PyTorch Tutorial

02. Linear Model

Machine learning

- Suppose that students would get ***y*** points in final exam, if they spent ***x*** hours in paper *PyTorch Tutorial*.

x (hours)	y (points)
1	2
2	4
3	6
4	?

- The question is what would be the grade if I study 4 hours?

Machine Learning

4 hours

? points

Prediction

x (hours)	y (points)
1	2
2	4
3	6
4	?

Machine Learning

4 hours →  → ? points

Prediction

x (hours)	y (points)
1	2
2	4
3	6
4	?

Training Set

Test Set

Supervised Learning

Model design

- What would be the best model for the data?
- Linear model?

x (hours)	y (points)
1	2
2	4
3	6
4	?

Linear Model

$$\hat{y} = x * \omega + b$$

Model design

- What would be the best model for the data?
- Linear model?

x (hours)	y (points)
1	2
2	4
3	6
4	?

Linear Model

$$\hat{y} = x * \omega$$

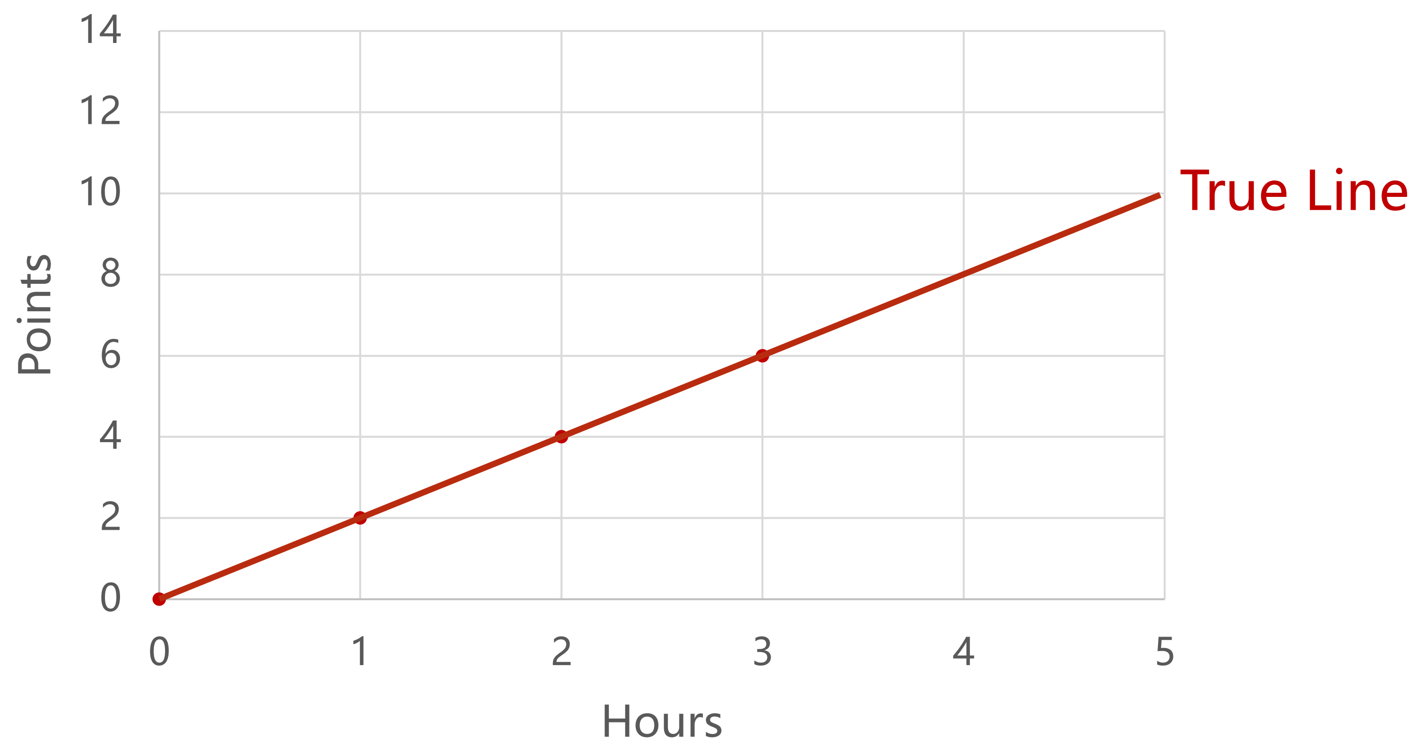
To simplify the model

Linear Regression

Linear Model

$$\hat{y} = x * \omega$$

x (hours)	y (points)
1	2
2	4
3	6

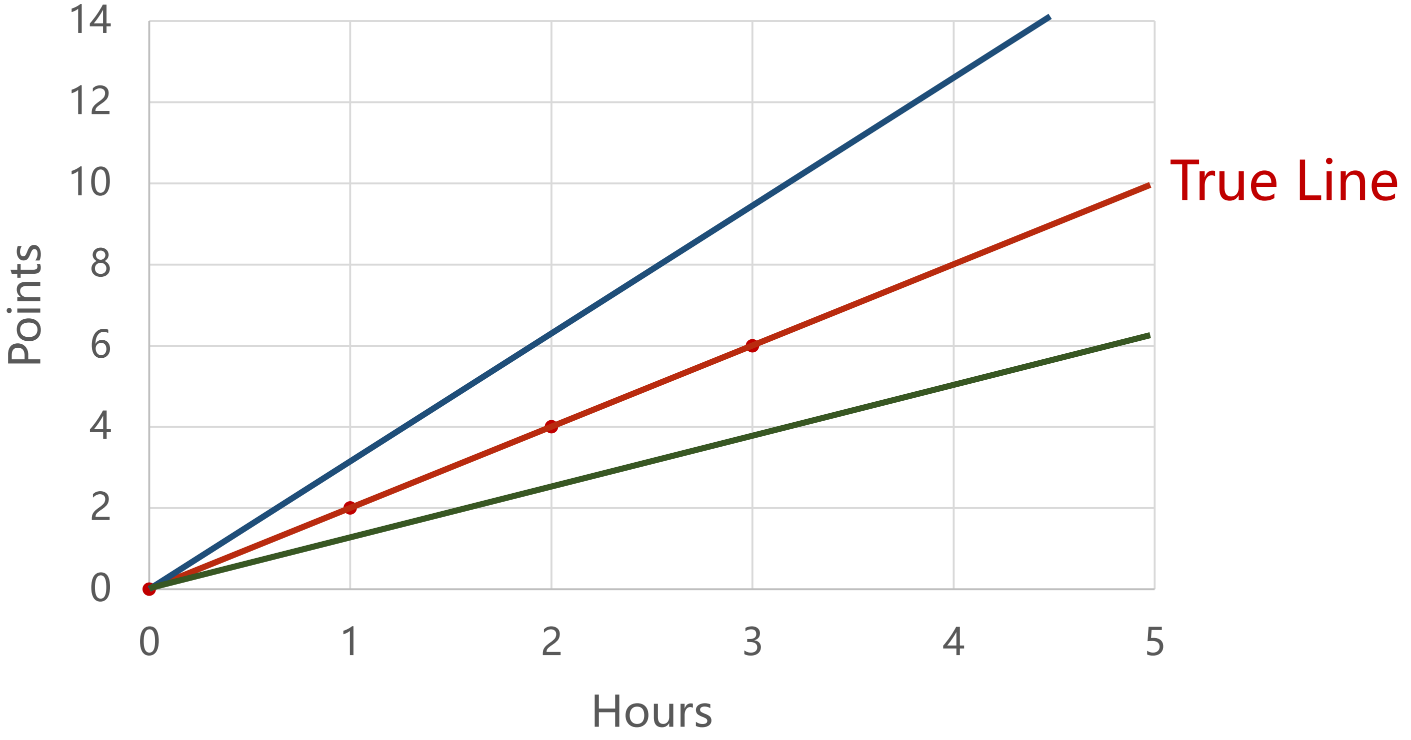


Linear Model

$$\hat{y} = x * \omega$$

x (hours)	y (points)
1	2
2	4
3	6

The machine starts with **a random guess**, ω = random value



Compute Loss

Training Loss (Error)

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

x (Hours)	y (Points)	y_predict (w=3)	Loss (w=3)
1	2	3	1
2	4	6	4
3	6	9	9
			mean = 14/3

Compute Loss

Training Loss (Error)

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

x (Hours)	y (Points)	y_predict (w=4)	Loss (w=4)
1	2	4	4
2	4	8	16
3	6	12	36
			mean = 56/3

Compute Loss

Training Loss (Error)

$$loss = (\hat{y} - y)^2 = (x * \omega \ - y)^2$$

x (Hours)	y (Points)	y_predict (w=0)	Loss (w=0)
1	2	0	4
2	4	0	16
3	6	0	36
			mean = 56/3

Compute Loss

Training Loss (Error)

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

x (Hours)	y (Points)	y_predict (w=1)	Loss (w=1)
1	2	1	1
2	4	2	4
3	6	3	9
			mean = 14/3

Compute Loss

Training Loss (Error)

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

x (Hours)	y (Points)	y_predict (w=2)	Loss (w=2)
1	2	2	0
2	4	4	0
3	6	6	0
			mean = 0

Loss function & Cost function

Training Loss (Error)

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

Mean Square Error

$$cost = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

Compute Cost

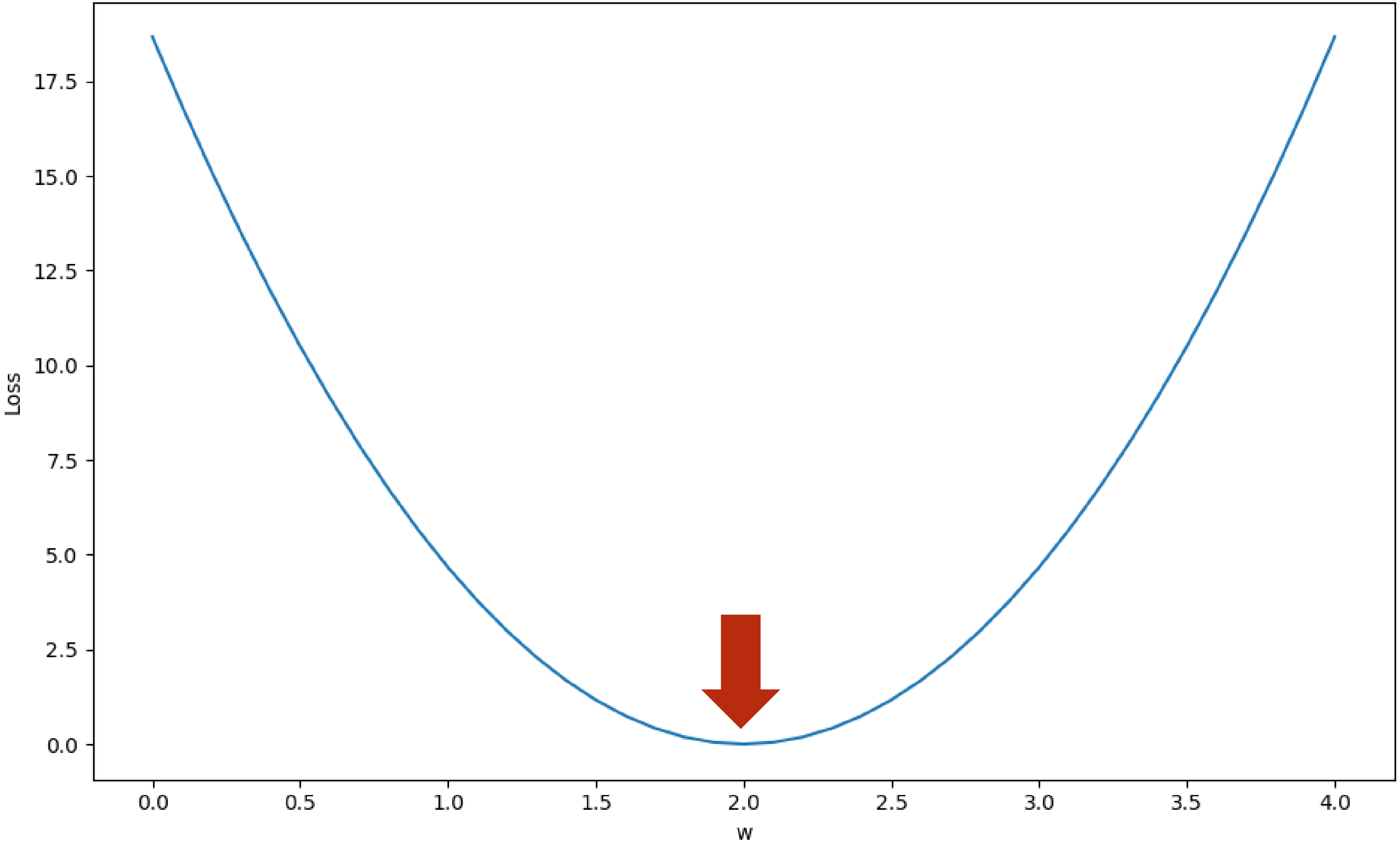
Mean Square Error

$$cost = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

x (Hours)	Loss (w=0)	Loss (w=1)	Loss (w=2)	Loss (w=3)	Loss (w=4)
1	4	1	0	1	4
2	16	4	0	4	16
3	36	9	0	9	36
MSE	18.7	4.7	0	4.7	18.7

Linear Regression

It can be found that when $\omega = 2$, the cost will be minimal.



How to draw the graph

```
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```


```
import numpy as np
import matplotlib.pyplot as plt
```


Import necessary library to draw the graph.

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]
```

Prepare the train set.

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
def forward(x):
    return x * w
```

Define the model:

Linear Model

$$\hat{y} = x * \omega$$

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)
```

Define the loss function:

Loss Function

$$loss = (\hat{y} - y)^2 = (x * \omega - y)^2$$

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
w_list = []
mse_list = []
```

List *w_list* save the weights ω.
List *mse_list* save the cost values of each ω.

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
for w in np.arange(0.0, 4.1, 0.1):
```

Compute cost value at [0.0, 0.1, 0.2, … , 4.0]

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
for x_val, y_val in zip(x_data, y_data):
    y_pred_val = forward(x_val)
    loss_val = loss(x_val, y_val)
    l_sum += loss_val
    print('\t', x_val, y_val, y_pred_val, loss_val)
```

For each sample in train set, the loss function values were computed.

ATTENTION:
Value of cost function is the sum of loss function.

How to draw the graph

```python
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

```python
w_list.append(w)
mse_list.append(l_sum / 3)
```

Save ω and correspondence **MSE**.

How to draw the graph

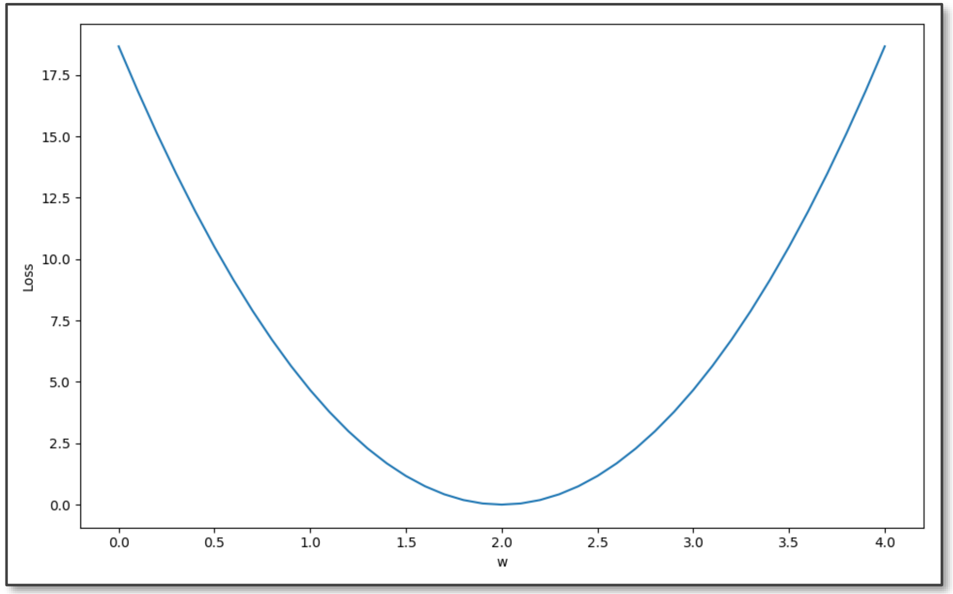
```
import numpy as np
import matplotlib.pyplot as plt

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

w_list = []
mse_list = []
for w in np.arange(0.0, 4.1, 0.1):
    print('w=', w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
        y_pred_val = forward(x_val)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
        print('\t', x_val, y_val, y_pred_val, loss_val)
    print('MSE=', l_sum / 3)
    w_list.append(w)
    mse_list.append(l_sum / 3)
```

Part of result

```
w= 0.0
        1.00  2.00  0.00  4.00
        2.00  4.00  0.00 16.00
        3.00  6.00  0.00 36.00
MSE= 18.666666666666668
w= 0.1
        1.00  2.00  0.10  3.61
        2.00  4.00  0.20 14.44
        3.00  6.00  0.30 32.49
MSE= 16.846666666666668
w= 0.2
        1.00  2.00  0.20  3.24
        2.00  4.00  0.40 12.96
        3.00  6.00  0.60 29.16
MSE= 15.120000000000003
w= 0.30000000000000004
        1.00  2.00  0.30  2.89
        2.00  4.00  0.60 11.56
        3.00  6.00  0.90 26.01
MSE= 13.486666666666665
w= 0.4
        1.00  2.00  0.40  2.56
        2.00  4.00  0.80 10.24
        3.00  6.00  1.20 23.04
MSE= 11.946666666666667
w= 0.5
        1.00  2.00  0.50  2.25
        2.00  4.00  1.00  9.00
        3.00  6.00  1.50 20.25
MSE= 10.5
```

Draw the graph

```
plt.plot(w_list, mse_list)
plt.ylabel('Loss')
plt.xlabel('w')
plt.show()
```

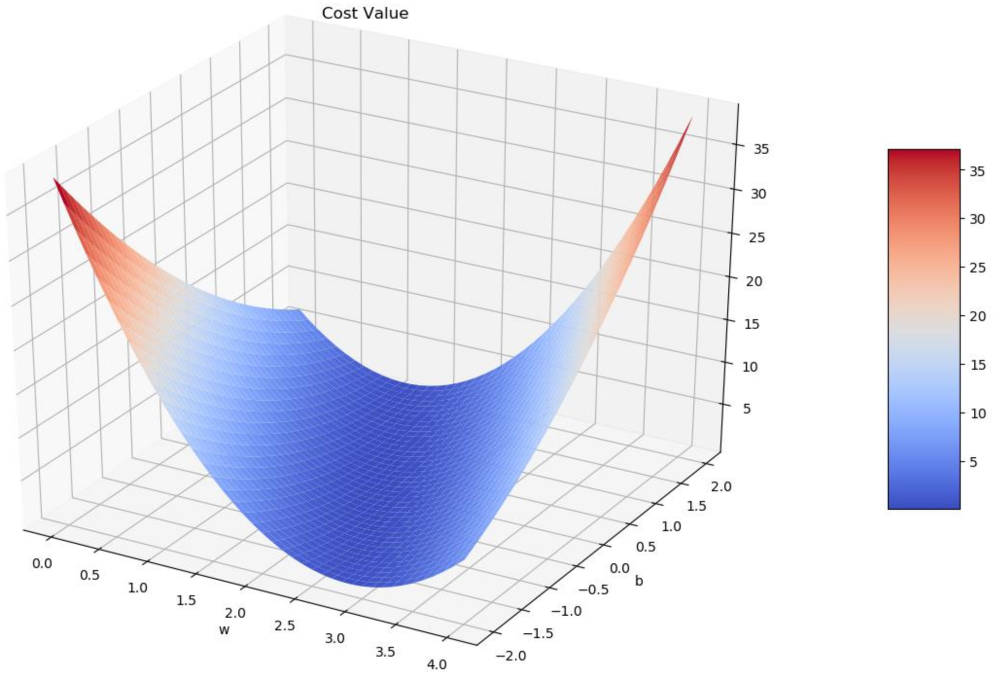


- Try to use the model in right-side, and draw the cost graph.

Linear Model

$$\hat{y} = x * \omega + b$$

- Tips:
  - You can read the material of how to draw 3d graph. [link]
  - Function *np.meshgrid()* is very popular for drawing 3d graph, read the [docs] and utilize vectorization calculation.



PyTorch Tutorial

02. Linear Model