_train.shape
X = tf.placeholder(tf.float32, X_train_shape)
y = tf.placeholder(tf.float32, y_train_shape)
```

準備變數供訓練時尋找最適係數（Weights）與殘差項（Bias）

In [8]:
```
W_shape = (X_train_shape[1], 1)
b_shape = (1,)
W = tf.Variable(tf.random_normal(W_shape))
b = tf.Variable(tf.random_normal(b_shape))
with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  print(sess.run(W))
  print(sess.run(b))
```

```
[[1.4759609]]
[-1.2539263]
```

檢查 X、W 與 b 的外觀，寫下 y_pred 的公式

In [9]:
```
print(X.shape)
print(W.shape)
print(b.shape)
y_pred = tf.matmul(X, W) + b
```

```
(354, 1)
(1, 1)
(1,)
```

(354, 1) 與 (1,) 可以相加運算嗎?

In [10]:
```
a = tf.ones((354, 1), dtype=tf.int32)
b = tf.constant(2, shape=(1,))
c = a + b
#with tf.Session() as sess:
  #print(sess.run(c))
```

寫下成本函數的公式

https://scikit-learn.org/stable/modules/linear_model.html (https://scikit-learn.org/stable/modules/linear_model.html)

In [11]:
```
loss = tf.reduce_sum((y - y_pred)**2)
```

tf.reduce_sum() 做了什麼事?

In [12]:
```
a = tf.ones((354, 1), dtype=tf.int32)
b = tf.reduce_sum(a)
with tf.Session() as sess:
  print(sess.run(b))
```

```
354
```

宣告 Optimizer 與學習速率

In [13]:
```
learning_rate = 0.001
opt = tf.train.GradientDescentOptimizer(learning_rate)
optimizer = opt.minimize(loss)
```

建構 TensorFlow 計算圖形完整程式碼

In [14]:

```
import tensorflow as tf

X_train_shape = X_train.shape
y_train_shape = y_train.shape
W_shape = (X_train_shape[1], 1)
b_shape = (1,)
learning_rate = 0.001
file_writer_path = "./graphs/linear-regression"

# placeholders
X = tf.placeholder(tf.float32, X_train_shape)
y = tf.placeholder(tf.float32, y_train_shape)
# variables
W = tf.Variable(tf.random_normal(W_shape))
b = tf.Variable(tf.random_normal(b_shape))
# prediction
y_pred = tf.matmul(X, W) + b
# loss
loss = tf.reduce_sum((y - y_pred)**2)
# optimizer
opt = tf.train.GradientDescentOptimizer(learning_rate)
optimizer = opt.minimize(loss)
```

訓練

In [15]:

```
n_steps = 1000
file_writer_path = "./graphs/linear-regression"

with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss = sess.run([optimizer, loss], feed_dict=feed_dict)
    if i % 100 == 0:
      print("step {}, loss: {}".format(i, loss))
  w_final, b_final = sess.run([W, b])
```

```
step 0, loss: 172169664.0

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in __init__(self, fetches, contraction_fn)
    299         self._unique_fetches.append(ops.get_default_graph().as_graph_
element(
--> 300             fetch, allow_tensor=True, allow_operation=True))
    301       except TypeError as e:

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/fra
mework/ops.py in as_graph_element(self, obj, allow_tensor, allow_operation)
   3489     with self._lock:
-> 3490       return self._as_graph_element_locked(obj, allow_tensor, allow_o
peration)
   3491

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/fra
```

```
mework/ops.py in _as_graph_element_locked(self, obj, allow_tensor, allow_oper
ation)
   3578       raise TypeError("Can not convert a %s into a %s." % (type(obj).
__name__,
-> 3579                                                           types_st
r))
   3580

TypeError: Can not convert a float32 into a Tensor or Operation.

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-15-d806a2b66b6d> in <module>
     11       y: y_train
     12     }
---> 13     _, loss = sess.run([optimizer, loss], feed_dict=feed_dict)
     14     if i % 100 == 0:
     15       print("step {}, loss: {}".format(i, loss))

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in run(self, fetches, feed_dict, options, run_metadata)
    927     try:
    928       result = self._run(None, fetches, feed_dict, options_ptr,
--> 929                          run_metadata_ptr)
    930       if run_metadata:
    931         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in _run(self, handle, fetches, feed_dict, options, run_metadat
a)
   1135     # Create a fetch handler to take care of the structure of fetche
s.
   1136     fetch_handler = _FetchHandler(
-> 1137         self._graph, fetches, feed_dict_tensor, feed_handles=feed_han
dles)
   1138
   1139     # Run request and get response.
```

```
~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in __init__(self, graph, fetches, feeds, feed_handles)
    469     """
    470     with graph.as_default():
--> 471       self._fetch_mapper = _FetchMapper.for_fetch(fetches)
    472     self._fetches = []
    473     self._targets = []

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in for_fetch(fetch)
    259     elif isinstance(fetch, (list, tuple)):
    260       # NOTE(touts): This is also the code path for namedtuples.
--> 261       return _ListFetchMapper(fetch)
    262     elif isinstance(fetch, collections.Mapping):
    263       return _DictFetchMapper(fetch)

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in __init__(self, fetches)
    368     """
    369     self._fetch_type = type(fetches)
--> 370     self._mappers = [_FetchMapper.for_fetch(fetch) for fetch in fetch
es]
    371     self._unique_fetches, self._value_indices = _uniquify_fetches(sel
f._mappers)
    372

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in <listcomp>(.0)
    368     """
    369     self._fetch_type = type(fetches)
--> 370     self._mappers = [_FetchMapper.for_fetch(fetch) for fetch in fetch
es]
    371     self._unique_fetches, self._value_indices = _uniquify_fetches(sel
f._mappers)
    372

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
```

```
ent/session.py in for_fetch(fetch)
    269           if isinstance(fetch, tensor_type):
    270             fetches, contraction_fn = fetch_fn(fetch)
--> 271             return _ElementFetchMapper(fetches, contraction_fn)
    272       # Did not find anything.
    273       raise TypeError('Fetch argument %r has invalid type %r' % (fetch,

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/python/cli
ent/session.py in __init__(self, fetches, contraction_fn)
    302           raise TypeError('Fetch argument %r has invalid type %r, '
    303                           'must be a string or Tensor. (%s)' %
--> 304                           (fetch, type(fetch), str(e)))
    305         except ValueError as e:
    306           raise ValueError('Fetch argument %r cannot be interpreted as
 a '

TypeError: Fetch argument 172169660.0 has invalid type <class 'numpy.float3
2'>, must be a string or Tensor. (Can not convert a float32 into a Tensor or
 Operation.)
```

發生了什麼事情?

```
for i in range(n_steps):
  # ...
  # 這邊的物件命名同樣為 loss，但型別已經不同，feed_dict 是張量，但 fetch 是 ndarray
  _, loss = sess.run([optimizer, loss], feed_dict=feed_dict)
  # ...
```

修改物件命名之後再來試一次

In [16]:
```
import tensorflow as tf

X_train_shape = X_train.shape
y_train_shape = y_train.shape
W_shape = (X_train_shape[1], 1)
b_shape = (1,)
learning_rate = 0.001
file_writer_path = "./graphs/linear-regression"

# placeholders
X = tf.placeholder(tf.float32, X_train_shape)
y = tf.placeholder(tf.float32, y_train_shape)
# variables
W = tf.Variable(tf.random_normal(W_shape))
b = tf.Variable(tf.random_normal(b_shape))
# prediction
y_pred = tf.matmul(X, W) + b
# loss
loss = tf.reduce_sum((y - y_pred)**2)
# optimizer
opt = tf.train.GradientDescentOptimizer(learning_rate)
optimizer = opt.minimize(loss)
```

In [17]:

```
n_steps = 1000
file_writer_path = "./graphs/linear-regression"

with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss_ = sess.run([optimizer, loss], feed_dict=feed_dict)
    if i % 100 == 0:
      print("step {}, loss: {}".format(i, loss_))
  w_final, b_final = sess.run([W, b])
```

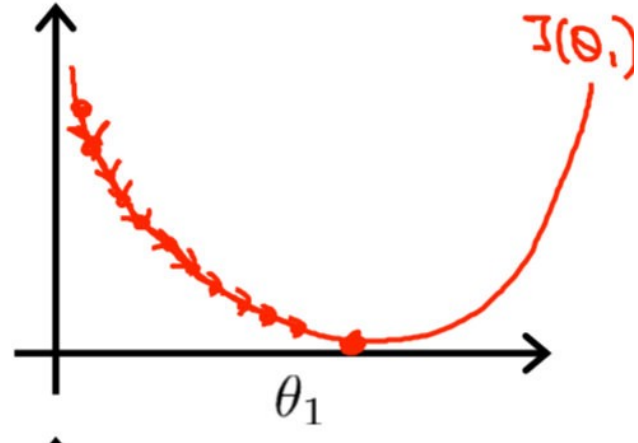
```
step 0, loss: 79753160.0
step 100, loss: nan
step 200, loss: nan

step 300, loss: nan
step 400, loss: nan
step 500, loss: nan
step 600, loss: nan
step 700, loss: nan
step 800, loss: nan
step 900, loss: nan
```

為什麼 loss 都是 nan?

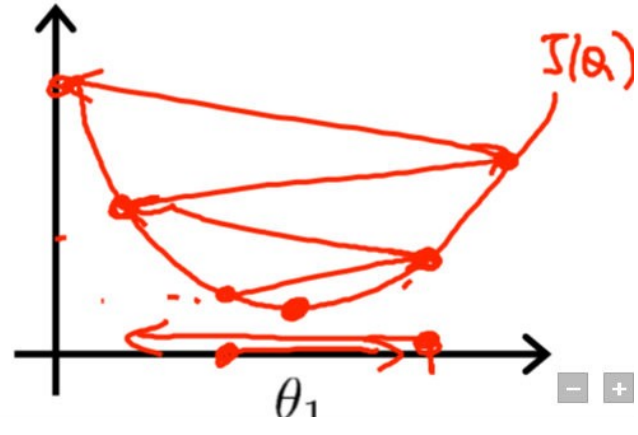
Learning Rate 太大的緣故

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Source: [Machine Learning | Coursera (https://www.coursera.org/learn/machine-learning)](https://www.coursera.org/learn/machine-learning)

降低 Learning Rate 之後再來試一次

In [18]:

```
import tensorflow as tf

X_train_shape = X_train.shape
y_train_shape = y_train.shape
W_shape = (X_train_shape[1], 1)
b_shape = (1,)
learning_rate = 0.00000001
file_writer_path = "./graphs/linear-regression"

# placeholders
X = tf.placeholder(tf.float32, X_train_shape)
y = tf.placeholder(tf.float32, y_train_shape)
# variables
W = tf.Variable(tf.random_normal(W_shape))
b = tf.Variable(tf.random_normal(b_shape))
# prediction
y_pred = tf.matmul(X, W) + b
# loss
loss = tf.reduce_sum((y - y_pred)**2)
# optimizer
opt = tf.train.GradientDescentOptimizer(learning_rate)
optimizer = opt.minimize(loss)
```

In [19]:

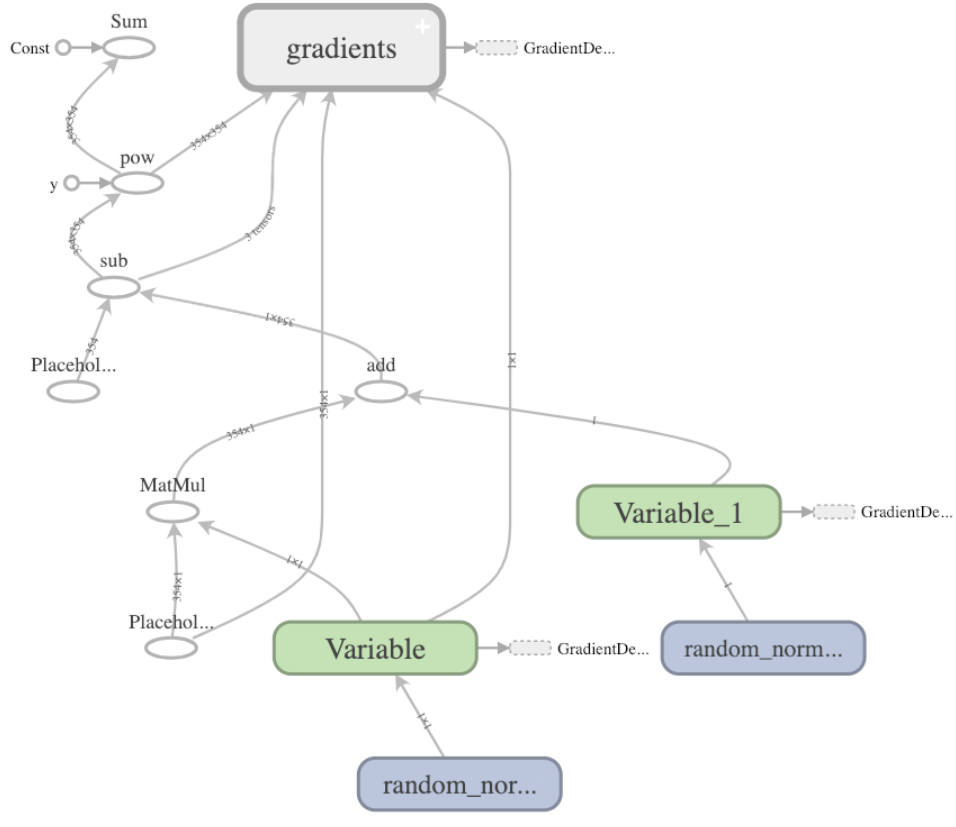
```
n_steps = 1000
file_writer_path = "./graphs/linear-regression"

with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss_ = sess.run([optimizer, loss], feed_dict=feed_dict)
    if i % 100 == 0:
      print("step {}, loss: {}".format(i, loss_))
  w_final, b_final = sess.run([W, b])
```

```
step 0, loss: 27572990.0
step 100, loss: 24951664.0
step 200, loss: 23302854.0

step 300, loss: 21845508.0
step 400, loss: 20557392.0
step 500, loss: 19418772.0
step 600, loss: 18412378.0
step 700, loss: 17522820.0
step 800, loss: 16736555.0
step 900, loss: 16041554.0
```

檢視 TensorBoard

回到 Terminal 啟動 TensorBoard

```
tensorboard --logdir=path/to/log-directory
```


Sum
Const
gradients
GradientDe...
pow
y
sub
Placehol...
add
MatMul
Placehol...
Variable
GradientDe...
random_nor...
Variable_1
GradientDe...
random_norm...


Variable
gradients
Variable_1
GradientDesc...

加入 Name Scopes

TensorFlow 不知道哪些節點應該歸類在一起

利用 `with tf.name_scope(name_of_that_scope)` 將節點歸類起來，讓 Graph 更簡潔

```
with tf.name_scope(name_of_that_scope):
  # declare op_1
  # declare op_2
  # ...
```

In [20]:

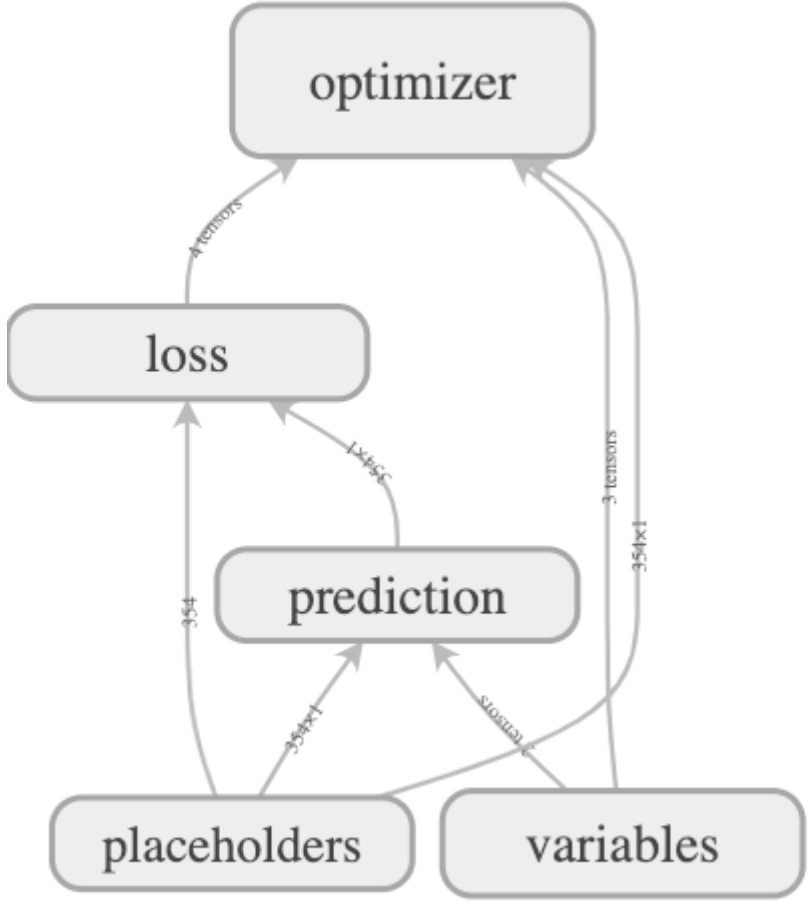
```
import tensorflow as tf

X_train_shape = X_train.shape
y_train_shape = y_train.shape
W_shape = (X_train_shape[1], 1)
b_shape = (1,)
learning_rate = 0.00000001
file_writer_path = "./graphs/linear-regression"

# placeholders
with tf.name_scope("placeholders"):
  X = tf.placeholder(tf.float32, X_train_shape)
  y = tf.placeholder(tf.float32, y_train_shape)
# variables
with tf.name_scope("variables"):
  W = tf.Variable(tf.random_normal(W_shape))
  b = tf.Variable(tf.random_normal(b_shape))
# prediction
with tf.name_scope("prediction"):
  y_pred = tf.matmul(X, W) + b
# loss
with tf.name_scope("loss"):
  loss = tf.reduce_sum((y - y_pred)**2)
# optimizer
with tf.name_scope("optimizer"):
  opt = tf.train.GradientDescentOptimizer(learning_rate)
  optimizer = opt.minimize(loss)
```

In [21]:

```
n_steps = 1000
file_writer_path = "./graphs/linear-regression"

with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss_ = sess.run([optimizer, loss], feed_dict=feed_dict)
    if i % 100 == 0:
      print("step {}, loss: {}".format(i, loss_))
  w_final, b_final = sess.run([W, b])
```

```
step 0, loss: 42952512.0
step 100, loss: 25857140.0
step 200, loss: 24103244.0

step 300, loss: 22552958.0
step 400, loss: 21182674.0
step 500, loss: 19971490.0
step 600, loss: 18900916.0
step 700, loss: 17954632.0
step 800, loss: 17118220.0
step 900, loss: 16378917.0
```

Graph with name scopes



調整訓練的次數與觀察 loss 是否漸趨收斂

In [22]:

```
n_steps = 10000
file_writer_path = "./graphs/linear-regression"
loss_history = []

with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss_ = sess.run([optimizer, loss], feed_dict=feed_dict)
    loss_history.append(loss_)
    if i % 500 == 0:
      print("step {}, loss: {}".format(i, loss_))
  w_final, b_final = sess.run([W, b])
```

```
step 0, loss: 49784224.0
step 500, loss: 19940696.0
step 1000, loss: 15708821.0
step 1500, loss: 13425650.0

step 2000, loss: 12193802.0
step 2500, loss: 11529185.0
step 3000, loss: 11170614.0
step 3500, loss: 10977148.0
step 4000, loss: 10872768.0
step 4500, loss: 10816446.0
step 5000, loss: 10786077.0
step 5500, loss: 10769688.0
step 6000, loss: 10760842.0
step 6500, loss: 10756062.0
step 7000, loss: 10753494.0
step 7500, loss: 10752112.0
step 8000, loss: 10751349.0
```

```
step 8500, loss: 10750960.0
step 9000, loss: 10750726.0
step 9500, loss: 10750618.0
```

In [23]:
```
import matplotlib.pyplot as plt

plt.plot(range(n_steps), loss_history)
plt.title("Loss Summary")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

In [24]:
```
import numpy as np

y_pred = np.dot(X_test, w_final) + b_final[0]
mse = mean_squared_error(y_test, y_pred)
print(w_final)
print(b_final)
print(mse)
```

```
[[0.00293602]]
[22.712822]
81.65840721929018
```

將 Loss 加入 TensorBoard 中的 Scalar 頁籤

- 增加一個 summaries 的 name scope
- 每一次的 step（epoch）都要將 loss 記錄起來

In [25]:

```
import tensorflow as tf

X_train_shape = X_train.shape
y_train_shape = y_train.shape
W_shape = (X_train_shape[1], 1)
b_shape = (1,)
learning_rate = 0.00000001
file_writer_path = "./graphs/linear-regression"

# placeholders
with tf.name_scope("placeholders"):
  X = tf.placeholder(tf.float32, X_train_shape)
  y = tf.placeholder(tf.float32, y_train_shape)
# variables
with tf.name_scope("variables"):
  W = tf.Variable(tf.random_normal(W_shape))
  b = tf.Variable(tf.random_normal(b_shape))
# prediction
with tf.name_scope("prediction"):
  y_pred = tf.matmul(X, W) + b
# loss
with tf.name_scope("loss"):
  loss = tf.reduce_sum((y - y_pred)**2)
# optimizer
with tf.name_scope("optimizer"):
  opt = tf.train.GradientDescentOptimizer(learning_rate)
  optimizer = opt.minimize(loss)
# summaries
with tf.name_scope("summary"):
  tf.summary.scalar("loss", loss)
  merged = tf.summary.merge_all()
```

In [26]:

```
n_steps = 10000
file_writer_path = "./graphs/linear-regression"
loss_history = []
train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())

with tf.Session() as sess:
  sess.run(W.initializer)
  sess.run(b.initializer)
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss_, summary = sess.run([optimizer, loss, merged], feed_dict=feed_dict)
    train_writer.add_summary(summary, i)
    if i % 500 == 0:
      print("step {}, loss: {}".format(i, loss_))
  w_final, b_final = sess.run([W, b])
```

```
step 0, loss: 82744600.0
step 500, loss: 20278968.0
step 1000, loss: 15891350.0
step 1500, loss: 13524104.0

step 2000, loss: 12246928.0
step 2500, loss: 11557848.0
step 3000, loss: 11186085.0
step 3500, loss: 10985492.0
step 4000, loss: 10877276.0
step 4500, loss: 10818892.0
step 5000, loss: 10787388.0
step 5500, loss: 10770382.0
step 6000, loss: 10761230.0
step 6500, loss: 10756275.0
step 7000, loss: 10753610.0
step 7500, loss: 10752166.0
step 8000, loss: 10751382.0
```

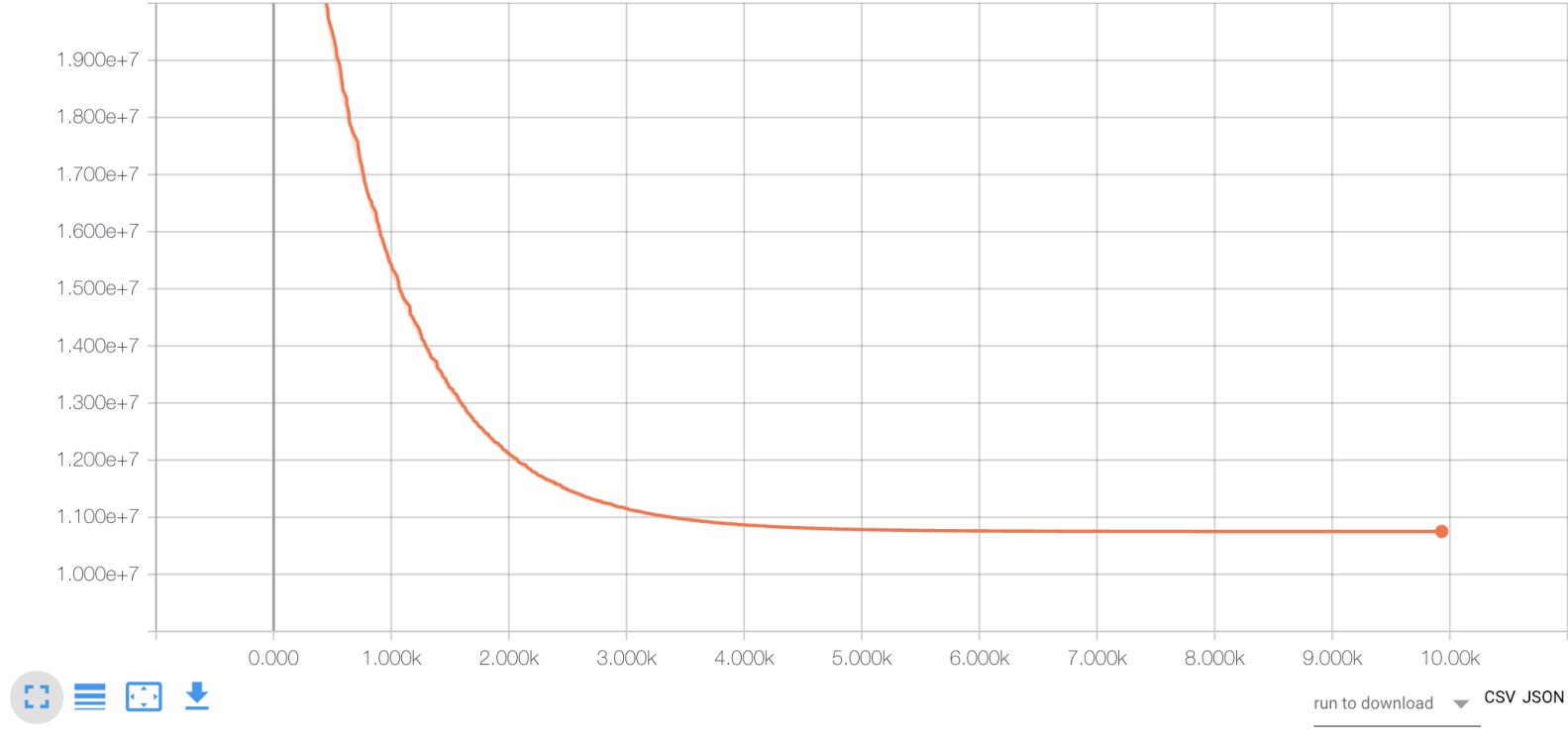
```
step 8500, loss: 10750964.0
step 9000, loss: 10750736.0
step 9500, loss: 10750624.0
```

summary

1

loss

tag: summary/loss



run to download CSV JSON

如果 Loss Function 沒有收斂怎麼辦?

- 增加 Steps
- 增加 Learning rate
- 更換 Optimizer

如果重新訓練的時候產生了錯誤呢?

- Restart Kernel 清空 Graph
- 或者在訓練之前執行:

```
tf.reset_default_graph()
```

隨堂練習

以 Boston 建立一個複迴歸模型： MEDV ~ RM + AGE + LSTAT

In [27]:
```
X_arr = boston.data[:, [4, 5, -1]]
y_arr = boston.target
print(X_arr.shape)
print(y_arr.shape)
```

```
(506, 3)
(506,)
```

In [31]:
```
import matplotlib.pyplot as plt

plt.plot(range(n_steps), loss_history)
plt.title("Loss Summary")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

In [33]:
```
print(w_final)
print(b_final)
print(mse)
```

```
[[1.4277564 ]
 [3.0061457 ]
 [0.19918345]]
[0.35683057]
81.85711376674335
```