Single neuron with one input data

1. 1. Understand MLP .ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

$x * W0T + b = (x1, x2, x3)(w1, w2, w3)T + b$

```
[9]  tensorX.mm(W0t)+MyNet[0].bias

     tensor([[0.2655]], grad_fn=<AddBackward0>)
```

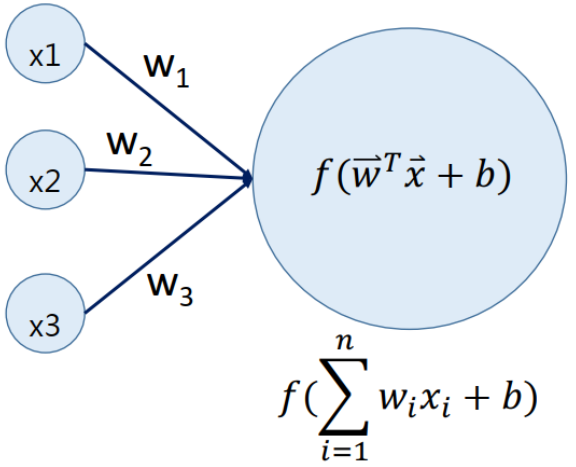
$Relu(x * W0T + b) = Relu((x1, x2, x3)(w1, w2, w3)T + b)$

```
[10] relu(tensorX.mm(W0t)+MyNet[0].bias)

     tensor([[0.2655]], grad_fn=<ReluBackward0>)
```

2. Using Excel to verify

$Relu(w1 * x1 + w2 * x2 + w3 * x3 + b)$

$$\begin{matrix} x1 & x2 & x3 \\ [1 & 2 & 3] \end{matrix}$$



	Weight
w1	0.5398
w2	-0.5377
w3	0.3111
bias	-0.1321

	Input
x1	1
x2	2
x3	3

	Weight x Input
	0.5398
	-1.0754
	0.9333
Total	0.3977
Total +Bias	0.26560

Reference: Dominico Laksma Paramestha 拉斯麥

Single neuron with multiple input data

1. 1. Understand MLP .ipynb

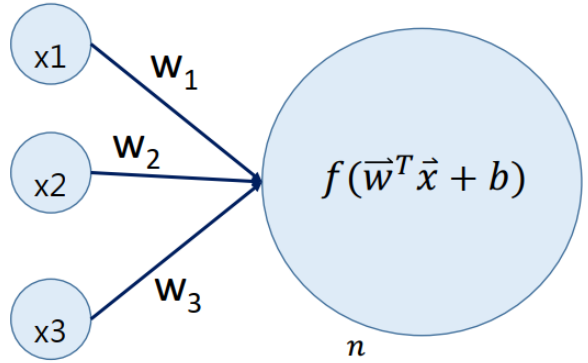
File Edit View Insert Runtime Tools Help Cannot sav

+ Code + Text Copy to Drive

```
[35] tensor([[1.9804],
             [4.5449],
             [7.1093]], grad_fn=<MmBackward0>)

[36] tensorX.mm(W0t)+MyNet[0].bias

     tensor([[2.3323],
             [4.8967],
             [7.4612]], grad_fn=<AddBackward0>)

[37] relu = nn.ReLU()

[38] relu(tensorX.mm(W0t)+MyNet[0].bias)

     tensor([[2.3323],
             [4.8967],
             [7.4612]], grad_fn=<ReluBackward0>)
```

2. Using Excel to verify

$$\begin{matrix} x1 & x2 & x3 \end{matrix}$$
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



$$f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Input	x1	x2	x3
	1	2	3
	4	5	6
	7	8	9

Weight	
w1	0.2118
w2	0.1606
w3	0.4825
Bias	0.3519

Input x Weight
1.9805
4.5452
7.1099

+ bias
2.3324
4.8971
7.4618

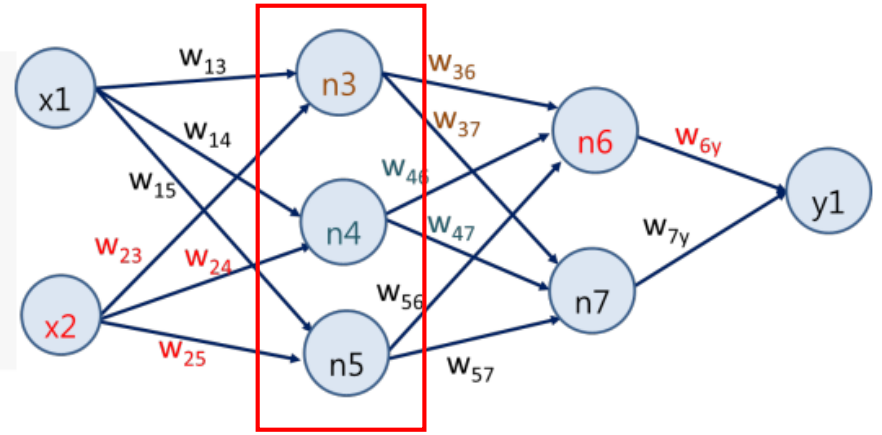
Reference: Dominico Laksma Paramestha 拉斯麥

Calculate n3, n4, n5 values (1st layer)

```
[5]  W0 = MyNet[0].weight
     b0 = MyNet[0].bias
     print(W0, W0.shape, b0)

     Parameter containing:
     tensor([[ 0.5339, -0.1518],
             [ 0.4930,  0.3742],
             [-0.2708, -0.6664]], requires_grad=True) torch.Size([3, 2]) Parameter containing:
     tensor([-0.7071,  0.5259, -0.1280], requires_grad=True)
```

```
[6]  #Calculate n3, n4, n5 using Pytorch matrix operation
     Layer1_1 = tensorX.mm(torch.transpose(W0, 1, 0)) + b0
     print(Layer1_1)

     tensor([[-0.4767,  1.7673, -1.7317],
             [-0.0946,  2.6345, -2.6689],
             [ 3.8733,  7.3268, -6.1684]], grad_fn=<AddBackward0>)
```

```
[7]  #Verify
     Layer1 = MyNet[0](tensorX)
     print(Layer1)

     tensor([[-0.4767,  1.7673, -1.7317],
             [-0.0946,  2.6345, -2.6689],
             [ 3.8733,  7.3268, -6.1684]], grad_fn=<AddmmBackward0>)
```



$$\begin{bmatrix} x_1^1 & x_2^1 \\ x_1^2 & x_2^2 \\ x_1^3 & x_2^3 \end{bmatrix} \begin{bmatrix} w_{13} & w_{14} & w_{15} \\ w_{23} & w_{24} & w_{25} \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} k_3^1 & k_4^1 & k_5^1 \\ k_3^2 & k_4^2 & k_5^2 \\ k_3^3 & k_4^3 & k_5^3 \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix}$$

Input x		
	1	2
	2	3
	10	5

Weight			
	0.5339	0.493	-0.2708
	-0.1518	0.3742	-0.6664

Bias			
	-0.7071	0.5259	-0.128

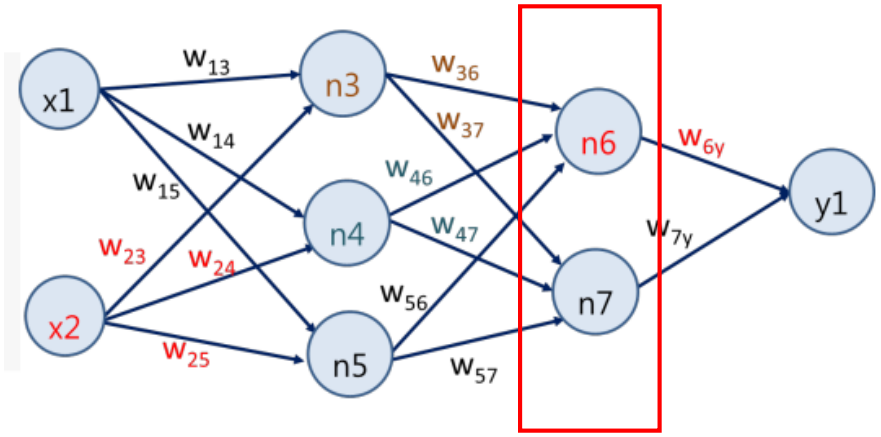
k = x * w				b3	b4	b5
0.2303	1.2414	-1.6036	+	-0.7071	0.5259	-0.128
0.6124	2.1086	-2.5408	+	-0.7071	0.5259	-0.128
4.58	6.801	-6.04	+	-0.7071	0.5259	-0.128

n		
-0.4768	1.7673	-1.7316
-0.0947	2.6345	-2.6688
3.8729	7.3269	-6.168

Calculate n6 and n7 (2nd hidden layer)

```
[8]  #Calculate n6, n7 using PyTorch matrix operation
     W1 = MyNet[1].weight
     b1 = MyNet[1].bias
     Layer2_1 = Layer1.mm(torch.transpose(W1, 1, 0)) +b1
     print(Layer2_1)

     tensor([[0.8263, 0.4333],
             [1.2301, 0.9959],
             [3.4501, 5.1976]], grad_fn=<AddBackward0>)
```

```
[9]  #verify
     Layer2 = MyNet[1](Layer1)
     print(Layer2)

     tensor([[0.8263, 0.4333],
             [1.2301, 0.9959],
             [3.4501, 5.1976]], grad_fn=<AddmmBackward0>)
```



$$\begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix} \begin{bmatrix} w_{36} & w_{37} \\ w_{46} & w_{47} \\ w_{56} & w_{57} \end{bmatrix} + [b_6 \quad b_7] = \begin{bmatrix} k_6^1 & k_7^1 \\ k_6^2 & k_7^2 \\ k_6^3 & k_7^3 \end{bmatrix} + \begin{bmatrix} b_6 & b_7 \\ b_6 & b_7 \\ b_6 & b_7 \end{bmatrix} = \begin{bmatrix} n_6^1 & n_7^1 \\ n_6^2 & n_7^2 \\ n_6^3 & n_7^3 \end{bmatrix}$$

Layer 1		
-0.4768	1.7673	-1.7316
-0.0947	2.6345	-2.6688
3.8729	7.3269	-6.168

Weight	0.3104	0.5095
	-0.0524	0.5542
	-0.3528	0.1202

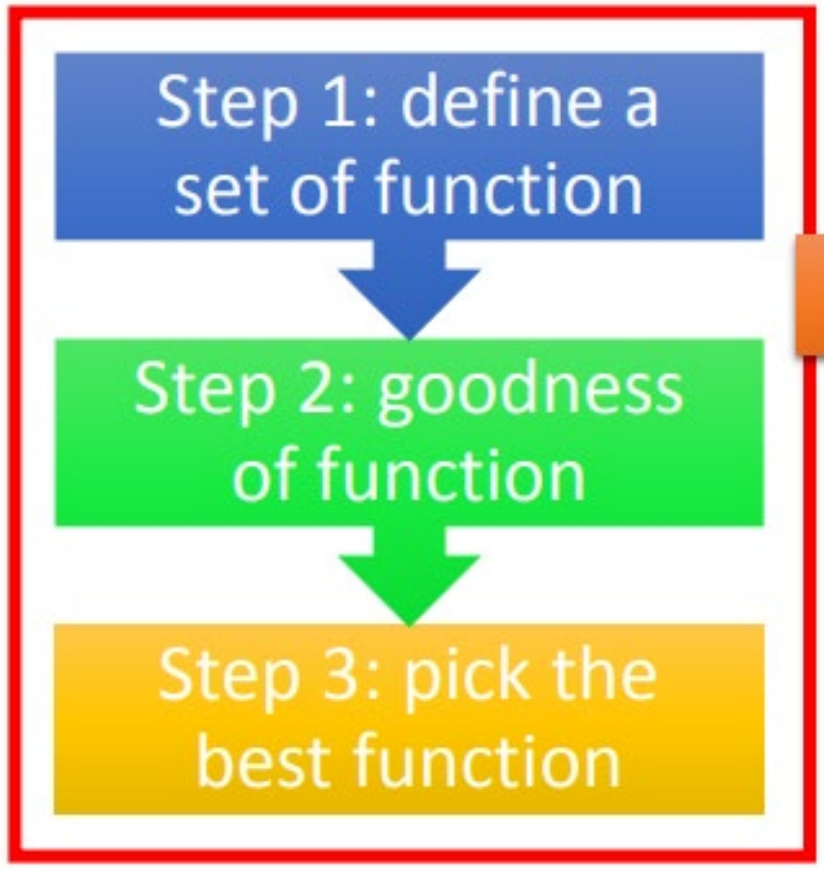
Bias	0.456	-0.0951

k = x * w	
0.370303	0.52837
0.77411	1.091
2.994289	5.292417

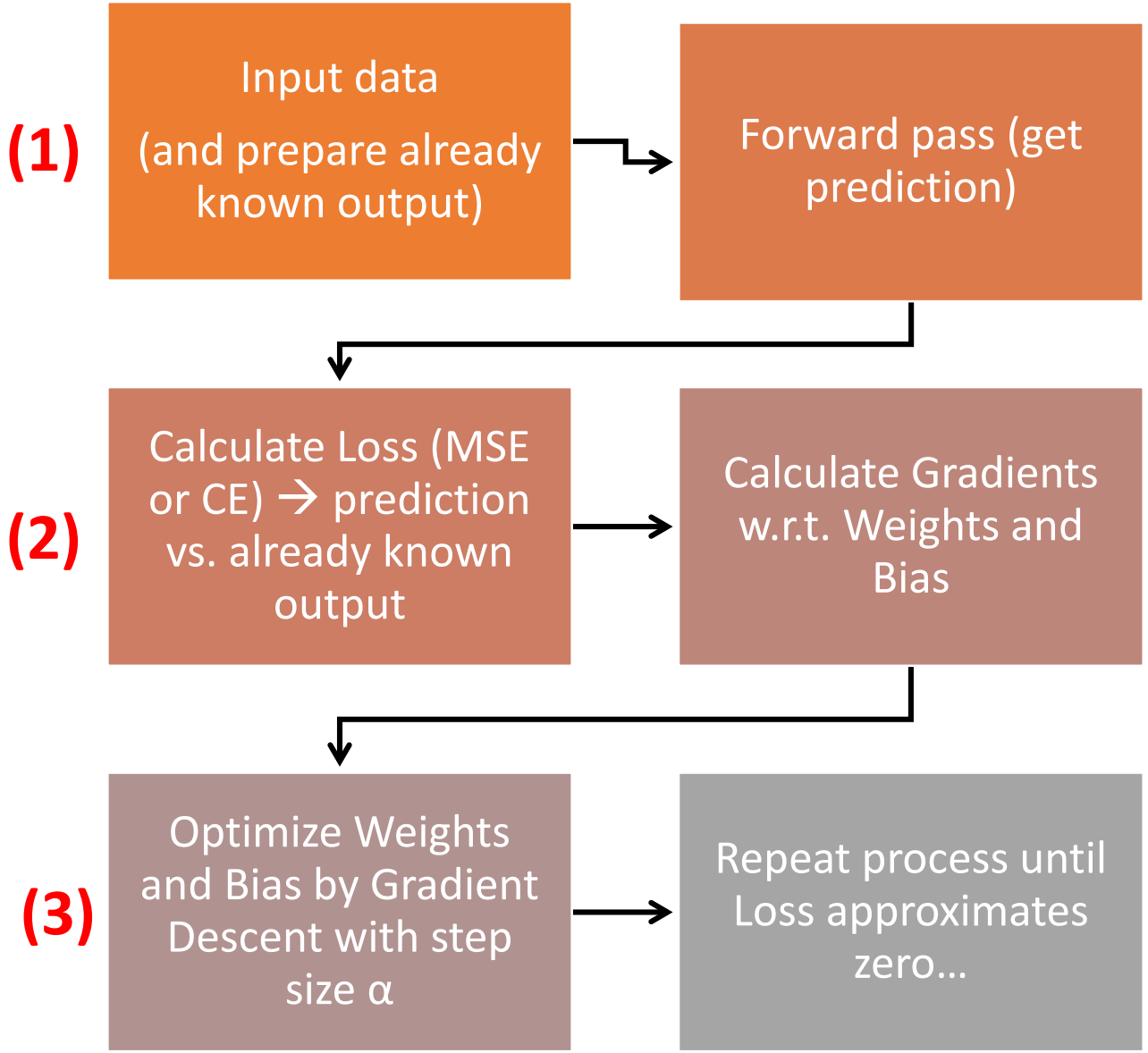
b6	b7
0.456	-0.0951
0.456	-0.0951
0.456	-0.0951

n	
0.826303	0.43327
1.23011	0.9959
3.450289	5.197317

How NN learns by supervised learning?



Reference: 李弘毅 ML Lecture 9-1
https://youtu.be/xki61j7z-30



Reference: Carlos Solorzano

How NN learns by supervised learning?

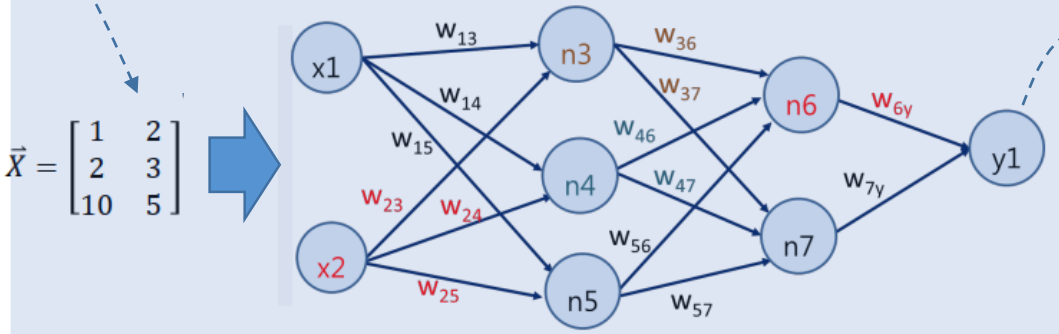
Send data as a input signal in NN

1

```
[32] lstX = [[1, 2], [2, 3], [10, 5]]
     lstY = [[7], [12], [40]]  # y=3x1+2x2
     tensorX = torch.FloatTensor(lstX)
     tensorY = torch.FloatTensor(lstY)

     tensorY_hat= MyNet(tensorX)
     print(tensorY, '\n', tensorY_hat)

     tensor([[ 7.],
             [12.],
             [40.]])
      tensor([[-0.6699],
             [-0.5647],
             [ 0.6561]], grad_fn=<AddmmBackward0>)

[33] loss = loss_func(tensorY , tensorY_hat)
     print(loss)

     tensor(588.2137, grad_fn=<MseLossBackward0>)
```

2 NN will process the input signal with initial weight and bias to get the prediction value

$$\vec{X} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix}$$



W13 = 0.1319 W23 = -0.5874 b3 = -0.5923
W14 = -0.6096 W24 = -0.5569 b4 = 0.129
W15 = 0.4028 W25 = 0.0784 b5 = 0.2674

```
0.weight
 tensor([[ 0.1319, -0.5874],
         [-0.6096, -0.5569],
         [ 0.4028,  0.0784]])
0.bias
 tensor([-0.5923,  0.1290,  0.2674])
```

W36 = -0.1401 W56 = 0.5679
W37 = 0.227 W57 = 0.3641

W46 = 0.1616 b6 = -0.4812
W47 = 0.0575 b7 = -0.0846

```
1.weight
 tensor([[-0.1401,  0.1616,  0.5679],
         [ 0.2270,  0.0575,  0.3641]])
1.bias
 tensor([-0.4812, -0.0846])
```

W6y = 0.6878
W7y = 0.6464

by=-0.4830

```
2.weight
 tensor([[0.6878, 0.6464]])
2.bias
 tensor([-0.4830])
```

Prediction

```
[[-0.6699],
 [-0.5647],
 [ 0.6561]],
```

Ground truth

$$\begin{bmatrix} y^1 \\ y^2 \\ y^3 \end{bmatrix} = \begin{bmatrix} 7 \\ 12 \\ 40 \end{bmatrix}$$

$$L = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$$

3 Calculate the difference between the actual value (ground truth) and the predicted value

4 Adjust the weight and bias to make the loss value become or close to zero, and use Gradient decent method to make a adjustment

5 NN will process the input signal with **NEW** weight and bias to get the better prediction value (smaller L)

Reference: Dominico Laksma Paramestha 拉斯麥

Calculate MSE using PyTorch and Excel

- K = number of data points.
- y_i = ground truth value.
- $\hat{y}_i$ = prediction from NN.

$$MSE = \frac{1}{K}\sum_{i=1}^{K}(y_i - \hat{y}_i)^2$$

```
tensorY = torch.FloatTensor(lstY)

tensorY_hat= MyNet(tensorX)
print(tensorY, '\n', tensorY_hat.detach().numpy())
```

```
tensor([[ 7.],
        [12.],
        [40.]])
 [[ 3.8501935]
 [ 6.6894126]
 [22.304363 ]]
```

```
[14] loss = loss_func(tensorY , tensorY_hat)
     print(loss)
```

```
tensor(117.0864, grad_fn=<MseLossBackward0>)
```

	A	B	C	D	E
1	y	y_hat		(y - y_hat)^2	
2	7	3.850194		9.921280987	
3	12	6.689413		28.20233853	
4	40	22.30436		313.1355688	
5					
6			sum	351.2591884	
7			mean	117.0863961	
8					

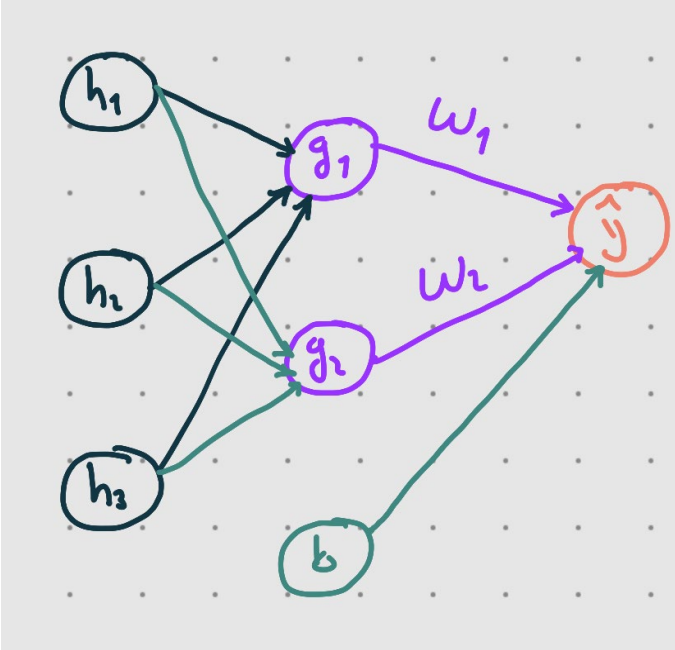
Reference: Carlos Solorzano 羅艾多

How to adjust w and b so that L 628.709 → 0

$$\nabla L = [\frac{\partial L}{w_{13}}, \frac{\partial L}{w_{14}}, \frac{\partial L}{b_3}, \dots, \frac{\partial L}{w_{36}}, \dots]$$

- Calculate partial derivative of L with respect to the weights.
- $\nabla L = \frac{\partial L}{\partial W}$
- By gradient descent, with a given step size α, update weights.
- $W_{new} = W_{old} - \alpha \frac{\partial L}{\partial W}$

How PyTorch calculate gradient?



$$\mathcal{L} = \frac{1}{K}\sum_{i}^{K}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{K}\left[(y_0 - \hat{y}_0)^2 + (y_1 - \hat{y}_1)^2 + \dots + (y_K - y_K)^2\right]$$

(The same!)

$$\mathcal{L} = \frac{1}{K}\left[\underbrace{(y_0 - \hat{y}_0)^2}_{f} + \underbrace{(y_1 - \hat{y}_1)^2}_{f} \dots + \dots \underbrace{(y_K - \hat{y}_K)^2}_{f}\right]$$

$$\mathcal{L} = \frac{1}{K}\left\{\underbrace{\left[y_0 - (g_1^0 w_1 + g_2^0 w_2 + b)\right]^2}_{f} + \dots + \left[y_K - (\dots)\right]^2\right\}$$

z

$$\frac{\partial L}{\partial W} = \left[\frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2}\right]$$

$$\frac{\partial z}{\partial w_1} = \frac{\partial z}{\partial f} \cdot \frac{\partial f}{\partial w_1}$$

$$\frac{\partial z}{\partial f} = \frac{\partial f^2}{\partial f}, \quad 2f = 2\left[y_0 - \overbrace{(g_1^0 w_1 + g_2^0 w_2 + b)}^{\hat{y}}\right]$$

$$\frac{\partial f}{\partial w_1} = -g_1^0$$

$$\frac{\partial z}{\partial w_1} = -2\left[y_0 - \hat{y}_0\right] \cdot g_1^0$$

$$\vdots$$

$$\frac{\partial L}{\partial w_1} = -\frac{2}{K} \cdot \sum_{i}^{K}(y_i - \hat{y}_i)\, g_1^i$$

$$\frac{\partial L}{\partial w_2} = -\frac{2}{K} \cdot \sum_{i}^{K}(y_i - \hat{y}_i)\, g_2^i$$

Reference: Carlos Solorzano 羅艾多

How PyTorch calculate gradient? (2)

Given $y = [7, 12, 40]^T$

$\hat{y} = \begin{bmatrix} 0.6619663 \\ 0.6377611 \\ 0.5434885 \end{bmatrix}$

Calc. gradient w.r.t. w_1

$g_1 = \begin{bmatrix} 0.5087925 \\ 0.698878 \\ 1.4169338 \end{bmatrix}$ $g_2 = \begin{bmatrix} 0.5024761 \\ 0.42866528 \\ 0.14511722 \end{bmatrix}$

$$\frac{\partial L}{\partial w_1} = -\frac{2}{3} \cdot (y - \hat{y}) \cdot g_1^T$$

$$\frac{\partial L}{\partial w_1} = -\frac{2}{3} \cdot \begin{bmatrix} 6.3380337 \\ 11.362239 \\ 39.456512 \end{bmatrix} \cdot [0.509,\ 0.699,\ 1.417]$$

$$\frac{\partial L}{\partial w_1} = -\frac{2}{3} \cdot 67.072828 = \boxed{-44.7152}$$

```
array([[0.5087925 , 0.5024761 ],
       [0.698878  , 0.42866528],
       [1.4169338 , 0.14511722]],
```

```
tensor([[ 7.],
        [12.],
        [40.]])
[[0.6619663]
 [0.6377611]
 [0.5434885]]
[[ 6.3380337]
 [11.362239 ]
 [39.456512 ]]
```

```
2.weight
 tensor([[0.1058, 0.6004]])
Gradient tensor([[-44.7152,  -9.1874]])
2.bias
 tensor([0.3065])
Gradient tensor([-38.1045])
```

NN learned as the loss value is reduced

$$w^1 \leftarrow w^0 - \eta \frac{dL}{dw}\Big|_{w=w^0}$$

In [5]:
```
loss = loss_func
print(loss)
```

```
tensor(628.7084,
```

```
if param.requires_grad:
  param = param - learning_rate*param.grad
```

In [10]:
```
tensorY_hat= My
print(tensorY)
loss = loss_func
print(loss)
```

```
tensor([[ 7.],
        [12.],
        [40.]])
tensor(459.2752,
```

628.70 → 459.275