Tensorflow Basic (2/2)

Optimization basic

- Training for prediction
  - Y : a set target variable
  - X : a set of feature vector to explain Y

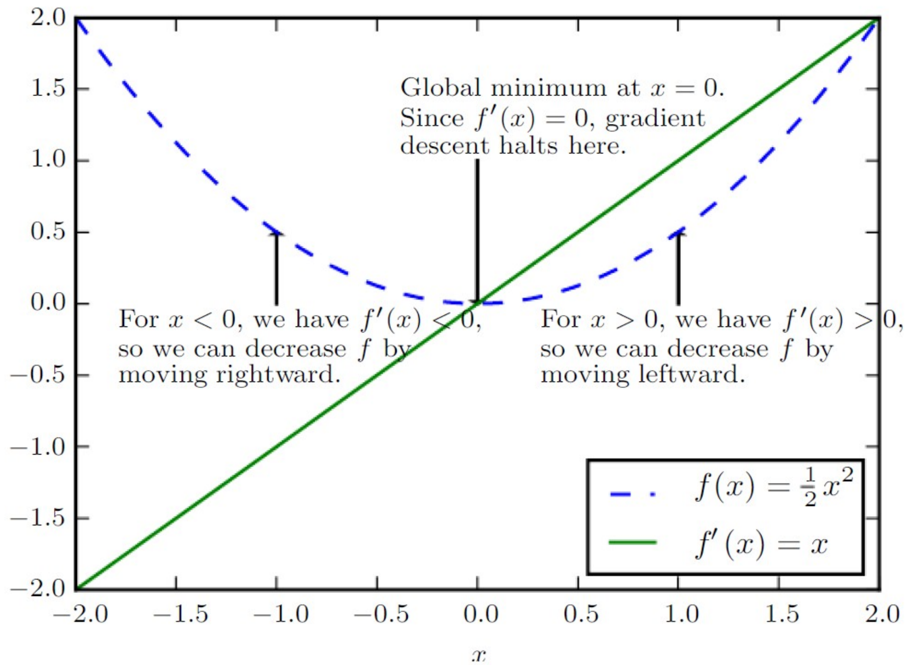
- Model selection to best capture the desired relation of X and Y
  - Linear regression, logistic classification, deep neural network, …
  - Our training data points will be used for tuning the model

Optimization basic

- Defining a loss function
  - A good measure is required to evaluate the model’s performance.
  - Measure the discrepancy or distance (loss) between the model’s predictions an d the observed targets.
  - The goal of optimization is to find the set of parameters (weight, biases, …) of the model that minimize the distance.
  - Mean squared $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$
  - Cross entropy $$H(p,q) = -\sum_{x} p(x)\,\log q(x)$$

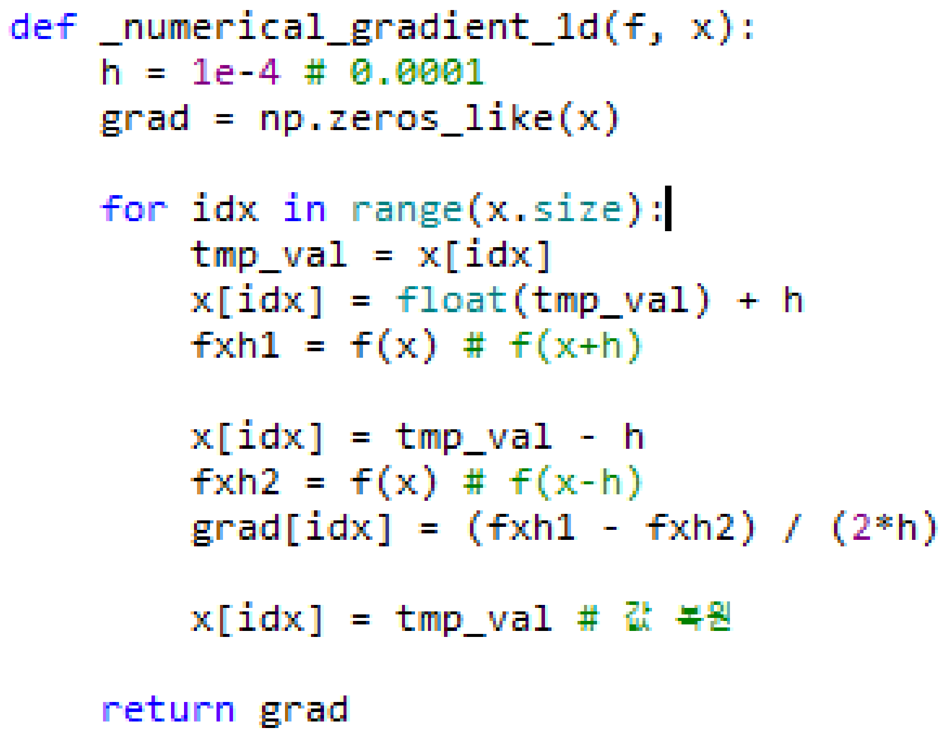
Optimization basic

- Gradient descent: Finding the set of X that minimizes or F(X).
- Derivative : $$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$
- Partial derivative, ∇
  - $y = f(x_1, x_2, x_3, \ldots, x_n)$, $\nabla f = (f_{x_1}, f_{x_2}, f_{x_3}, \ldots, f_{x_n})$
- Gradient descent
  - $x' = x - \varepsilon * \nabla f$ ε: learning rate



Optimization basic

* Gradient descent optimizer

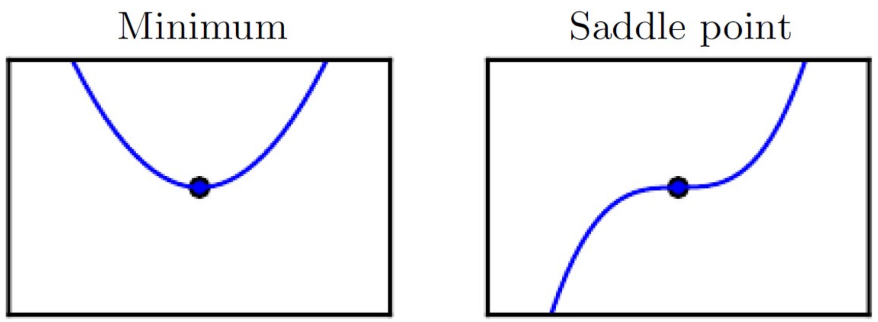


```python
def _numerical_gradient_1d(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)

        x[idx] = tmp_val # 값 복원

    return grad
```

f: loss function
x: parameters to tune

Iteratively update the set of parameters in a way that decrease the loss over time.

But the point we find could be a global minimum or a saddle point.
⇒ Second derivative



Optimization basic

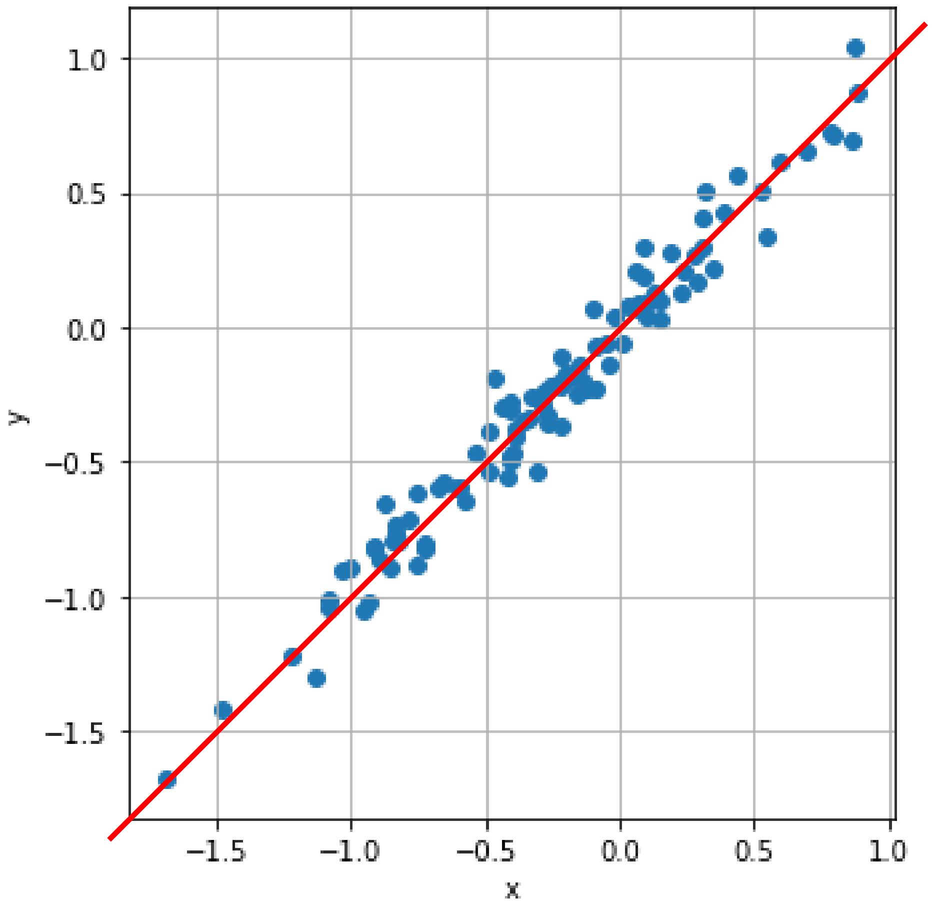
- Stochastic gradient descent
  - Instead of feeding the entire dataset to the algorithm for the computation of each iteration, a subse t of the data is sampled (*mini-batch*) and fed.

  - Using smaller batches usually works faster and the smaller the size of the batch, the faster are the calculations. However, there is a trade-off in that small samples lead to lower hardware utilization and tend t have high variance, causing large fluctuations of the objective function.

  - Nevertheless, it turns out that some fluctuations are beneficial since they enable the set of paramet ers to jump to new and potentially better local minima.

  - Using a relatively small batches is preferred recently.

Linear regression



Given: $X = \{x_1, x_2, \ldots, x_m\}$, $Y = \{y_1, y_2, \ldots, y_m\}$

Assumption: $y_i \approx \hat{y}_i = a_1 * x_{i,1} + a_2 * x_{i,2} + \ldots + a_m * x_{i,m} + b$

Goal: Find $a_1, a_2, \ldots, a_m, b$ that minimize

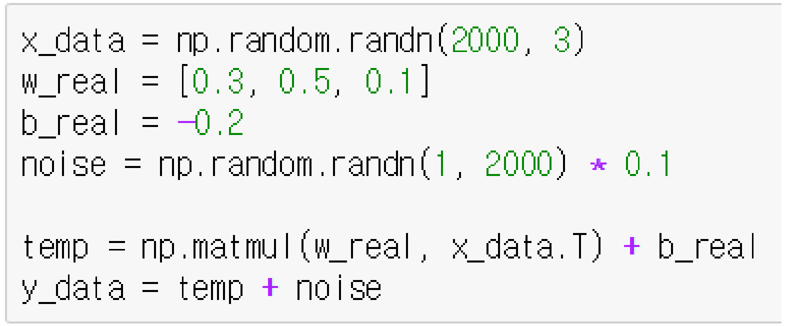
$\hat{Y} = \Phi * \theta$, where $\Phi = [x_1, x_2, ..., x_m, 1]$, $\theta = [a_1, a_2, \ldots, a_m, b]^T$

$$\hat{Y} = \begin{bmatrix} [x_{1,1}, x_{1,2}, ..., x_{1,m}, 1] \\ [x_{2,1}, x_{2,2}, ..., x_{2,m}, 1] \\ ... \\ [x_{n,1}, x_{n,2}, ..., x_{n,m}, 1] \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ \ldots \\ a_m \\ b \end{bmatrix} = \Phi * \theta$$

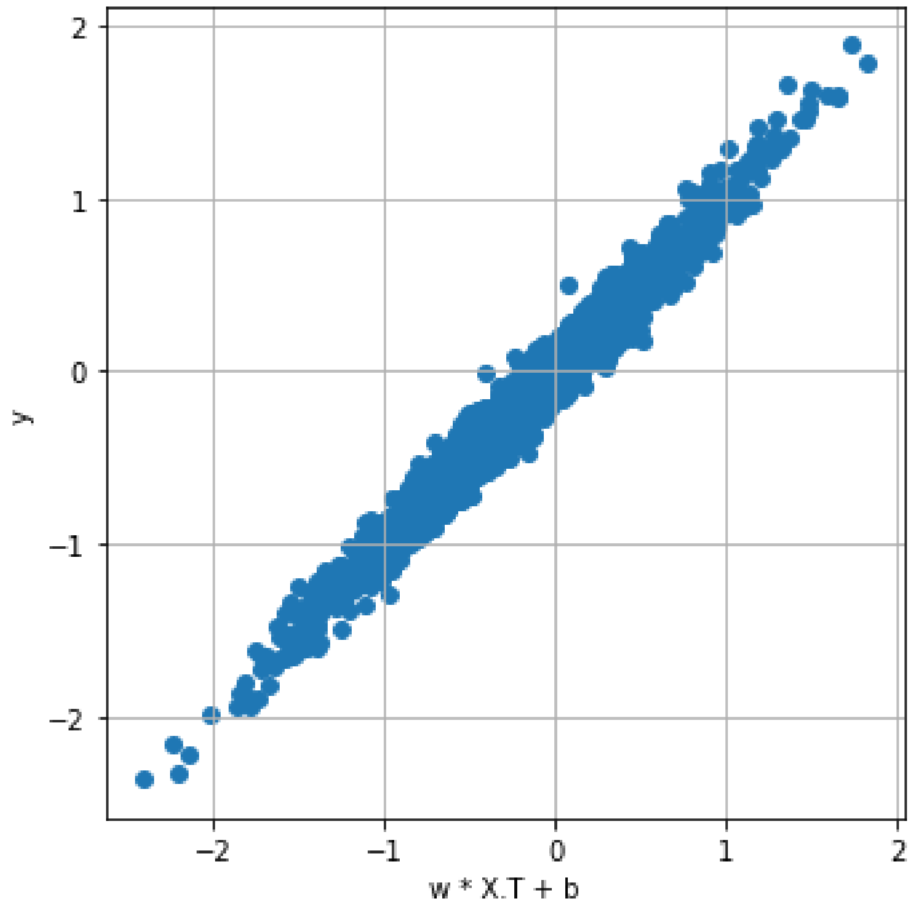
Linear regression with Tensorflow

1. Data generation

- y: target variable
- x: feature vector
- y = x * W + b

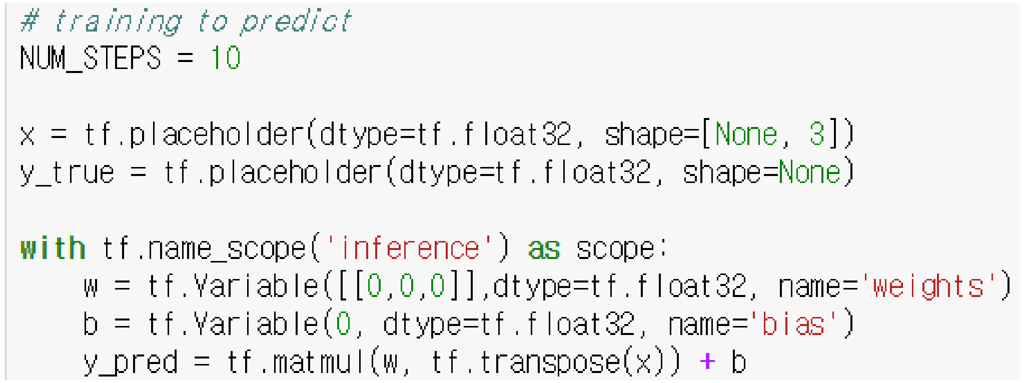


```
x_data = np.random.randn(2000, 3)
w_real = [0.3, 0.5, 0.1]
b_real = -0.2
noise = np.random.randn(1, 2000) * 0.1

temp = np.matmul(w_real, x_data.T) + b_real
y_data = temp + noise
```



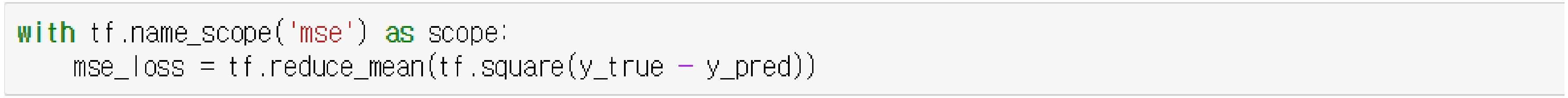
Linear regression with Tensorflow

2. Model definition

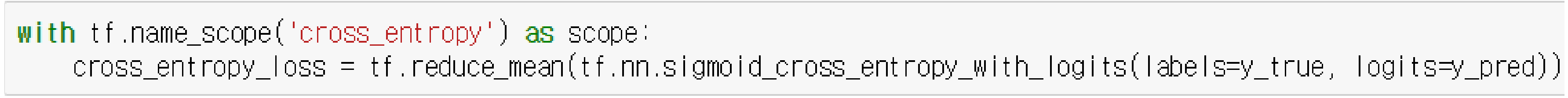


```
# training to predict
NUM_STEPS = 10

x = tf.placeholder(dtype=tf.float32, shape=[None, 3])
y_true = tf.placeholder(dtype=tf.float32, shape=None)

with tf.name_scope('inference') as scope:
    w = tf.Variable([[0,0,0]],dtype=tf.float32, name='weights')
    b = tf.Variable(0, dtype=tf.float32, name='bias')
    y_pred = tf.matmul(w, tf.transpose(x)) + b
```

3. Loss function definition



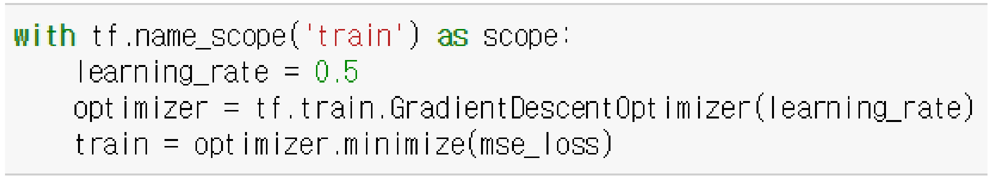
```
with tf.name_scope('mse') as scope:
    mse_loss = tf.reduce_mean(tf.square(y_true - y_pred))
```



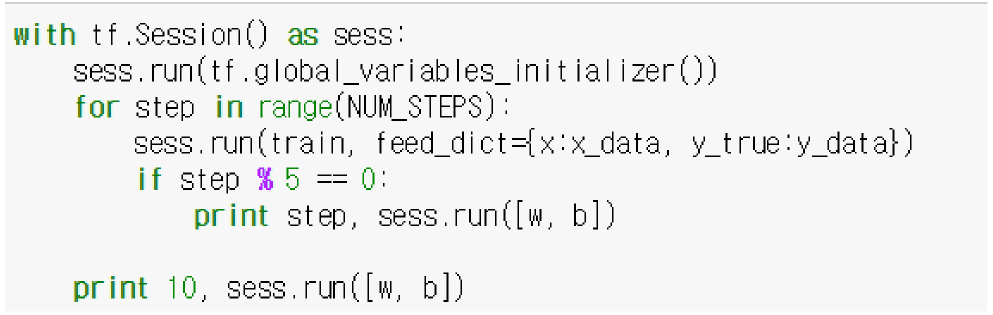
```
with tf.name_scope('cross_entropy') as scope:
    cross_entropy_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=y_true, logits=y_pred))
```

Linear regression with Tensorflow

4. Training

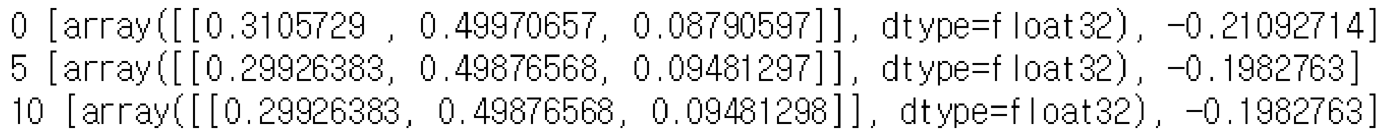


```
with tf.name_scope('train') as scope:
    learning_rate = 0.5
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train = optimizer.minimize(mse_loss)
```



```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(NUM_STEPS):
        sess.run(train, feed_dict={x:x_data, y_true:y_data})
        if step % 5 == 0:
            print step, sess.run([w, b])

    print 10, sess.run([w, b])
```

5. Result



```
0 [array([[0.3105729 , 0.49970657, 0.08790597]], dtype=float32), -0.21092714]
5 [array([[0.29926383, 0.49876568, 0.09481297]], dtype=float32), -0.1982763]
10 [array([[0.29926383, 0.49876568, 0.09481298]], dtype=float32), -0.1982763]
```