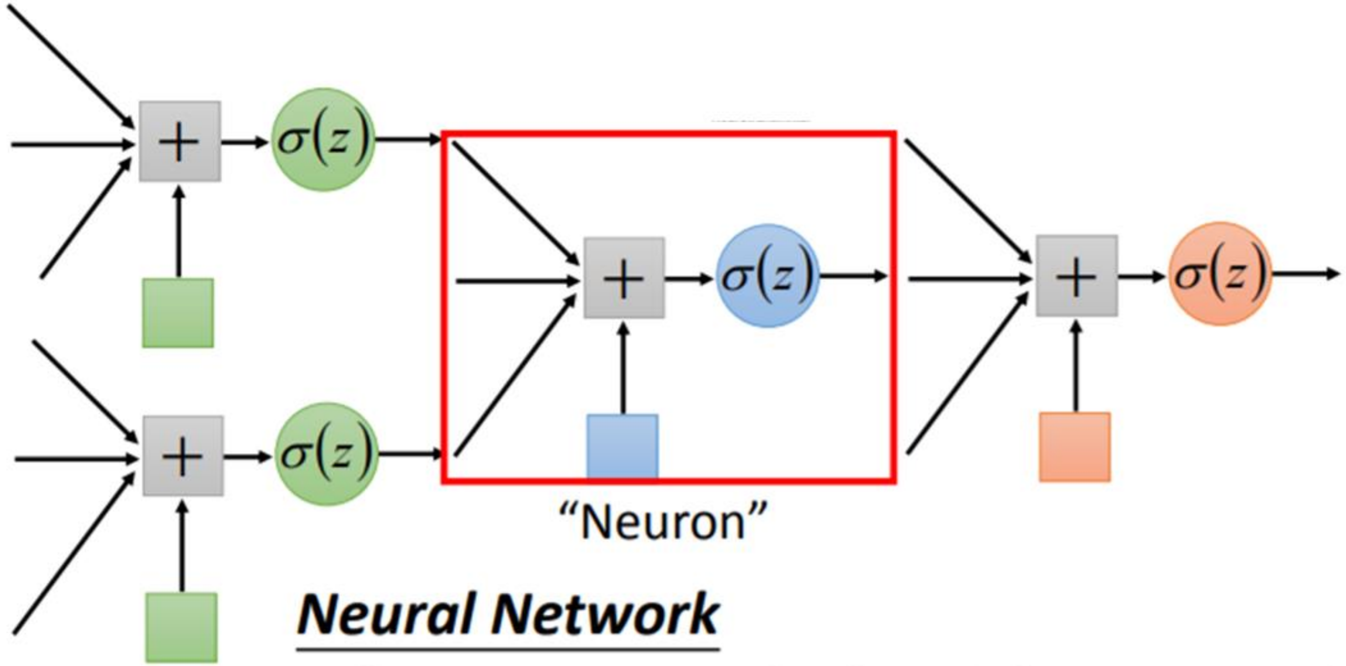
Neural network



Neural Network

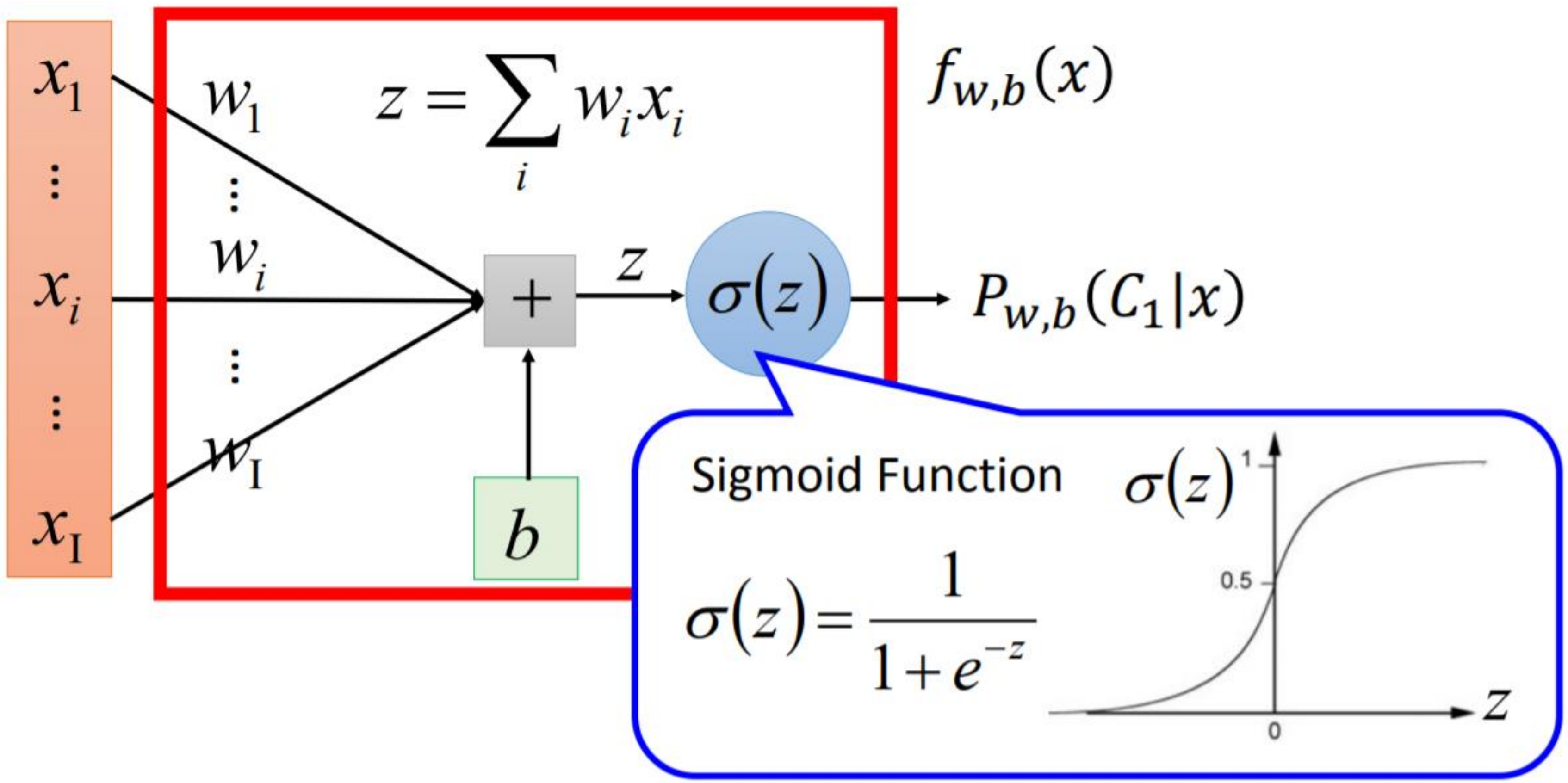
Different connection leads to different network structures

θ Network parameter θ: all the weights and biases in the "neurons"

Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

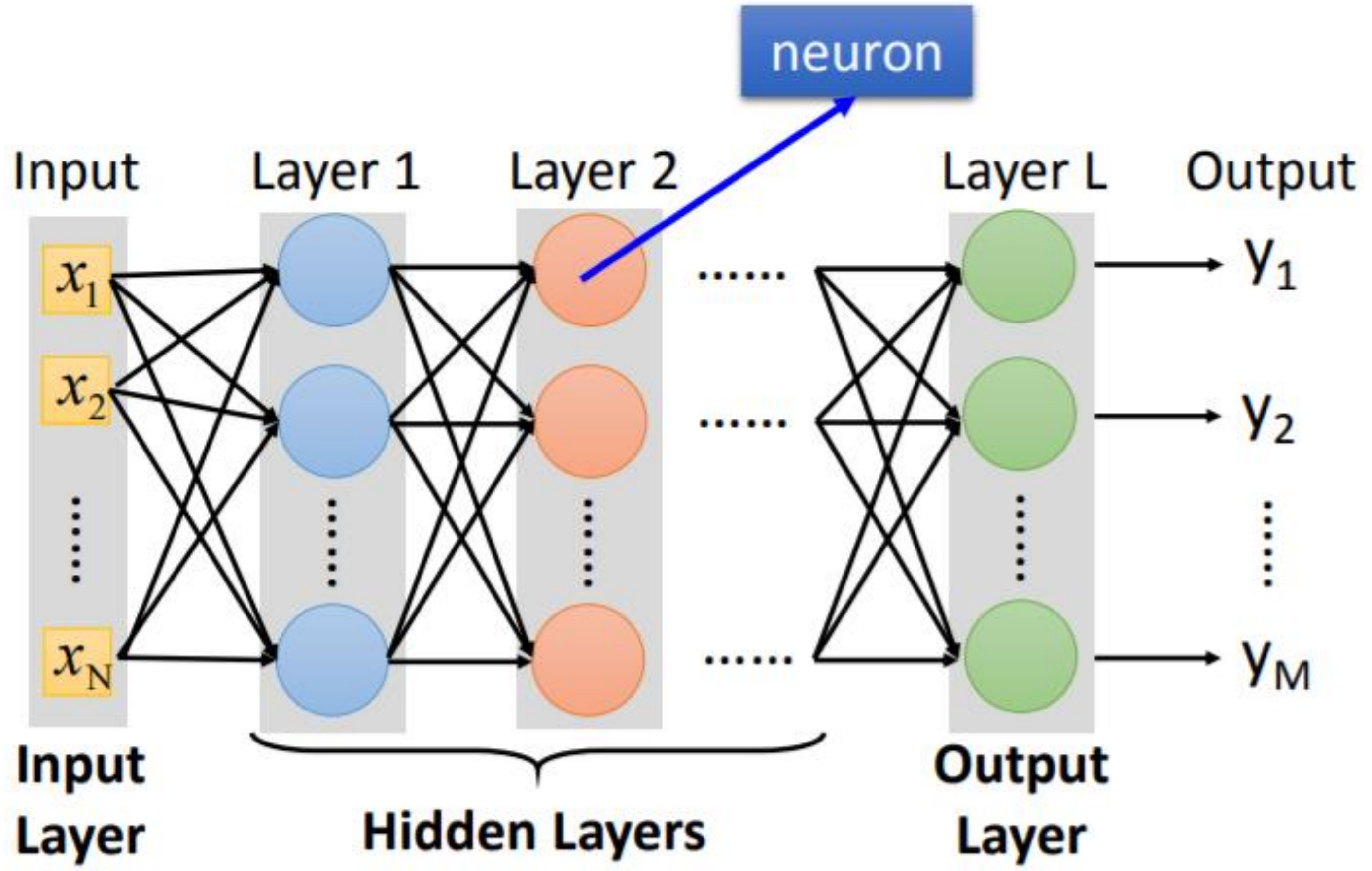
Each neuron is a classifier

$$P(C_1 | x) = \sigma(w \cdot x + b) = \sigma\left(\sum_i w_i x_i + b\right)$$



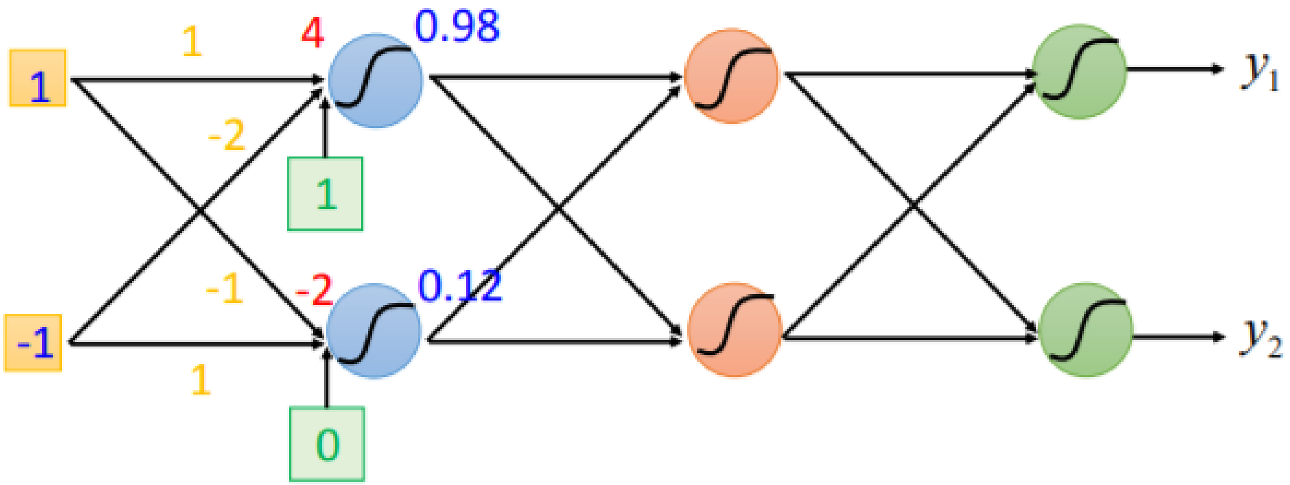
Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

MLP is a fully connected feedforward network



Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

Fully connected feed forward network is implemented as matrix operation



$$y = \sigma(w \cdot x + b)$$

$$\sigma\Bigg(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} \Bigg) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

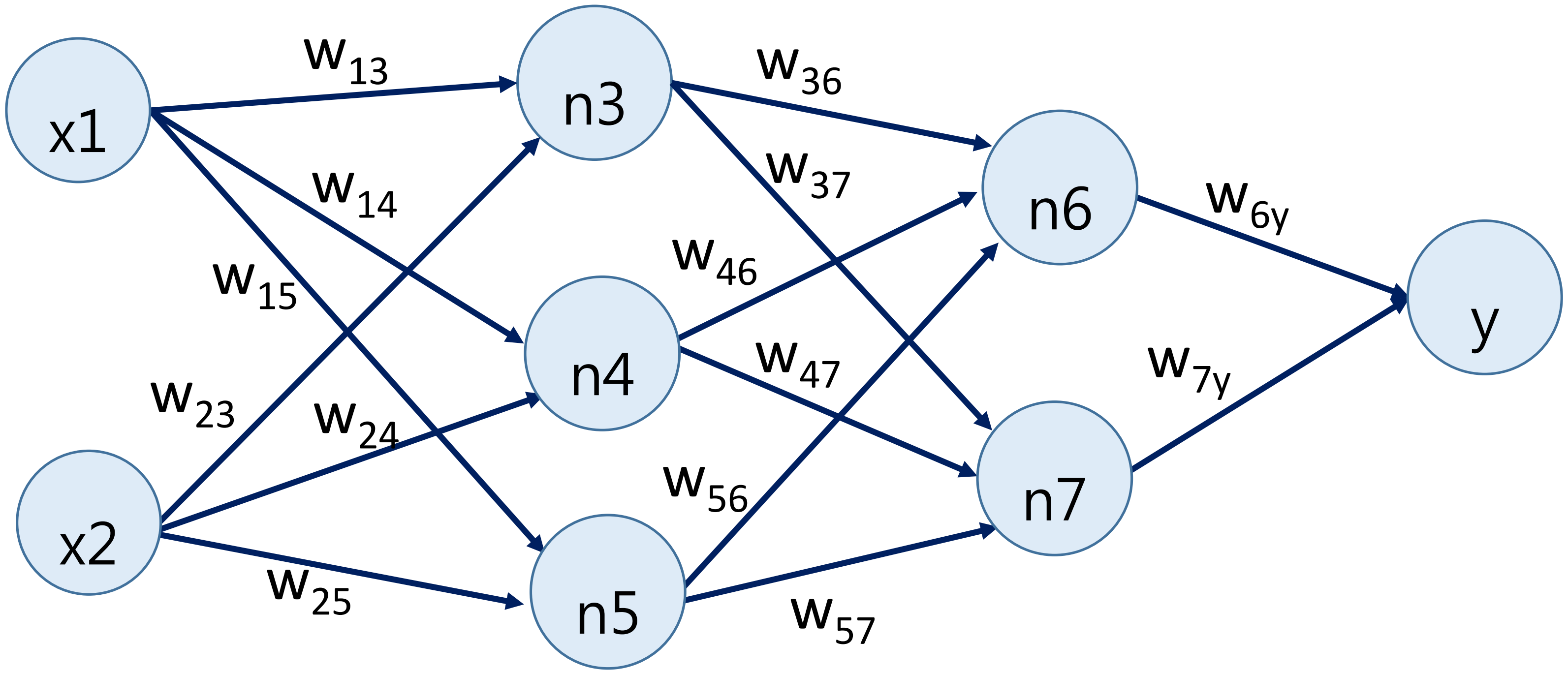
Practice

- Run “6. Matrix operation.ipynb"

Matrix operation

```
MyNet = nn.Sequential(
    nn.Linear(2, 3),
    nn.Linear(3, 2),
    nn.Linear(2, 1)
)
```

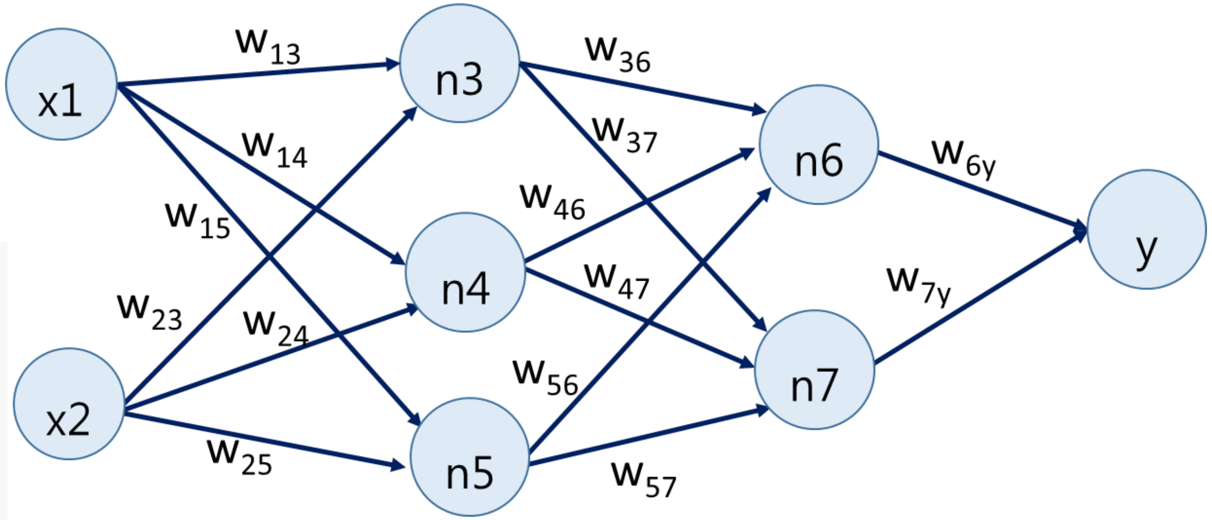
$$\begin{array}{cc} & \begin{array}{cc} x_1 & x_2 \end{array} \\ \vec{X} = & \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix} \end{array}$$



$$\hat{y} = \begin{bmatrix} 7 \\ 12 \\ 40 \end{bmatrix}$$

Matrix operation

```python
for  param  in  MyNet.parameters():
        if  param.requires_grad:
                print(param.data)
```




```
tensor([[ 0.4727, -0.5188],
        [-0.5681, -0.6032],
        [-0.0252, -0.3011]])
tensor([-0.6986, -0.6602, -0.4860])
tensor([[-0.5549,  0.2550,  0.4584],
        [ 0.2930,  0.0849, -0.3146]])
tensor([0.1677, 0.0736])
tensor([[ 0.4106, -0.3618]])
tensor([-0.2270])
```

$$\begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}$$

$$\begin{bmatrix} b_3 & b_4 & b_5 \end{bmatrix}$$

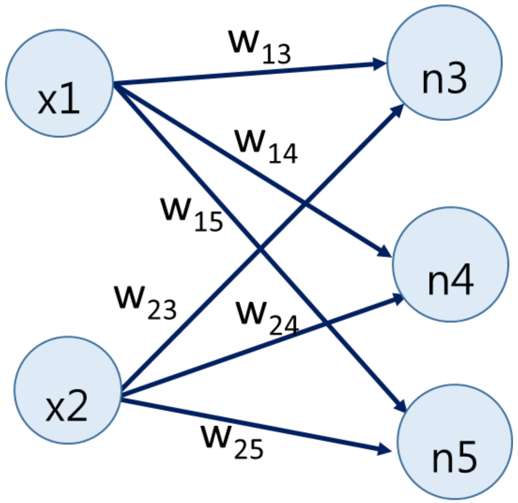
$$\begin{bmatrix} w_{36} & w_{46} & w_{56} \\ w_{37} & w_{47} & w_{57} \end{bmatrix}$$

$$\begin{bmatrix} b_6 & b_7 \end{bmatrix}$$

$$\begin{bmatrix} w_{6y} & w_{7y} \end{bmatrix}$$

$$\begin{bmatrix} b_y \end{bmatrix}$$

$$\vec{X} = \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix} \end{array}$$




```
W1 = MyNet[0].weight
b1 = MyNet[0].bias
print(W1, W1.shape, b1)
```

```
Parameter containing:
tensor([[ 0.4727, -0.5188],
        [-0.5681, -0.6032],
        [-0.0252, -0.3011]],
tensor([-0.6986, -0.6602, -0.4860], r
```

$$\begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} w_{13} & w_{14} & w_{15} \\ w_{23} & w_{24} & w_{25} \end{bmatrix}}_{} + \begin{bmatrix} b_3 & b_4 & b_5 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} k_3^1 & k_4^1 & k_5^1 \\ k_3^2 & k_4^2 & k_5^2 \\ k_3^3 & k_4^3 & k_5^3 \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \end{bmatrix}}_{}$$

$$\boxed{\begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix}}$$

Use Excel to verify

```
#Calculate n3, n4, n5
HiddenLayer1 = MyNet[0](tensorX)
print(HiddenLayer1)
```

```
tensor([[-1.2635, -2.4348, -1.1135],
        [-1.3097, -3.6061, -1.4398],
        [ 1.4340, -9.3577, -2.2441]],
```

```
#Calculate n3, n4, n5 using Pytorch matrix operation
HiddenLayer1 = tensorX.mm(torch.transpose(W1, 1, 0)) + b1
print(HiddenLayer1)
```

```
tensor([[-1.2635, -2.4348, -1.1135],
        [-1.3097, -3.6061, -1.4398],
        [ 1.4340, -9.3577, -2.2441]], grad_fn=<AddBackward0>)
```

```
#Calculate n6, n7 using PyTorch matrix operation
W2 = MyNet[1].weight
b2 = MyNet[1].bias
HiddenLayer2 = HiddenLayer1.mm(torch.transpose(W2, 1, 0)) +b2
print(HiddenLayer2)
```

```
tensor([[-0.2625, -0.1530],
        [-0.6852, -0.1632],
        [-4.0429,  0.4054]], grad_fn=<AddBackward0>)
```

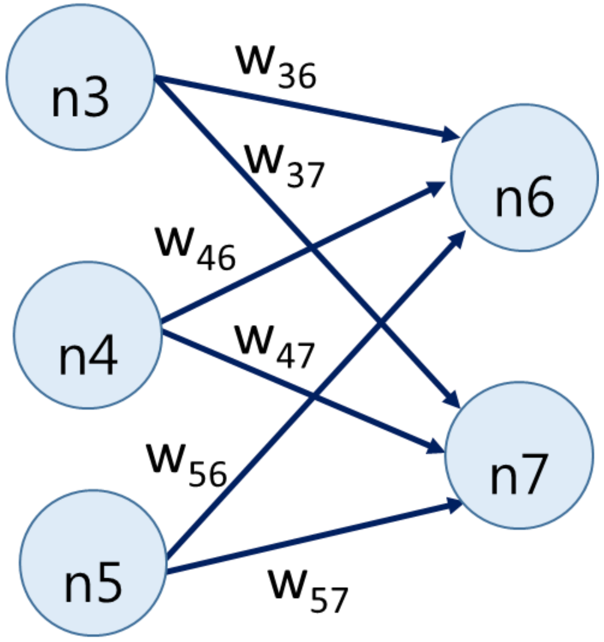
$$\begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} -1.2635 & -2.4348 & -1.1135 \\ -1.3097 & -3.6061 & -1.4398 \\ 1.4340 & -9.3577 & -2.2441 \end{bmatrix} \begin{bmatrix} w_{36} & w_{37} \\ w_{46} & w_{47} \\ w_{56} & w_{57} \end{bmatrix}} + [b_6 \quad b_7]$$

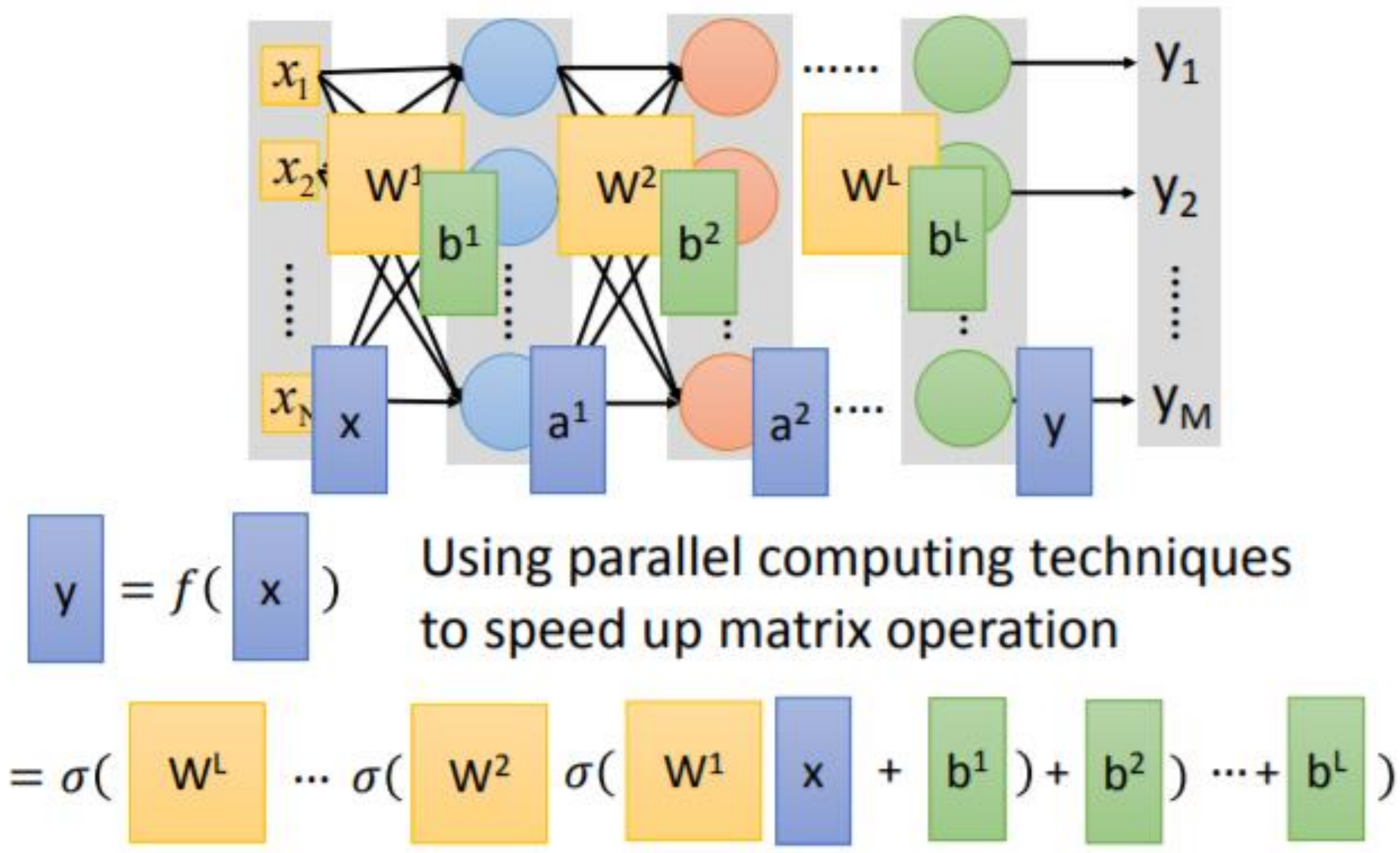
$$\underbrace{\begin{bmatrix} k_6^1 & k_7^1 \\ k_6^2 & k_7^2 \\ k_6^3 & k_7^3 \end{bmatrix} + \begin{bmatrix} b_6 & b_7 \\ b_6 & b_7 \\ b_6 & b_7 \end{bmatrix}}$$

$$\begin{bmatrix} n_6^1 & n_7^1 \\ n_6^2 & n_7^2 \\ n_6^3 & n_7^3 \end{bmatrix}$$

Use Excel to verify



Use parallel computing to speed up matrix operation



Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

Use parallel computing to speed up matrix operation

```
In [2]: if(torch.cuda.is_available()):
            device = torch.device("cuda")
            print(device, torch.cuda.get_device_name(0))
        else:
            device= torch.device("cpu")
            print(device)
```

```
cuda Tesla P100-PCIE-16GB
```

```
tensorX = torch.FloatTensor(trainX).to(device)
tensorY_hat = torch.FloatTensor(trainY_hat).to(device)
print(tensorX.shape, tensorY_hat.shape)
```

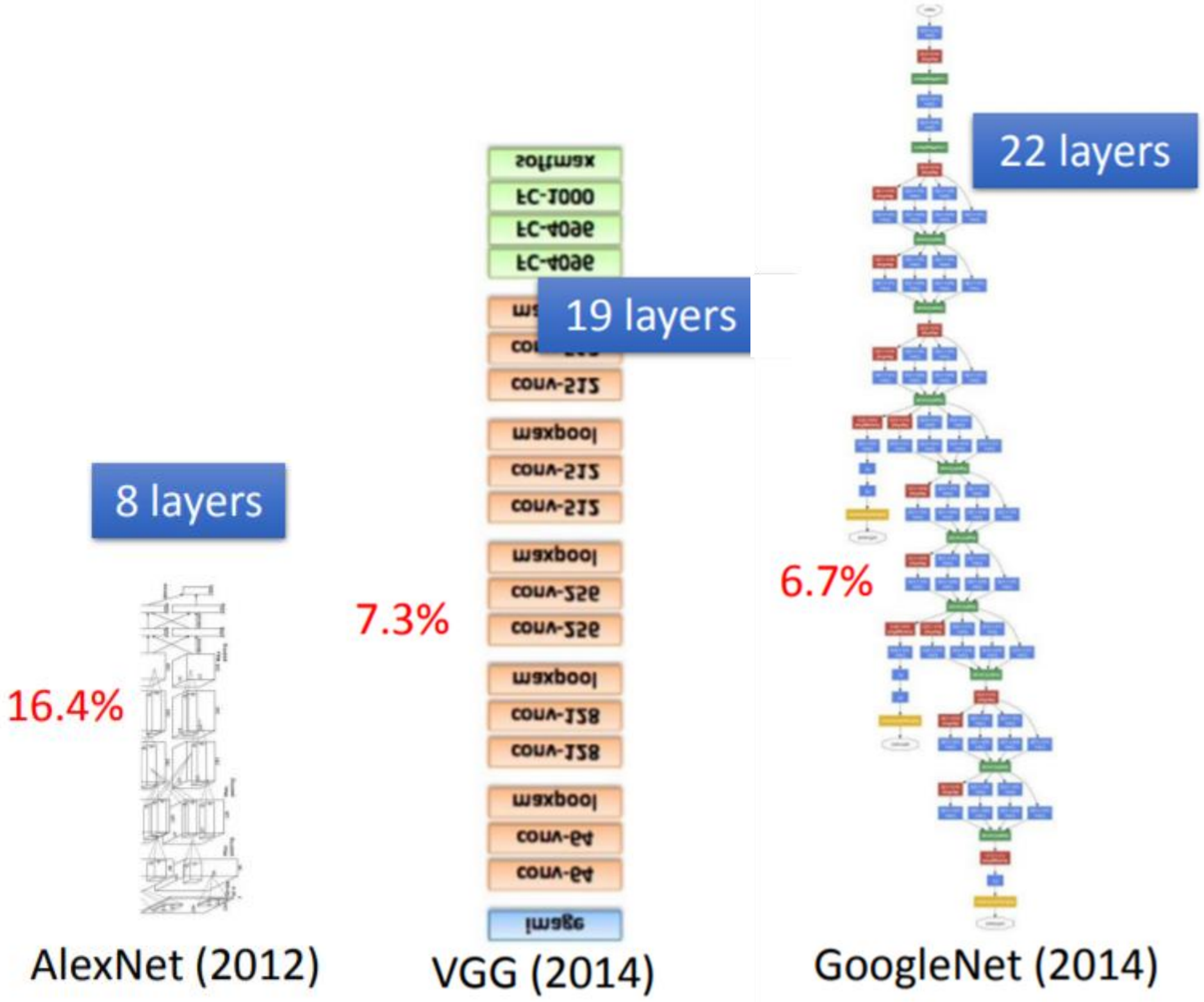
```
torch.Size([128, 2]) torch.Size([128, 1])
```

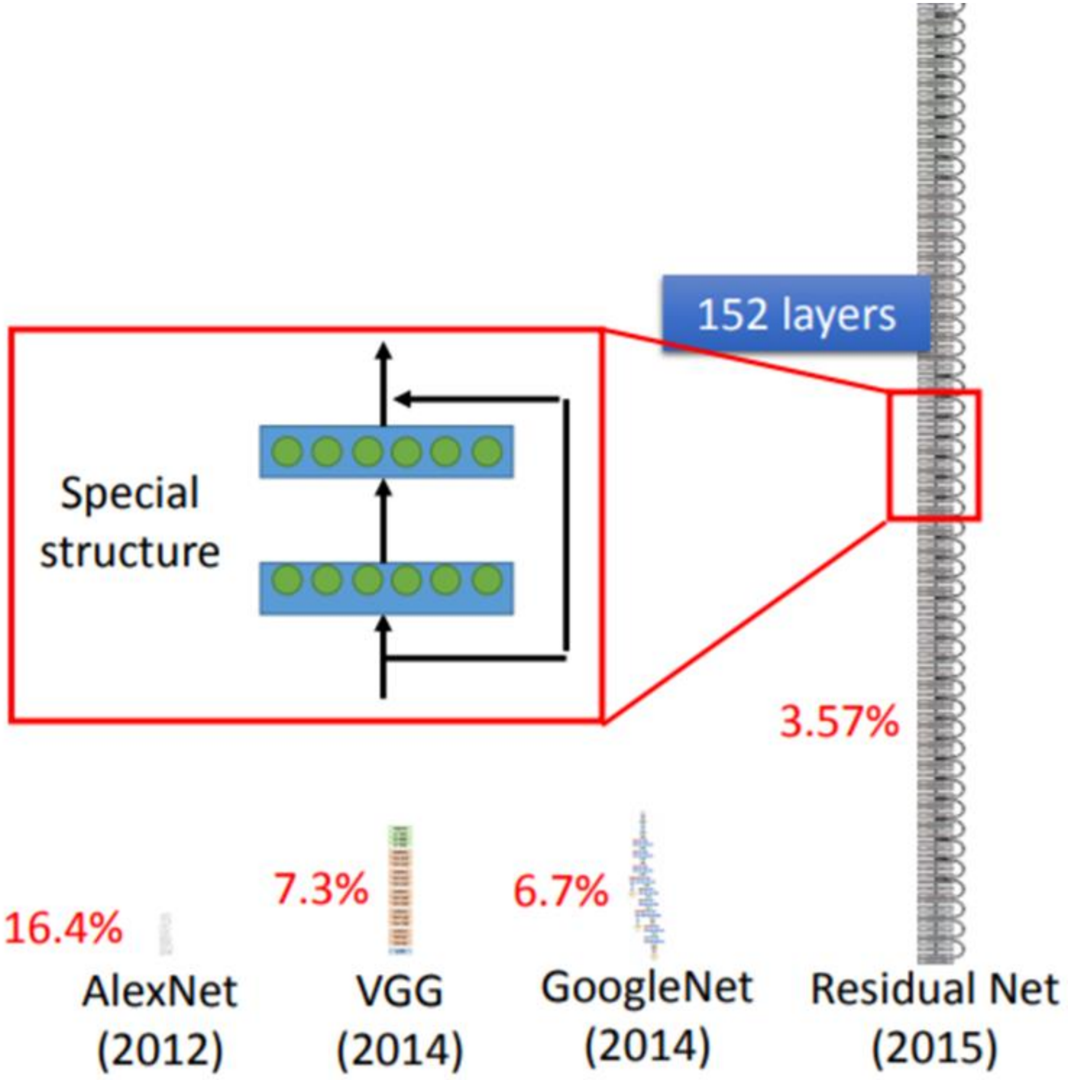
```
conv1_out = conv1(imageTensor.to(device))
conv1_out.shape
#output image (feature map) has 64 channels
```

```
torch.Size([1, 64, 55, 55])
```

Why deep ?

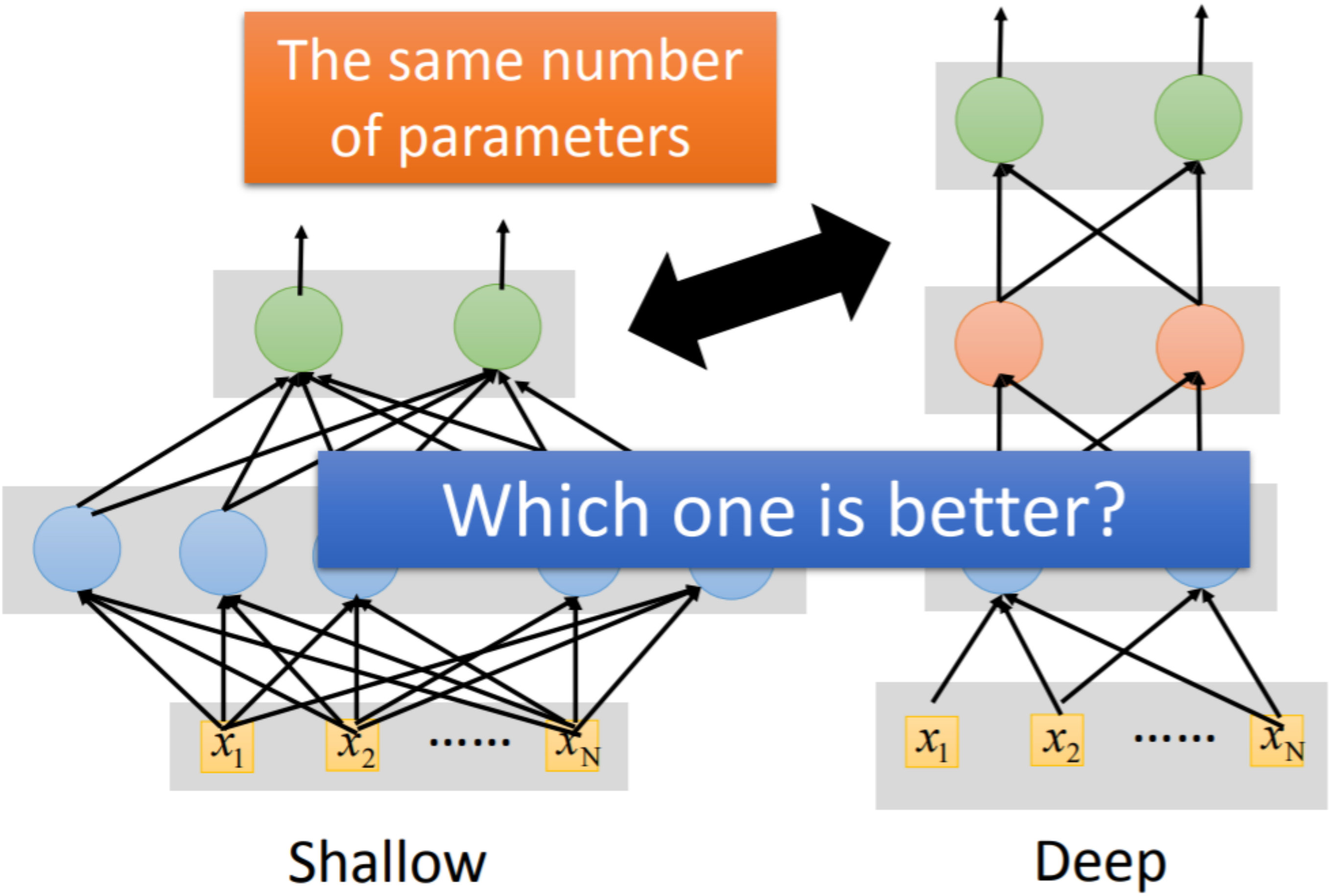
Going deeper and deeper…





Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

With same number of parameters, which NN is better?



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Deep is better

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

deep + thin

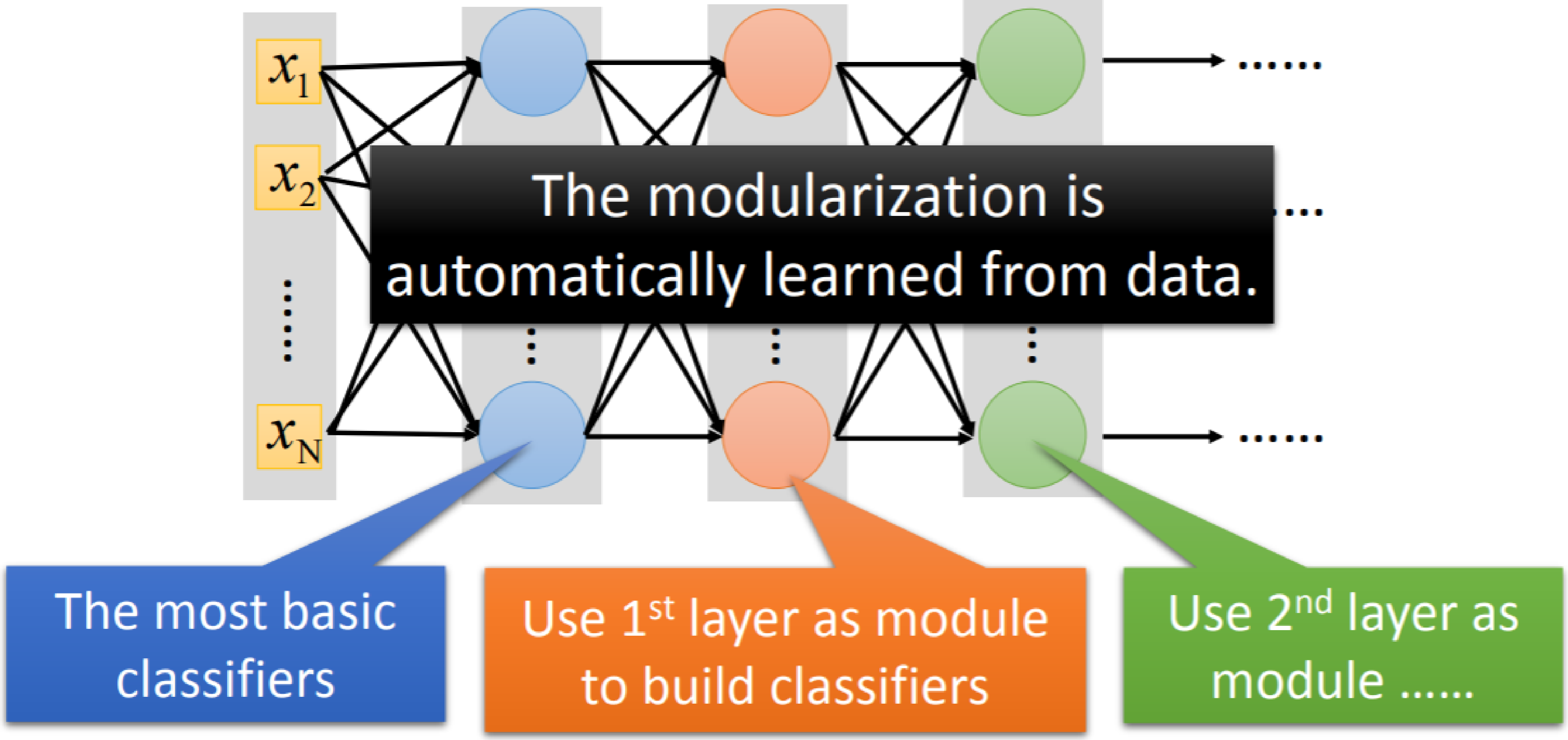
short + fat

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Reason 1 – Modularization

- Deep → Modularization → Less training data?



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

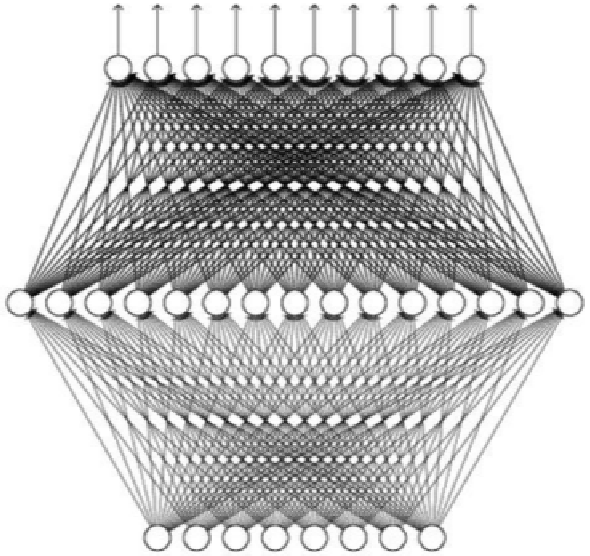
Universality theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

Yes, shallow network can represent any function.

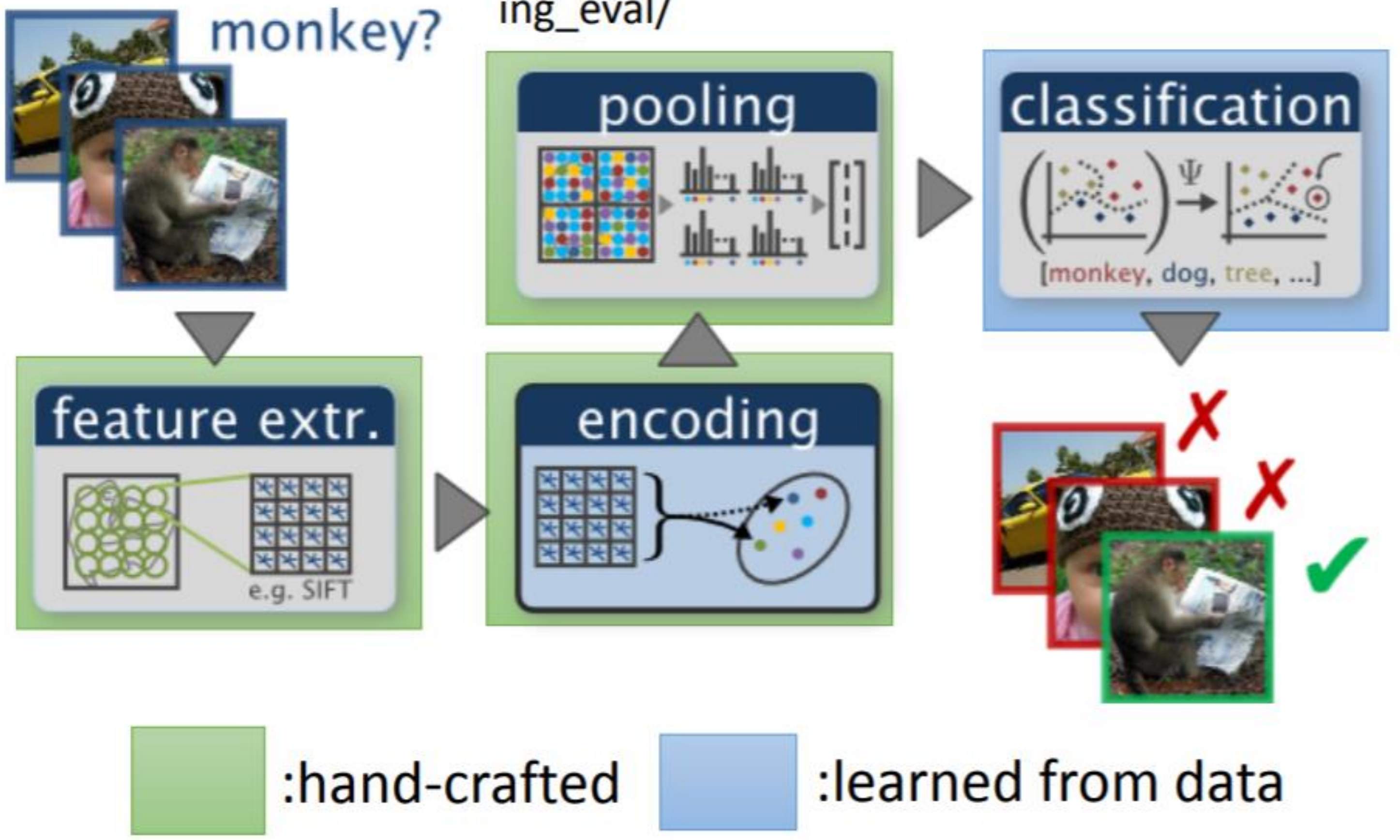
However, using deep structure is more effective.

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Reason 2: End-to-end learning

- Shallow Approach

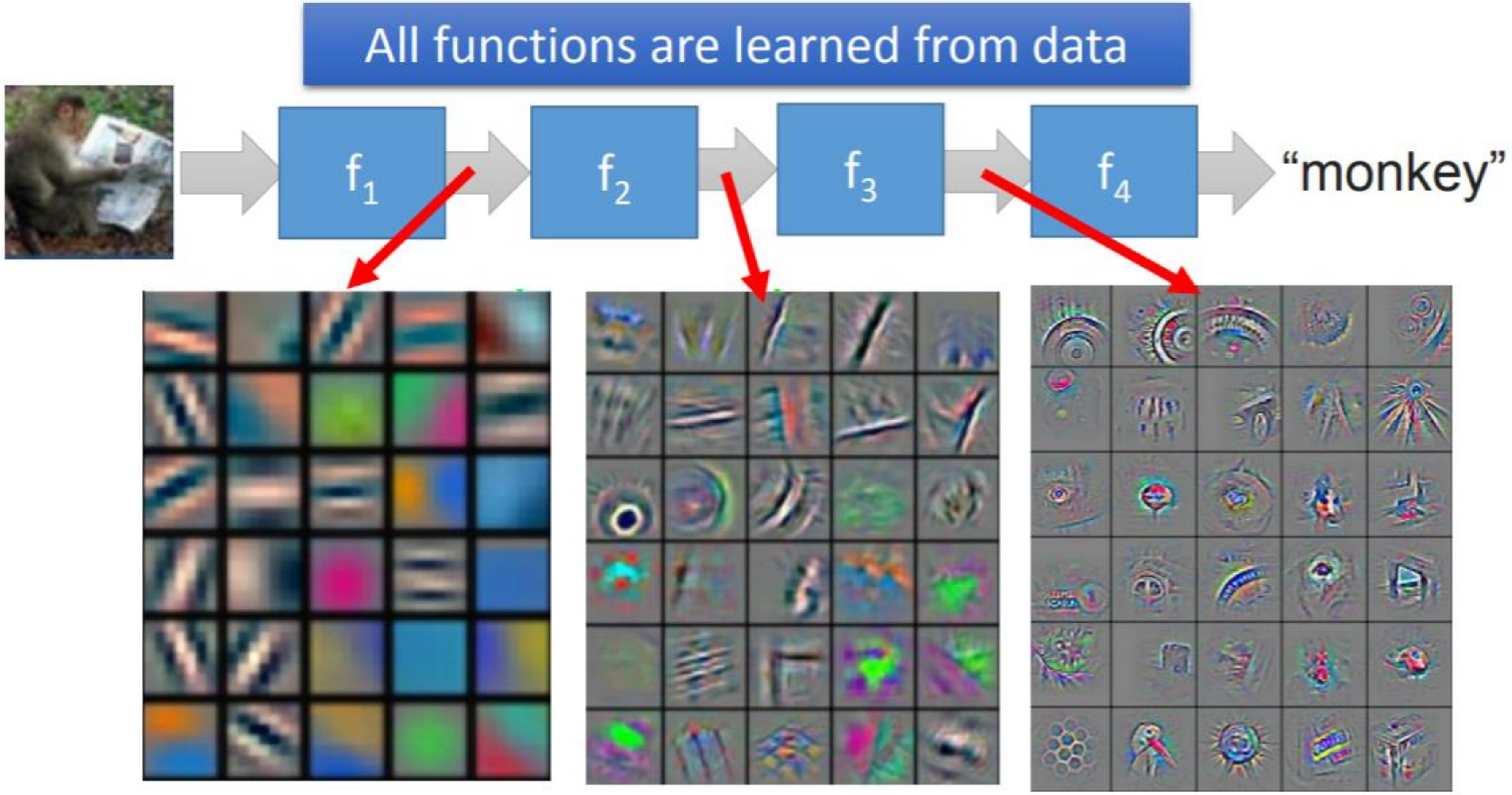
http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

End-to-end learning

- Deep Learning

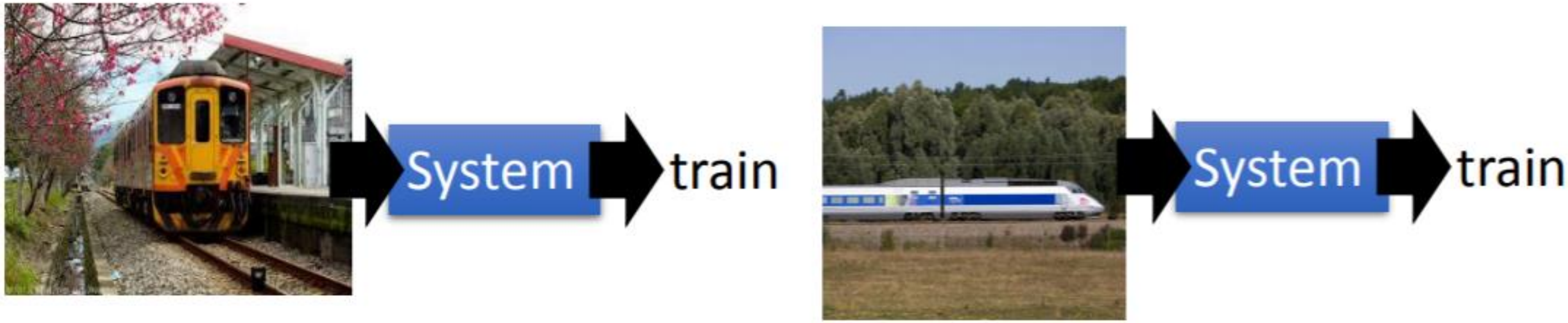


Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Reason 3 - Easier to handle complex task

- Very similar input, different output

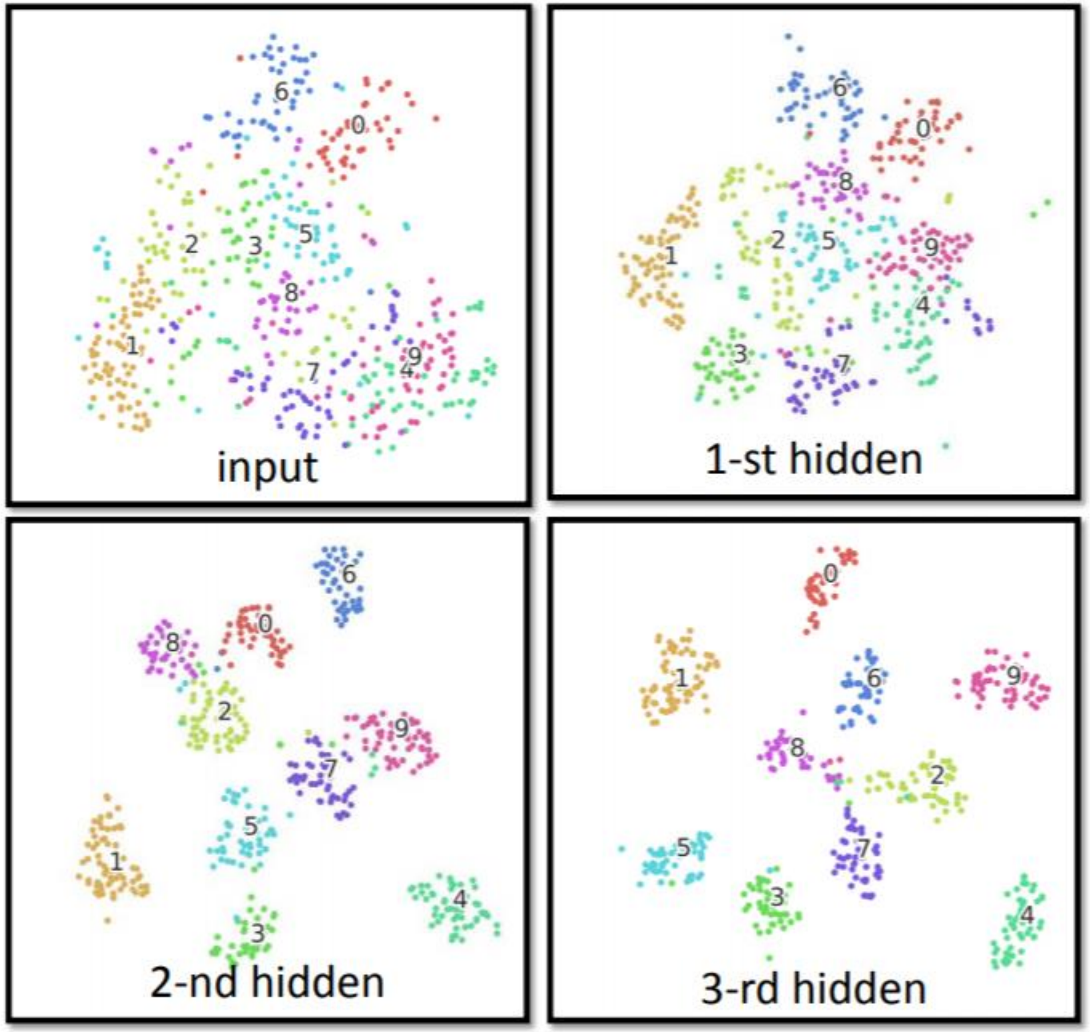
- Very different input, similar output



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Easier to handle complex task with DL

MNIST



How to implement this in PyTorch?

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8