Linear models

Puteaux, Fall/Winter 2020-2021

```
################################
##                            ##
##  Deep Learning in Python   ##
##                            ##
################################
```

```
§2 Introduction to TensorFlow in Python
```

```
§2.2 Linear models
```

1 Input data

1.1 How to import data for use in TensorFlow?

- **Data can be imported using TensorFlow**:
  - useful for managing complex pipelines
- **The simpler option used to import data**:
  - import data using pandas
  - convert data to NumPy array
  - use in TensorFlow without modification
- Pandas also has methods for handling data in other formats:
  - e.g., `read_json()` , `read_html()` , `read_excel()`

1.2 Code of how to import and convert data:

[1]:
```python
# Import numpy and pandas
import numpy as np
import pandas as pd

# Load data from csv
housing = pd.read_csv('ref1. King county house sales.csv')

# Convert to numpy array
housing = np.array(housing)
```

```
print(housing)
```

```
[[7129300520 '20141013T000000' 221900.0 … -122.257 1340 5650]
 [6414100192 '20141209T000000' 538000.0 … -122.319 1690 7639]
 [5631500400 '20150225T000000' 180000.0 … -122.23299999999999 2720 8062]
 …
 [1523300141 '20140623T000000' 402101.0 … -122.29899999999999 1020 2007]
 [291310100 '20150116T000000' 400000.0 … -122.069 1410 1287]
 [1523300157 '20141015T000000' 325000.0 … -122.29899999999999 1020 1357]]
```

1.3 What are the parameters of `read_csv()`?

Parameter	Description	Default
`filepath_or_buffer`	Accepts a file path or a URL.	`None`
`sep`	Delimiter between columns.	`,`
`delim_whitespace`	Boolean for whether to delimit whitespace.	`False`
`encoding`	Specifies encoding to be used if any.	`None`

1.4 How to use mixed type datasets?

date	price	bedrooms
20141013T000000	221900	3
20141209T000000	538000	3
20150225T000000	180000	2
20141209T000000	604000	4
20150218T000000	510000	3
20140627T000000	257500	3
20150115T000000	291850	3
20150415T000000	229500	3

floors	waterfront	view
1	0	0
2	0	0
1	1	0
1	0	0
1	0	2
2	0	0
1	0	4
1	0	0

1.5 Code of setting the data type:

[2]:
```
# Load KC dataset
housing = pd.read_csv('ref1. King county house sales.csv')

# Convert price column to float32
price = np.array(housing['price'], np.float32)

# Convert waterfront column to Boolean
waterfront = np.array(housing['waterfront'], np.bool)

print(price)
print(waterfront)
```

```
[221900. 538000. 180000. … 402101. 400000. 325000.]
[False False False … False False False]
```

[3]:
```
import tensorflow as tf
```

[4]:
```
# Load KC dataset
housing = pd.read_csv('ref1. King county house sales.csv')

# Convert price column to float32
price = tf.cast(housing['price'], tf.float32)

# Convert waterfront column to Boolean
waterfront = tf.cast(housing['waterfront'], tf.bool)

print(price)
print(waterfront)
```

```
tf.Tensor([221900. 538000. 180000. … 402101. 400000. 325000.], shape=(21613,),
dtype=float32)
tf.Tensor([False False False … False False False], shape=(21613,), dtype=bool)
```

1.6 Practice exercises for input data:

▶ **Pandas data loading practice:**

[5]:
```
# Import pandas under the alias pd
import pandas as pd

# Assign the path to a string variable named data_path
data_path = 'ref1. King county house sales.csv'

# Load the dataset as a dataframe named housing
housing = pd.read_csv(data_path)

# Print the price column of housing
print(housing['price'])
```

```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           …
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21613, dtype: float64
```

▶ **Data type setting practice:**

[6]:
```
# Import numpy and tensorflow with their standard aliases
import numpy as np
import tensorflow as tf

# Use a numpy array to define price as a 32-bit float
price = np.array(housing['price'], np.float32)

# Define waterfront as a Boolean using cast
waterfront = tf.cast(housing['waterfront'], tf.bool)

# Print price and waterfront
print(price)
print(waterfront)
```

```
[221900. 538000. 180000. … 402101. 400000. 325000.]
tf.Tensor([False False False … False False False], shape=(21613,), dtype=bool)
```

2 Loss functions

2.1 What is the loss functions?

- **Fundamental TensorFlow operation**:
  - used to train a model
  - measure of model fit
- **Higher value → worse fit**:
  - minimize the loss function

2.2 What are the common loss functions in TensorFlow?

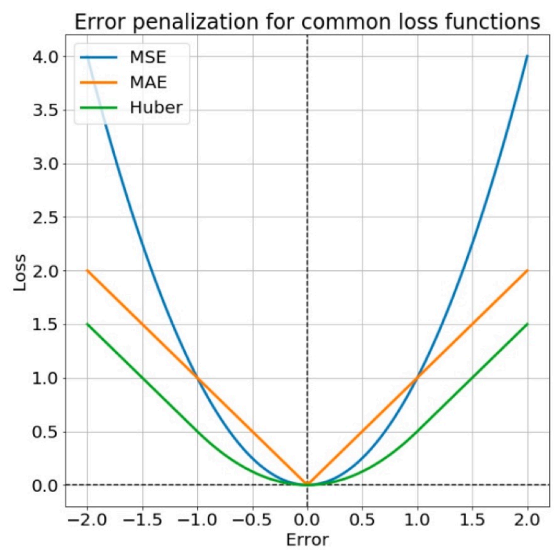
- **TensorFlow has operations for common loss functions**:
  - mean squared error (MSE)
  - mean absolute error (MAE)
  - huber error
- **Loss functions are accessible from `tf.keras.losses()`**:
  - `tf.keras.losses.mse()`
  - `tf.keras.losses.mae()`
  - `tf.keras.losses.Huber()`

2.3 Why is it in need to care about loss functions?

- **MSE**:
  - strongly penalizes outliers

- high (gradient) sensitivity near minimum

- **MAE**:
  - scales linearly with the size of the error
  - low sensitivity near minimum
- **Huber**:
  - similar to MSE near minimum
  - similar to MAE away from the minimum



2.4 Code of defining the loss function:

[7]:
```
import pandas as pd
import numpy as np

housing = pd.read_csv('ref1. King county house sales.csv')

price_log = np.log(np.array(housing['price'], np.float32))
size_log = np.log(np.array(housing['sqft_lot'], np.float32))
targets = price_log
features = size_log
intercept = 0.1
slope = 0.1

predictions = intercept + features * slope
```

[8]:
```
# Import TensorFlow under standard alias
import tensorflow as tf

# Compute the MSE loss
```

```
loss = tf.keras.losses.mse(targets, predictions)

loss
```

[8]: <tf.Tensor: shape=(), dtype=float32, numpy=145.44653>

[9]:
```
# Define a linear regression model
def linear_regression(intercept, slope=slope, features=features):
    return intercept + features * slope
```

[10]:
```
# Define a loss function to compute the MSE
def loss_function(intercept, slope, targets=targets, features=features):
    # Compute the predictions for a linear model
    predictions = linear_regression(intercept, slope, features)

    # Return the loss
    return tf.keras.losses.mse(targets, predictions)
```

[11]:
```
from sklearn.model_selection import train_test_split

train_features, test_features, train_targets, test_targets = train_test_split(
    features, targets, test_size=0.3, random_state=42)
```

[12]:
```
# Compute the loss for test data inputs
loss_function(intercept, slope, test_targets, test_features)
```

[12]: <tf.Tensor: shape=(), dtype=float32, numpy=145.57338>

[13]:
```
# Compute the loss for default data inputs
loss_function(intercept, slope)
```

[13]: <tf.Tensor: shape=(), dtype=float32, numpy=145.44653>

2.5 Practice exercises for loss functions:

▶ **Package pre-loading:**

[14]:
```
import pandas as pd
import numpy as np
```

▶ **Data pre-loading:**

[15]:
```
housing = pd.read_csv('ref1. King county house sales.csv')
price = np.array(housing['price'], np.float32)

predictions = np.loadtxt('ref5. Predictions.csv', delimiter=',')
```

▶ **TensorFlow loss functions practice:**

[16]:
```python
# Import the keras module from tensorflow
from tensorflow import keras

# Compute the mean squared error (mse)
loss = keras.losses.mse(price, predictions)

# Print the mean squared error (mse)
print(loss.numpy())
```

```
141171604777.141
```

[17]:
```python
# Import the keras module from tensorflow
from tensorflow import keras

# Compute the mean absolute error (mae)
loss = keras.losses.mae(price, predictions)

# Print the mean absolute error (mae)
print(loss.numpy())
```

```
268827.9930163703
```

▶ **Package re-pre-loading:**

[18]:
```python
from tensorflow import constant, Variable, float32
```

▶ **Data re-pre-loading:**

[19]:
```python
features = constant([1., 2., 3., 4., 5.], dtype=float32)
targets = constant([2., 4., 6., 8., 10.], dtype=float32)
```

▶ **Loss function modification practice:**

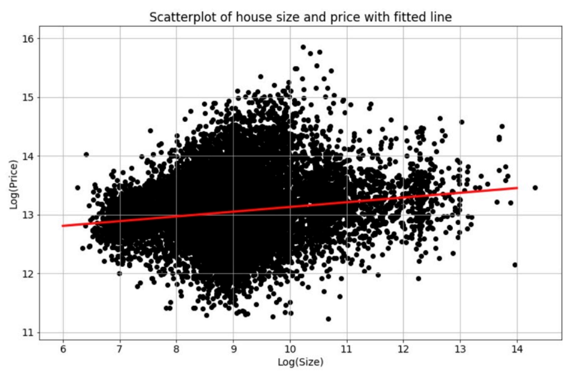
[20]:
```python
# Initialize a variable named scalar
scalar = Variable(1.0, float32)


# Define the model
def model(scalar, features=features):
    return scalar * features


# Define a loss function
def loss_function(scalar, features=features, targets=targets):
    # Compute the predicted values
    predictions = model(scalar, features)

    # Return the mean absolute error loss
    return keras.losses.mae(targets, predictions)
```

```
# Evaluate the loss function and print the loss
print(loss_function(scalar).numpy())
```

```
3.0
```

3 Linear regression

3.1 What is linear regression?



3.2 How to make a linear regression model?

- A linear regression model assumes a linear relationship:
  - e.g., for the example of king county house sales dataset,
    - $price = intercept + size \times slope + error$
- Univariate regression models have only one feature,
  - e.g., for the example above, there could be only one feature, `size`
- Multiple regression models have more than one feature,
  - e.g., for the example above, there also could be two feature, `size` and `location`

3.3 Code of linear regression in TensorFlow:

[21]:
```
import pandas as pd
import numpy as np
import tensorflow as tf

housing = pd.read_csv('ref1. King county house sales.csv')
```

[22]:
```
# Define the targets and features
price = np.array(housing['price'], np.float32)
size = np.array(housing['sqft_living'], np.float32)
```

```python
# Define the intercept and slope
intercept = tf.Variable(0.1, np.float32)
slope = tf.Variable(0.1, np.float32)
```

[23]:
```python
# Define a linear regression model
def linear_regression(intercept, slope, features=size):
    return intercept + features * slope
```

[24]:
```python
# Compute the predicted values and loss
def loss_function(intercept, slope, targets=price, features=size):
    predictions = linear_regression(intercept, slope)
    return tf.keras.losses.mse(targets, predictions)
```

[25]:
```python
# Define an optimization operation
opt = tf.keras.optimizers.Adam()
```

[26]:
```python
# Minimize the loss function and print the loss
for j in range(50):
    opt.minimize(lambda: loss_function(intercept, slope),
                 var_list=[intercept, slope])
    print(loss_function(intercept, slope))
```

```
tf.Tensor(426196570000.0, shape=(), dtype=float32)
tf.Tensor(426193880000.0, shape=(), dtype=float32)
tf.Tensor(426191160000.0, shape=(), dtype=float32)
tf.Tensor(426188400000.0, shape=(), dtype=float32)
tf.Tensor(426185700000.0, shape=(), dtype=float32)
tf.Tensor(426182930000.0, shape=(), dtype=float32)
tf.Tensor(426180280000.0, shape=(), dtype=float32)
tf.Tensor(426177500000.0, shape=(), dtype=float32)
tf.Tensor(426174800000.0, shape=(), dtype=float32)
tf.Tensor(426172060000.0, shape=(), dtype=float32)
tf.Tensor(426169340000.0, shape=(), dtype=float32)
tf.Tensor(426166650000.0, shape=(), dtype=float32)
tf.Tensor(426163930000.0, shape=(), dtype=float32)
tf.Tensor(426161180000.0, shape=(), dtype=float32)
tf.Tensor(426158520000.0, shape=(), dtype=float32)
tf.Tensor(426155770000.0, shape=(), dtype=float32)
tf.Tensor(426153050000.0, shape=(), dtype=float32)
tf.Tensor(426150300000.0, shape=(), dtype=float32)
tf.Tensor(426147580000.0, shape=(), dtype=float32)
tf.Tensor(426144900000.0, shape=(), dtype=float32)
tf.Tensor(426142170000.0, shape=(), dtype=float32)
tf.Tensor(426139420000.0, shape=(), dtype=float32)
tf.Tensor(426136730000.0, shape=(), dtype=float32)
tf.Tensor(426134080000.0, shape=(), dtype=float32)
tf.Tensor(426131300000.0, shape=(), dtype=float32)
```

```
tf.Tensor(426128600000.0, shape=(), dtype=float32)
tf.Tensor(426125850000.0, shape=(), dtype=float32)
tf.Tensor(426123130000.0, shape=(), dtype=float32)
tf.Tensor(426120400000.0, shape=(), dtype=float32)
tf.Tensor(426117660000.0, shape=(), dtype=float32)
tf.Tensor(426114940000.0, shape=(), dtype=float32)
tf.Tensor(426112320000.0, shape=(), dtype=float32)
tf.Tensor(426109530000.0, shape=(), dtype=float32)
tf.Tensor(426106780000.0, shape=(), dtype=float32)
tf.Tensor(426104060000.0, shape=(), dtype=float32)
tf.Tensor(426101370000.0, shape=(), dtype=float32)
tf.Tensor(426098600000.0, shape=(), dtype=float32)
tf.Tensor(426095900000.0, shape=(), dtype=float32)
tf.Tensor(426093200000.0, shape=(), dtype=float32)
tf.Tensor(426090500000.0, shape=(), dtype=float32)
tf.Tensor(426087780000.0, shape=(), dtype=float32)
tf.Tensor(426085100000.0, shape=(), dtype=float32)
tf.Tensor(426082300000.0, shape=(), dtype=float32)
tf.Tensor(426079620000.0, shape=(), dtype=float32)
tf.Tensor(426076900000.0, shape=(), dtype=float32)
tf.Tensor(426074140000.0, shape=(), dtype=float32)
tf.Tensor(426071460000.0, shape=(), dtype=float32)
tf.Tensor(426068740000.0, shape=(), dtype=float32)
tf.Tensor(426066020000.0, shape=(), dtype=float32)
tf.Tensor(426063330000.0, shape=(), dtype=float32)
```

3.4 Practice exercises for linear regression:

▶ **Package pre-loading:**

[27]:
```
import pandas as pd
import numpy as np
from tensorflow import add, multiply, keras
```

▶ **Data pre-loading:**

[28]:
```
housing = pd.read_csv('ref1. King county house sales.csv')
price_log = np.log(np.array(housing['price'], np.float32))
size_log = np.log(np.array(housing['sqft_lot'], np.float32))
```

▶ **Linear regression set-up practice:**

[29]:
```
# Define a linear regression model
def linear_regression(intercept, slope, features=size_log):
    return add(intercept, multiply(features, slope))


# Set loss_function() to take the variables as arguments
```

```
def loss_function(intercept, slope, features=size_log, targets=price_log):
    # Set the predicted values
    predictions = linear_regression(intercept, slope, features)

    # Return the mean squared error loss
    return keras.losses.mse(targets, predictions)


# Compute the loss for different slope and intercept values
print(loss_function(0.1, 0.1).numpy())
print(loss_function(0.1, 0.5).numpy())
```

```
145.44653
71.866
```

▶ **Package re-pre-loading:**

[30]:
```
from tensorflow import Variable
import matplotlib.pyplot as plt
```

▶ **Data re-pre-loading:**

[31]:
```
intercept = Variable(5, dtype=np.float32)
slope = Variable(0.001, dtype=np.float32)
```

▶ **Code pre-loading:**

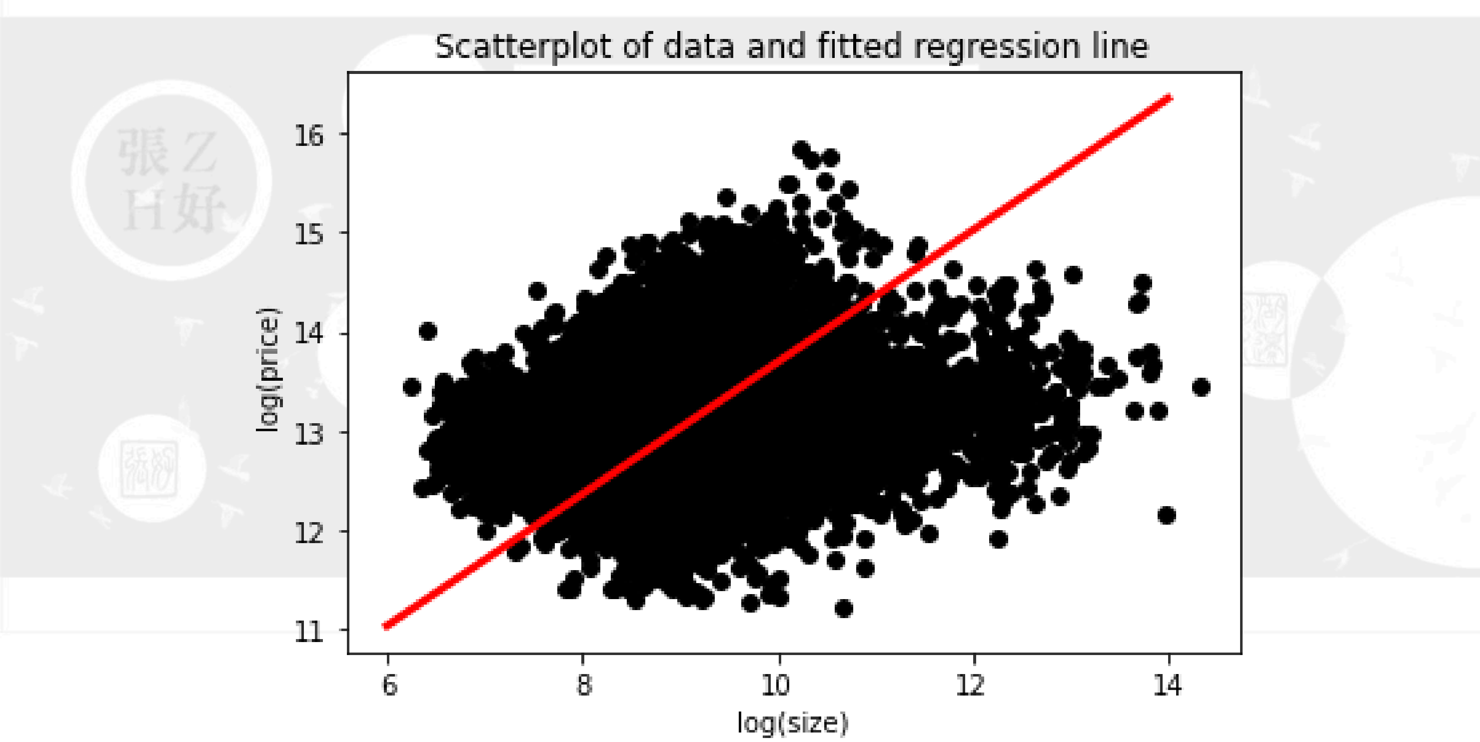
[32]:
```
def plot_results(intercept, slope):
    size_range = np.linspace(6, 14, 100)
    price_pred = [intercept + slope * s for s in size_range]
    plt.scatter(size_log, price_log, color='black')
    plt.plot(size_range, price_pred, linewidth=3.0, color='red')
    plt.xlabel('log(size)')
    plt.ylabel('log(price)')
    plt.title('Scatterplot of data and fitted regression line')
    plt.show()
```

▶ **Linear model training practice:**

[33]:
```
# Initialize an adam optimizer
opt = keras.optimizers.Adam(0.5)

for j in range(100):
    # Apply minimize, pass the loss function, and supply the variables
    opt.minimize(lambda: loss_function(intercept, slope),
                 var_list=[intercept, slope])

    # Print every 10th value of the loss
    if j % 10 == 0:
```

```
        print(loss_function(intercept, slope).numpy())

# Plot data and regression line
plot_results(intercept, slope)
```

```
9.681214
11.737101
1.1297756
1.6701097
0.80904144
0.81259125
0.6220206
0.6118439
0.5933767
0.57028323
```



▶ **Data re-pre-loading:**

```
[34]: bedrooms = np.array(housing['bedrooms'], np.float32)
      params = Variable([0.1, 0.05, 0.02], dtype=np.float32)
```

▶ **Code pre-loading:**

[35]:
```
def print_results(params):
    return print(
        'loss: {:0.3f}, intercept: {:0.3f}, slope_1: {:0.3f}, slope_2: {:0.3f}'
        .format(
            loss_function(params).numpy(), params[0].numpy(),
            params[1].numpy(), params[2].numpy()))
```

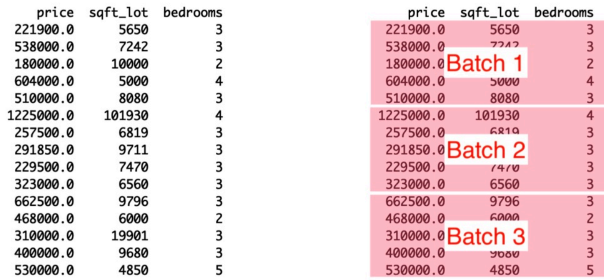
▶ **Multiple linear regression practice:**

[36]:
```
# Define the linear regression model
def linear_regression(params, feature1=size_log, feature2=bedrooms):
    return params[0] + feature1 * params[1] + feature2 * params[2]


# Define the loss function
def loss_function(params,
                  targets=price_log,
                  feature1=size_log,
                  feature2=bedrooms):
    # Set the predicted values
    predictions = linear_regression(params, feature1, feature2)

    # Use the mean absolute error loss
    return keras.losses.mae(targets, predictions)


# Define the optimize operation
opt = keras.optimizers.Adam()

# Perform minimization and print trainable variables
for j in range(10):
    opt.minimize(lambda: loss_function(params), var_list=[params])
    print_results(params)
```

```
loss: 12.418, intercept: 0.101, slope_1: 0.051, slope_2: 0.021
loss: 12.404, intercept: 0.102, slope_1: 0.052, slope_2: 0.022
loss: 12.391, intercept: 0.103, slope_1: 0.053, slope_2: 0.023
loss: 12.377, intercept: 0.104, slope_1: 0.054, slope_2: 0.024
loss: 12.364, intercept: 0.105, slope_1: 0.055, slope_2: 0.025
loss: 12.351, intercept: 0.106, slope_1: 0.056, slope_2: 0.026
loss: 12.337, intercept: 0.107, slope_1: 0.057, slope_2: 0.027
loss: 12.324, intercept: 0.108, slope_1: 0.058, slope_2: 0.028
loss: 12.311, intercept: 0.109, slope_1: 0.059, slope_2: 0.029
loss: 12.297, intercept: 0.110, slope_1: 0.060, slope_2: 0.030
```

4 Linear regression

4.1 What is batch training?



price	sqft_lot	bedrooms
221900.0	5650	3
538000.0	7242	3
180000.0	10000	2
604000.0	5000	4
510000.0	8080	3
1225000.0	101930	4
257500.0	6819	3
291850.0	9711	3
229500.0	7470	3
323000.0	6560	3
662500.0	9796	3
468000.0	6000	2
310000.0	19901	3
400000.0	9680	3
530000.0	4850	5

4.2 What is the `chunksize` parameter?

- `pandas.read_csv()` allows to load data in batches:
  - avoid loading entire dataset
  - `chunksize` parameter provides batch size

4.3 Code of the `chunksize` parameter:

[37]:
```python
# Import pandas and numpy
import pandas as pd
import numpy as np

# Load data in batches
for batch in pd.read_csv('ref1. King county house sales.csv', chunksize=100):
    # Extract price column
    price = np.array(batch['price'], np.float32)

    # Extract size column
    size = np.array(batch['sqft_living'], np.float32)
```

4.4 Code of training a linear model in batches:

[38]:
```python
# Import tensorflow, pandas, and numpy
import tensorflow as tf
import pandas as pd
import numpy as np
```

[39]:
```python
# Define trainable variables
intercept = tf.Variable(0.1, tf.float32)
slope = tf.Variable(0.1, tf.float32)
```

[40]:
```python
# Define the model
def linear_regression(intercept, slope, features):
```

```
    return intercept + features * slope
```

[41]:
```
# Compute predicted values and return loss function
def loss_function(intercept, slope, targets, features):
    predictions = linear_regression(intercept, slope, features)
    return tf.keras.losses.mse(targets, predictions)
```

[42]:
```
# Define optimization operation
opt = tf.keras.optimizers.Adam()
```

[43]:
```
# Load the data in batches from pandas
for batch in pd.read_csv('ref1. King county house sales.csv', chunksize=100):
    # Extract the target and feature columns
    price_batch = np.array(batch['price'], np.float32)
    size_batch = np.array(batch['sqft_lot'], np.float32)

    # Minimize the loss function
    opt.minimize(
        lambda: loss_function(intercept, slope, price_batch, size_batch),
        var_list=[intercept, slope])
```

[44]:
```
# Print parameter values
print(intercept.numpy(), slope.numpy())
```

```
0.31781912 0.29831016
```

4.5 Compare full sample versus batch training, what are the differences?

- Full Sample
    1. One update per epoch
    2. Accepts dataset without modification
    3. Limited by memory
- Batch Training
    1. Multiple updates per epoch
    2. Requires the division of dataset
    3. No limit on dataset size

4.6 Practice exercises for batch training:

▶ **Package pre-loading:**

[45]:
```
from tensorflow import Variable, keras, float32
```

▶ **Batch train preparing practice:**

[46]:
```python
# Define the intercept and slope
intercept = Variable(10.0, float32)
slope = Variable(0.5, float32)


# Define the model
def linear_regression(intercept, slope, features):
    # Define the predicted values
    return intercept + features * slope


# Define the loss function
def loss_function(intercept, slope, targets, features):
    # Define the predicted values
    predictions = linear_regression(intercept, slope, features)

    # Define the MSE loss
    return keras.losses.mse(targets, predictions)
```

▶ **Package re-pre-loading:**

[47]:
```python
import pandas as pd
import numpy as np
```

▶ **Linear model batches training practice:**

[48]:
```python
# Initialize adam optimizer
opt = keras.optimizers.Adam()

# Load data in batches
for batch in pd.read_csv('ref1. King county house sales.csv', chunksize=100):
    size_batch = np.array(batch['sqft_lot'], np.float32)

    # Extract the price values for the current batch
    price_batch = np.array(batch['price'], np.float32)

    # Complete the loss, fill in the variable list, and minimize
    opt.minimize(
        lambda: loss_function(intercept, slope, price_batch, size_batch),
        var_list=[intercept, slope])

# Print trained parameters
print(intercept.numpy(), slope.numpy())
```

```
10.217888 0.7016
```