Machine Learning with TensorFlow

Industrial AI Lab.

Prof. Seungchul Lee

Training Neural Networks: Deep Learning Libraries

- TensorFlow
  - Platform: Linux, Mac OS, Windows
  - Written in: C++, Python
  - Interface: Python, C/C++, Java, Go, R

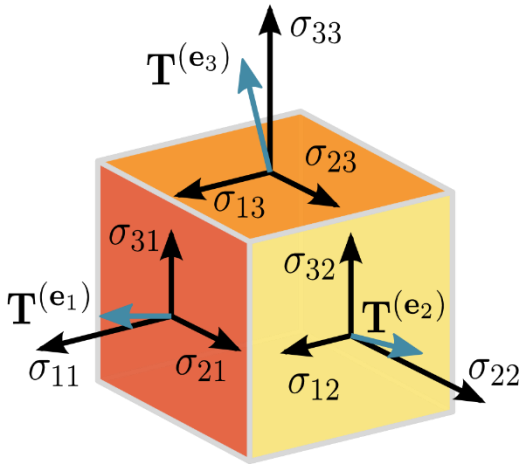
- Keras

- PyTorch

TensorFlow

- Developed by Google and it is one of the most popular Machine Learning libraries on GitHub.
- It is a framework to perform computation very efficiently, and it can tap into the GPU in order to speed it up even further.
- TensorFlow is one of the widely used libraries for implementing machine learning and deep learning involving large number of mathematical operations.

- Tensor and Flow
  - TensorFlow gets its name from tensors, which are arrays of arbitrary dimensionality.
  - The "flow" part of the name refers to computation flowing through a graph.



Computational Graph

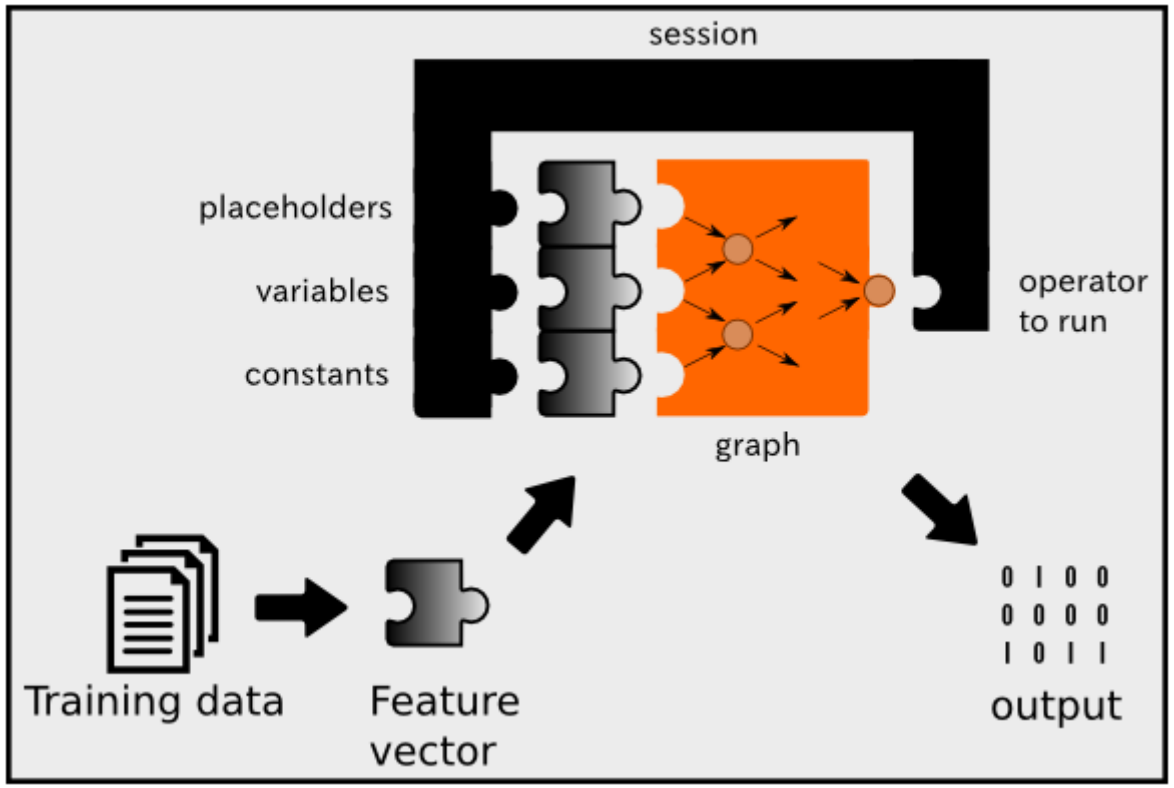
- TensorFlow is an open-source software library for deep learning
  - tf.constant
  - tf.Variable
  - tf.placeholder

```
import tensorflow as tf

a = tf.constant([1,2,3])
b = tf.constant(4, shape=[1,3])

A = a + b
B = a*b

print(A)
```

```
Tensor("add_1:0", shape=(1, 3), dtype=int32)
```

TensorFlow: Session

- To run any of the three defined operations, we need to create a session for that graph. The session will also allocate memory to store the current value of the variable.



TensorFlow

```
import tensorflow as tf

a = tf.constant([1,2,3])
b = tf.constant(4, shape=[1,3])

A = a + b
B = a*b

print(A)
```

```
sess = tf.Session()
sess.run(A)
```

```
array([[5, 6, 7]])
```

```
sess.run(B)
```

```
array([[ 4,  8, 12]])
```

```
a = tf.constant([1,2,3])
b = tf.constant([4,5,6])

result = tf.multiply(a, b)

with tf.Session() as sess:
    output = sess.run(result)
    print(output)
```

← Interactive Session:
run the result and close the Session automatically

TensorFlow: tf.Variable

- tf.Variable is regarded as the decision variable in optimization.
- We should initialize variables.

```
x1 = tf.Variable([1, 1], dtype = tf.float32)
x2 = tf.Variable([2, 2], dtype = tf.float32)
y = x1 + x2

print(y)
```

```
<tf.Tensor 'add_8:0' shape=(2,) dtype=float32>
```

```
sess = tf.Session()

init = tf.global_variables_initializer()
sess.run(init)

sess.run(y)
```

```
array([ 3.,  3.], dtype=float32)
```

TensorFlow: Placeholder

- The value of tf.placeholder must be fed using the feed_dict optional argument to Session.run()

```
sess = tf.Session()
x = tf.placeholder(tf.float32, shape = [2,2])

sess.run(x, feed_dict = {x : [[1,2],[3,4]]})
```

```
array([[ 1.,  2.],
       [ 3.,  4.]], dtype=float32)
```

```
a = tf.placeholder(tf.float32, shape = [2])
b = tf.placeholder(tf.float32, shape = [2])

sum = a + b

sess.run(sum, feed_dict = {a : [1,2], b : [3,4]})
```

```
array([ 4.,  6.], dtype=float32)
```

Tensor Manipulation: Adding

```
x1 = tf.constant(1, shape = [3])
x2 = tf.constant(2, shape = [3])
output = tf.add(x1, x2)

with tf.Session() as sess:
    result = sess.run(output)
    print(result)
```

```
[3 3 3]
```

```
x1 = tf.constant(1, shape = [2, 3])
x2 = tf.constant(2, shape = [2, 3])
output = tf.add(x1, x2)

with tf.Session() as sess:
    result = sess.run(output)
    print(result)
```

```
[[3 3 3]
 [3 3 3]]
```

Tensor Manipulation: Multiplying

```
x1 = tf.constant([[1, 2],
                  [3, 4]])
x2 = tf.constant([[2],[3]])
```

```
output1 = tf.matmul(x1, x2)

with tf.Session() as sess:
    result = sess.run(output1)
    print(result)
```

```
[[ 8]
 [18]]
```

```
output2 = x1*x2

with tf.Session() as sess:
    result = sess.run(output2)
    print(result)
```

```
[[ 2  4]
 [ 9 12]]
```

Tensor Manipulation: Reshape

```
x = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
x_re = tf.reshape(x, [4,2])

sess = tf.Session()
sess.run(x_re)
```

```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

```
x_re = tf.reshape(x, [2,-1])

sess = tf.Session()
sess.run(x_re)
```

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

TensorFlow as an Optimization Solver

```
w = tf.Variable(0, dtype = tf.float32)
cost =  w*w - 8*w + 16

LR = 0.05
optm = tf.train.GradientDescentOptimizer(LR).minimize(cost)

init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)

print(sess.run(w))
```

```
0.0
```

$$\min_{\omega} \ (\omega - 4)^2$$

```
# runs one step of gradient descent
sess.run(optm)
print(sess.run(w))

# runs two step of gradient descent
sess.run(optm)
print(sess.run(w))
```

```
0.4
0.76
```

```
for _ in range(100):
    sess.run(optm)

print(sess.run(w))
sess.close()
```

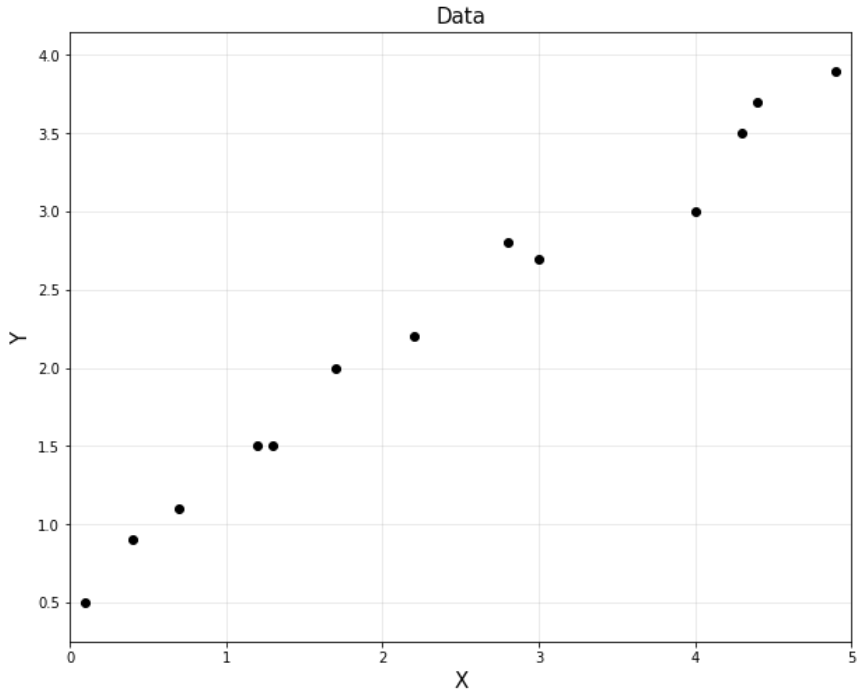
```
3.99991
```

Machine Learning with TensorFlow

Regression

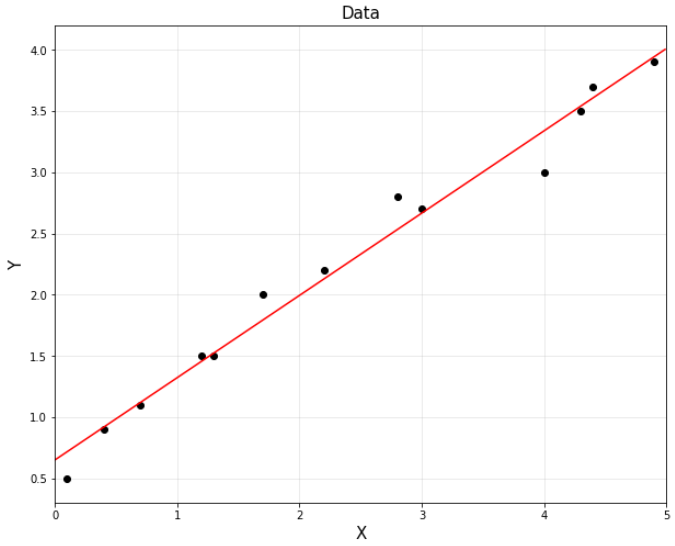
- Given (x_i, y_i) for $i = 1, \cdots, m$,
- Want to estimate

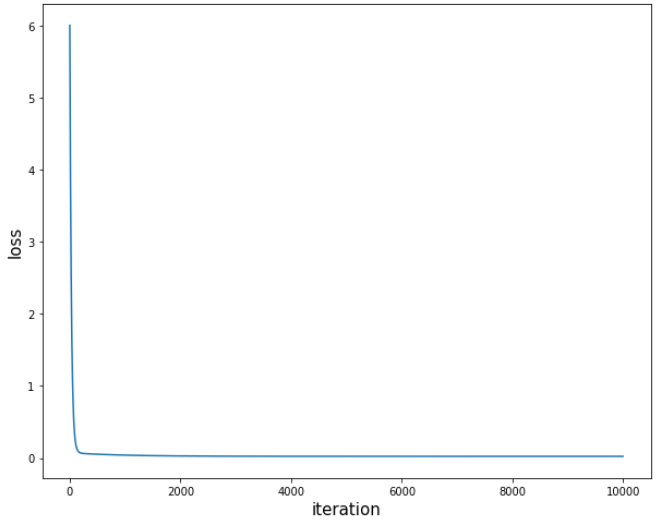
$$\hat{y}_i = \omega x_i + b \quad \text{such that} \quad \min_{\omega,b} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$



Regression with TensorFlow

```
LR = 0.001
n_iter = 10000

x = tf.placeholder(tf.float32, [m, 1])
y = tf.placeholder(tf.float32, [m, 1])

w = tf.Variable([[0]], dtype = tf.float32)
b = tf.Variable([[0]], dtype = tf.float32)

#y_pred = tf.matmul(x, w) + b
y_pred = tf.add(tf.matmul(x, w), b)
loss = tf.square(y_pred - y)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

loss_record = []
for epoch in range(n_iter):
    _, c = sess.run([optm, loss], feed_dict = {x: train_x, y: train_y})
    loss_record.append(c)

w_val = sess.run(w)
b_val = sess.run(b)

sess.close()
```

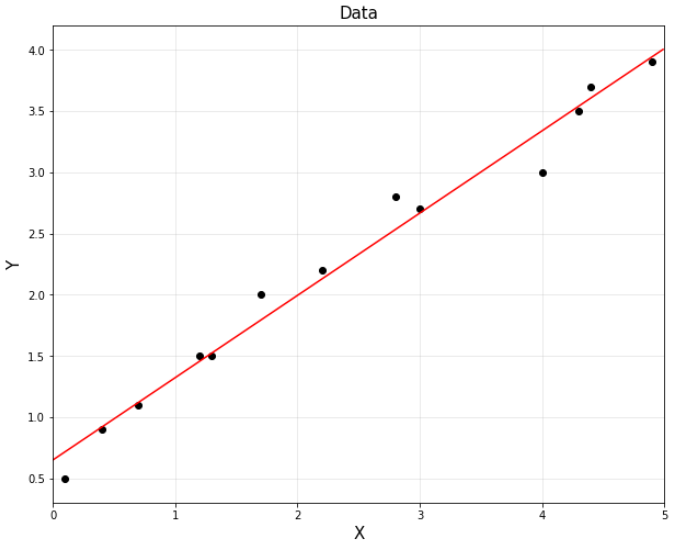


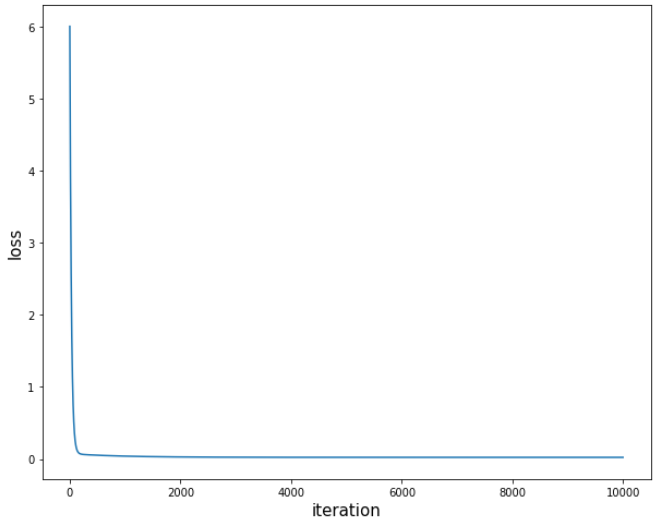


Regression with TensorFlow

- with tf.Session() as sess:

```
LR = 0.001
n_iter = 10000

x = tf.placeholder(tf.float32, [m, 1])
y = tf.placeholder(tf.float32, [m, 1])

w = tf.Variable([[0]], dtype = tf.float32)
b = tf.Variable([[0]], dtype = tf.float32)

#y_pred = tf.matmul(x, w) + b
y_pred = tf.add(tf.matmul(x, w), b)
loss = tf.square(y_pred - y)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(n_iter):
        sess.run(optm, feed_dict = {x: train_x, y: train_y})
    w_val = sess.run(w)
    b_val = sess.run(b)
```

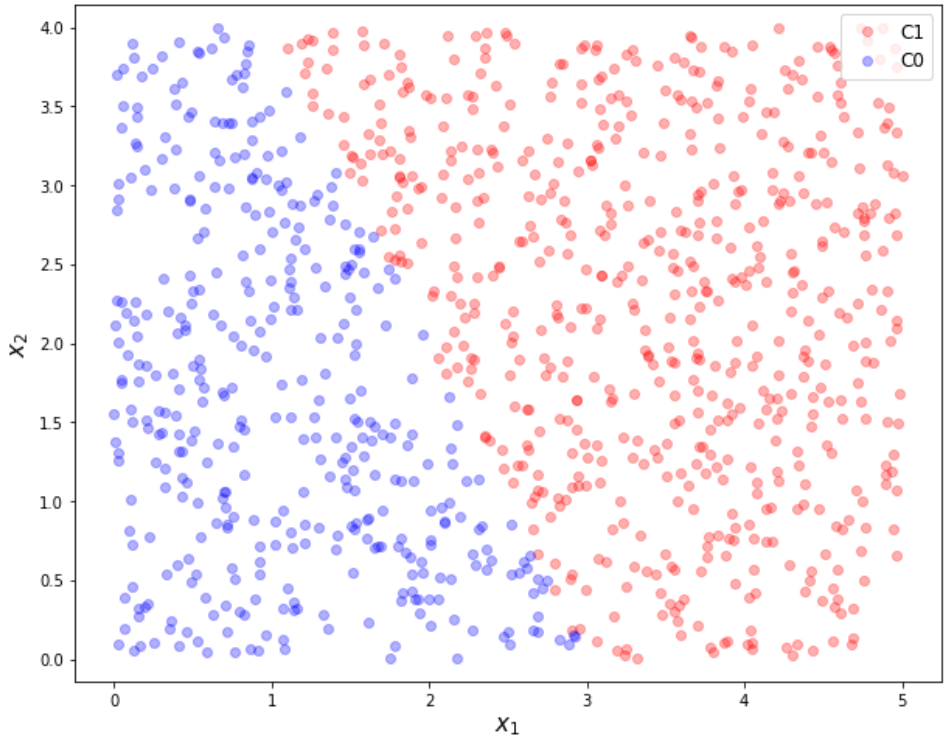




Logistic Regression

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots & \vdots \end{bmatrix}, \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \end{bmatrix}$$

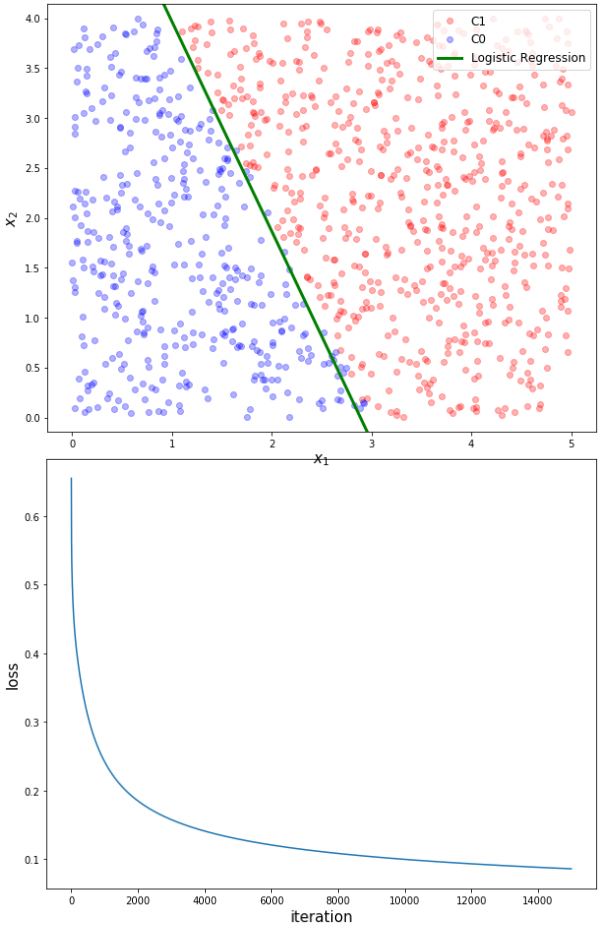


Logistic Regression with TensorFlow

$$\ell(\omega) = \log \mathscr{L}(\omega) = \sum_{i=1}^{m} y^{(i)} \log h_\omega \left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\omega \left(x^{(i)}\right)\right)$$

$$\Rightarrow \frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log h_\omega \left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\omega \left(x^{(i)}\right)\right)$$

```
LR = 0.05
n_iter = 15000

X = tf.placeholder(tf.float32, [m, 3])
y = tf.placeholder(tf.float32, [m, 1])

w = tf.Variable([[0],[0],[0]], dtype = tf.float32)

y_pred = tf.sigmoid(tf.matmul(X,w))
loss = - y*tf.log(y_pred) - (1-y)*tf.log(1-y_pred)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)

loss_record = []
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(n_iter):
        _, c = sess.run([optm, loss], feed_dict = {X: train_X, y: train_y})
        loss_record.append(c)

    w_hat = sess.run(w)
```



Logistic Regression with TensorFlow

- TensorFlow embedded functions
  - tf.nn.sigmoid_cross_entropy_with_logits for binary classification
  - tf.nn.softmax_cross_entropy_with_logits for multiclass classification

```
LR = 0.05
n_iter = 30000

X = tf.placeholder(tf.float32, [m, 3])
y = tf.placeholder(tf.float32, [m, 1])

w = tf.Variable(tf.random_normal([3,1]), dtype = tf.float32)

y_pred = tf.matmul(X,w)
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits = y_pred, labels = y)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(n_iter):
        sess.run(optm, feed_dict = {X: train_X, y: train_y})

    w_hat = sess.run(w)
```