



# Deep Learning With PyTorch

## Lecture 1

张天翔 Apr 3rd 2021



有精神就好

# REF

Lecture notes

<https://github.com/sagizty/Insight/tree/main/StarUp/PPT>

codes

<https://github.com/sagizty/Insight/tree/main/StarUp>

A online book for suggestion

<https://zh-v1.d2l.ai>

PyTorch Tutorials

<https://pytorch.org/tutorials>



深度学习

What is called  
DeepLearning

# AI

ML

DL (Deep learning)

CV

ANN (Artificial Neural Network)

CNN (Convolutional  
Neural Networks)

MLP

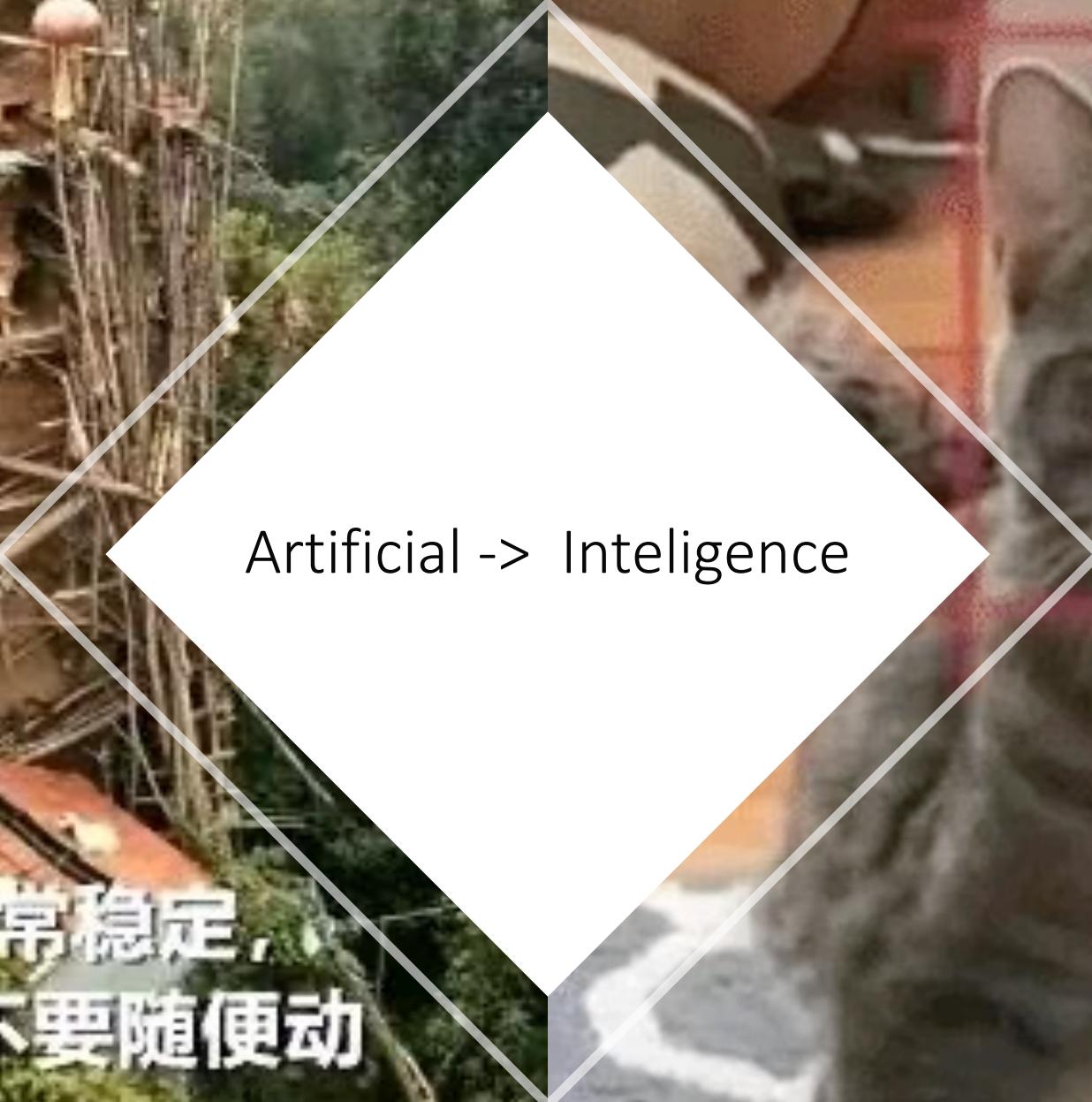
Transformer

RNN (Recurrent  
Neural Network)

LSTM, GRU

NLP

暴

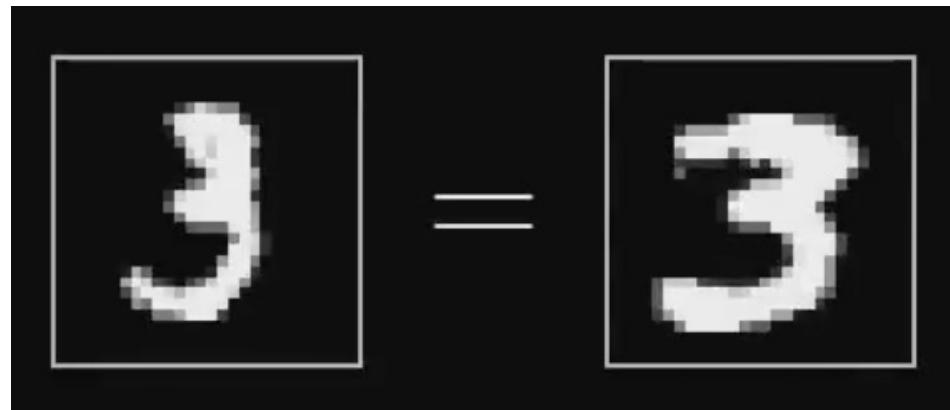


系统非常稳定，  
所有代码不要随便动

Artificial -> Intelligence



dog  
狗



So why its 3 to you?

# Transform an abstractive sample into a machine representation

00000000
00111100
00000100
00111100
00000100
00111100
00000000

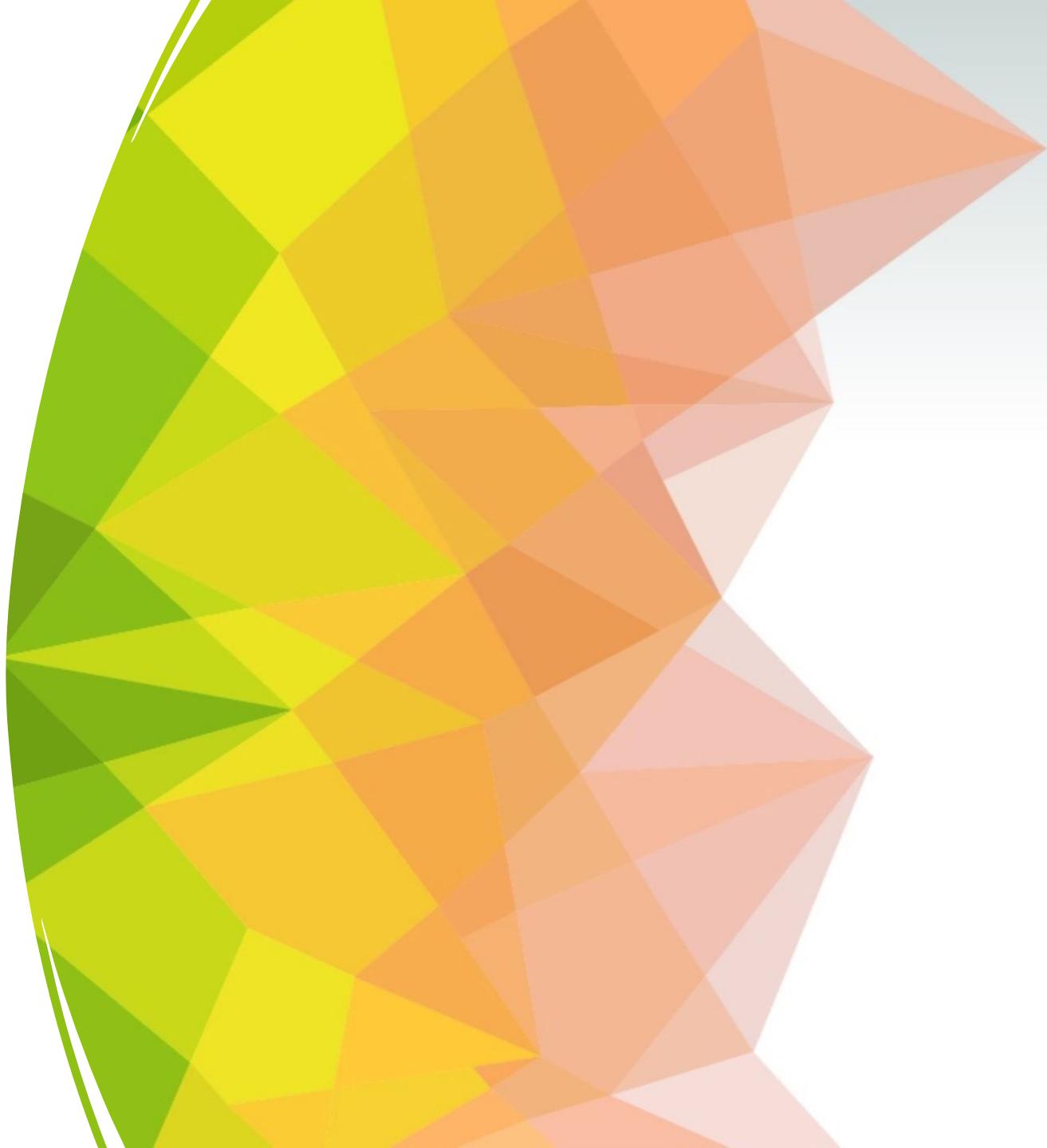
# So why its 3 to Machines?

# What is called Deep Learning

An approach to enchant machine with wisdom

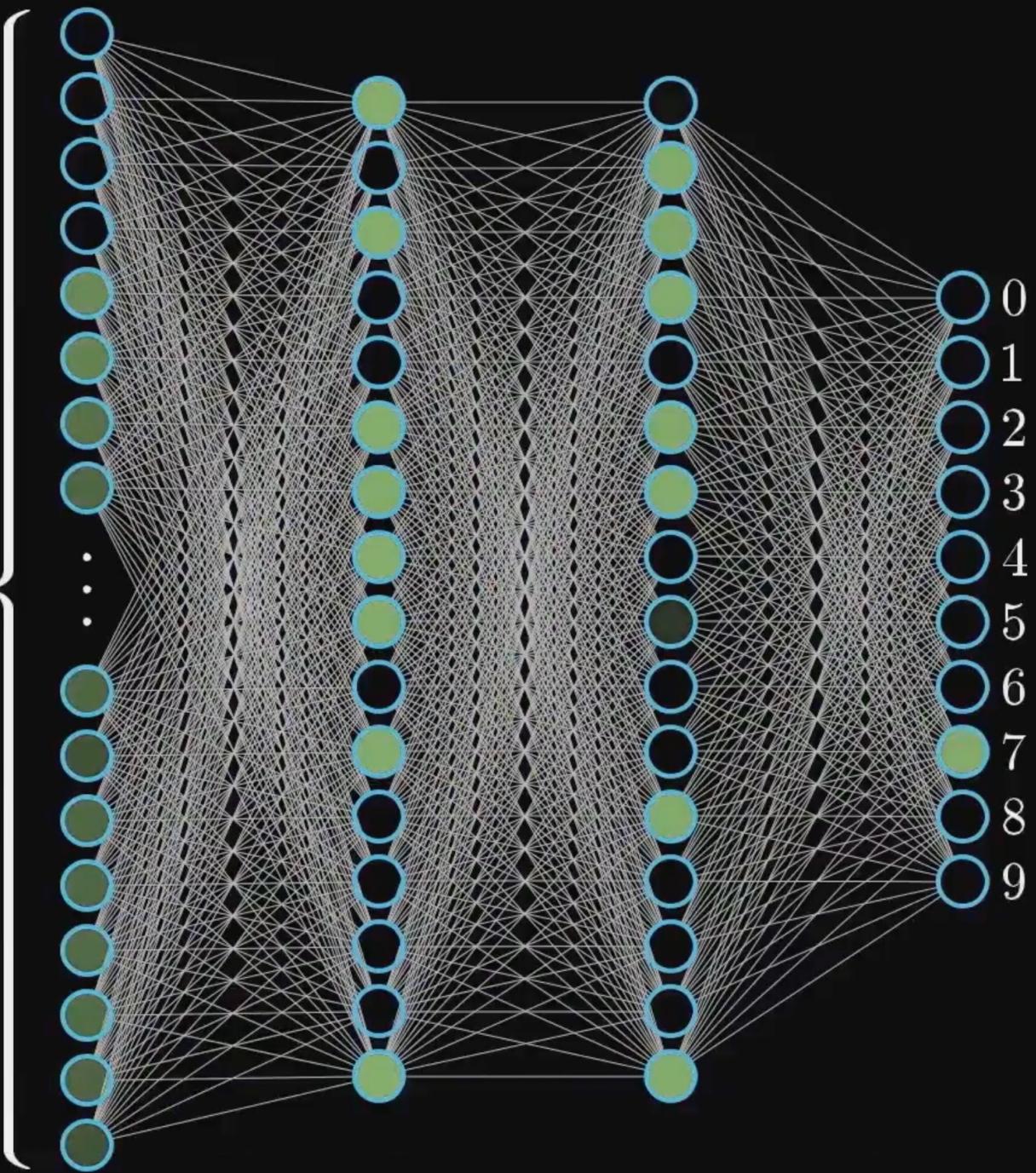
The first lecture will lay on MLP

1. Nerial networks and activations
2. What is Training and supervision,  
loss back-propagation
3. CNN, convolution, and pooling

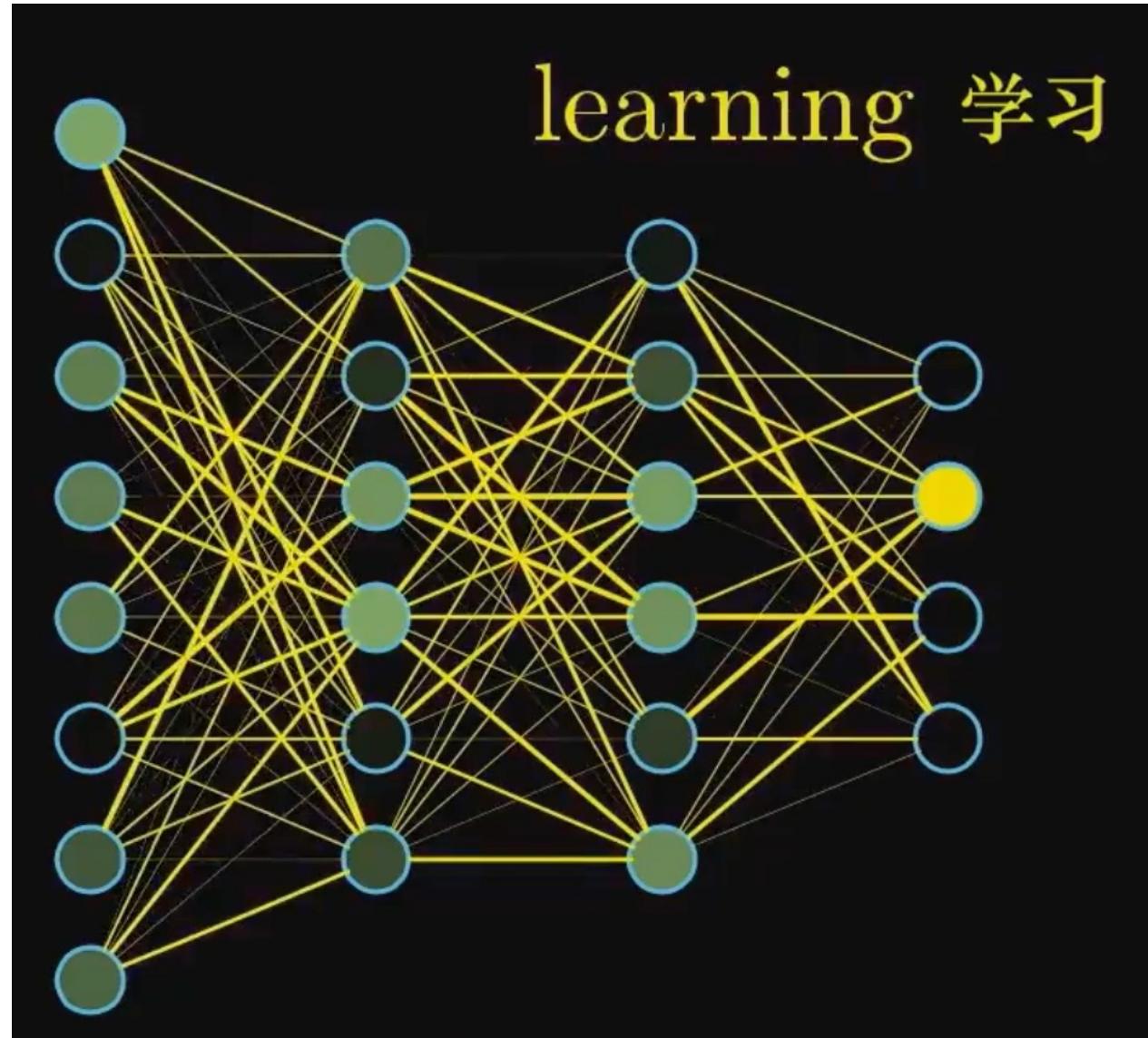




784



# Neural Network

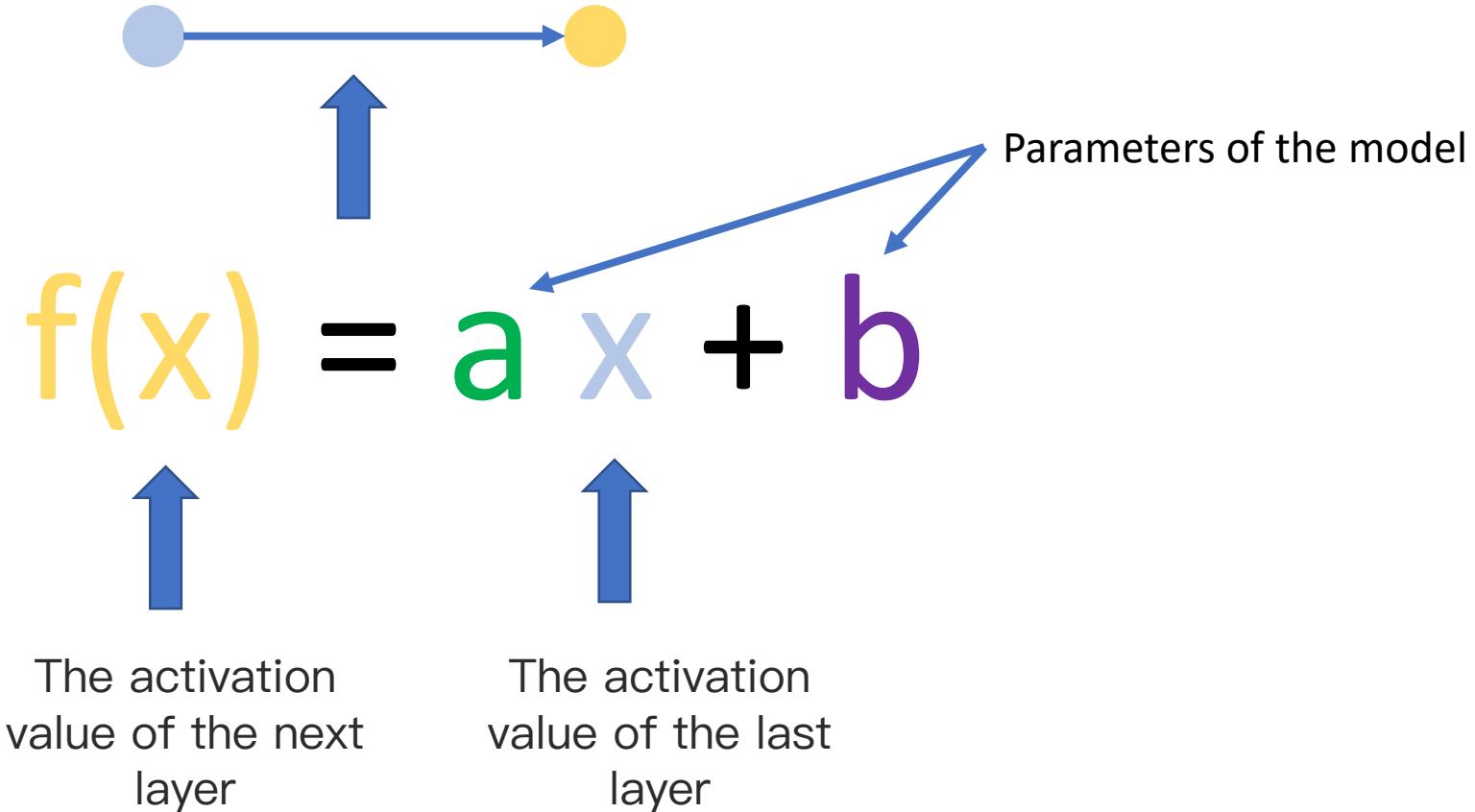


Neure  
(Frame to put figures in)

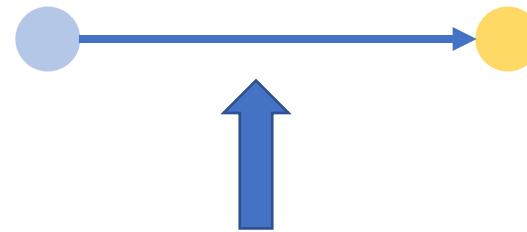
Connections  
(mapping function)

Figures in each layer  
is the "activation  
value" from the  
previous layer

# How to calculate the next layer?



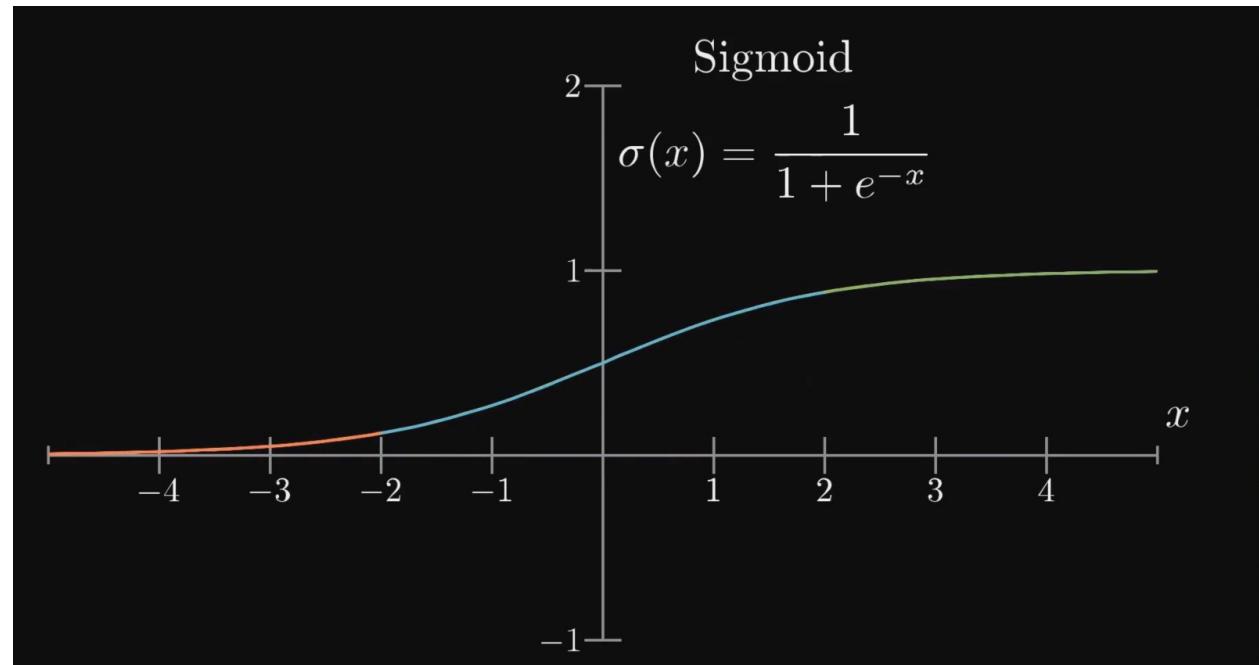
# How to calculate the next layer?



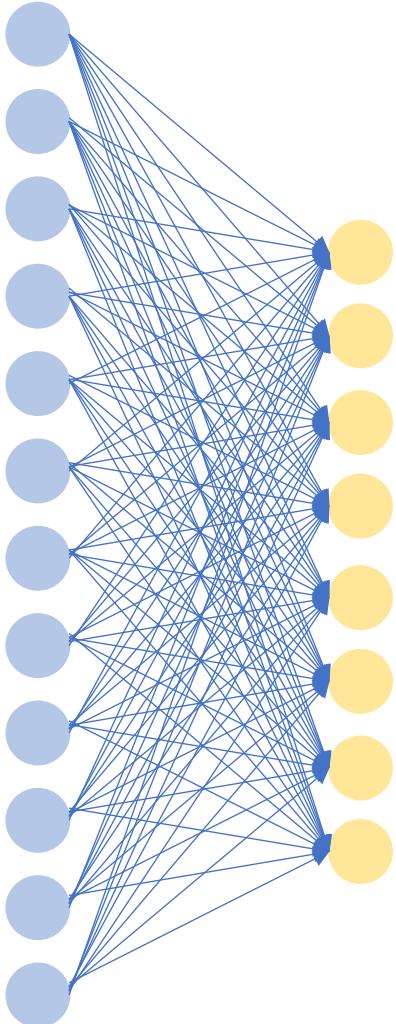
$$f(x) = \sigma(ax + b)$$



The activation  
function

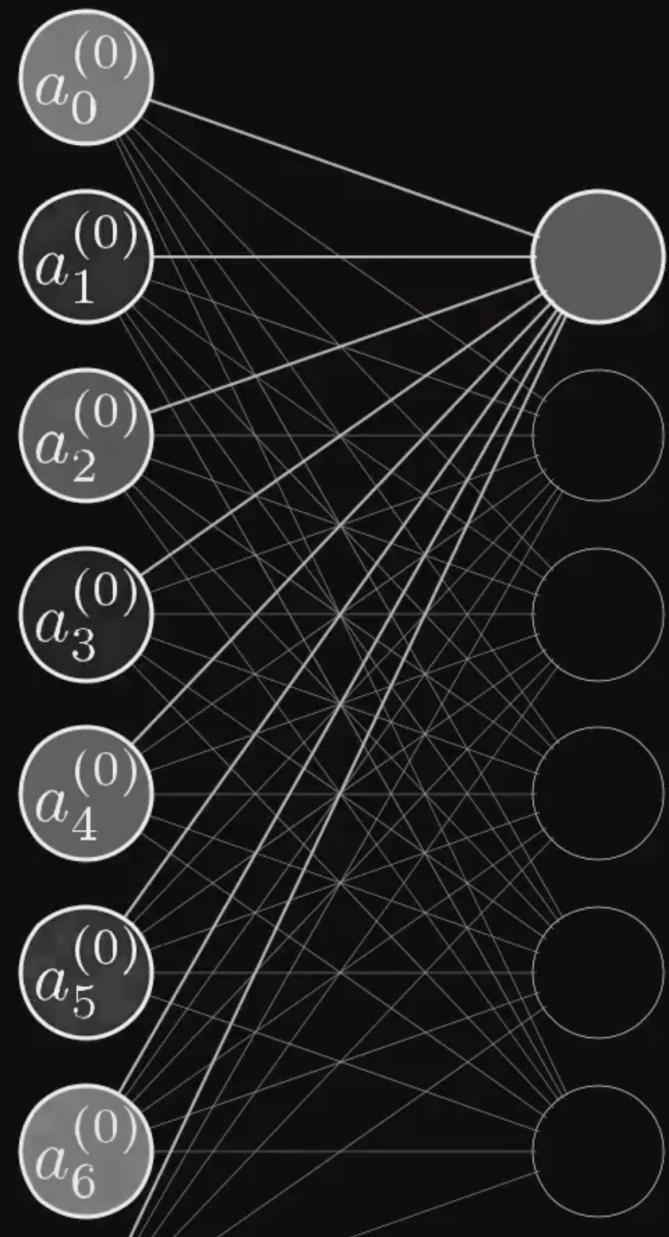


# How to calculate the next layer?



$$F(X) = \sigma(A X + B)$$

Essentially a ‘layer’ is a matrix representation of the sum of the function of each line



# Sigmoid

$$a_0^{(1)} = \sigma \left( \underbrace{w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \cdots + w_{0,n} a_n^{(0)}}_{\text{Bias}} + b_0 \right)$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ \vdots \\ ? \end{bmatrix}$$

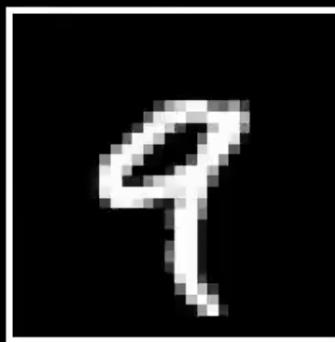
**Deep Neural Network :**  
Feature extraction + Pattern modeling

$$\text{g} = \text{o} + \text{l}$$

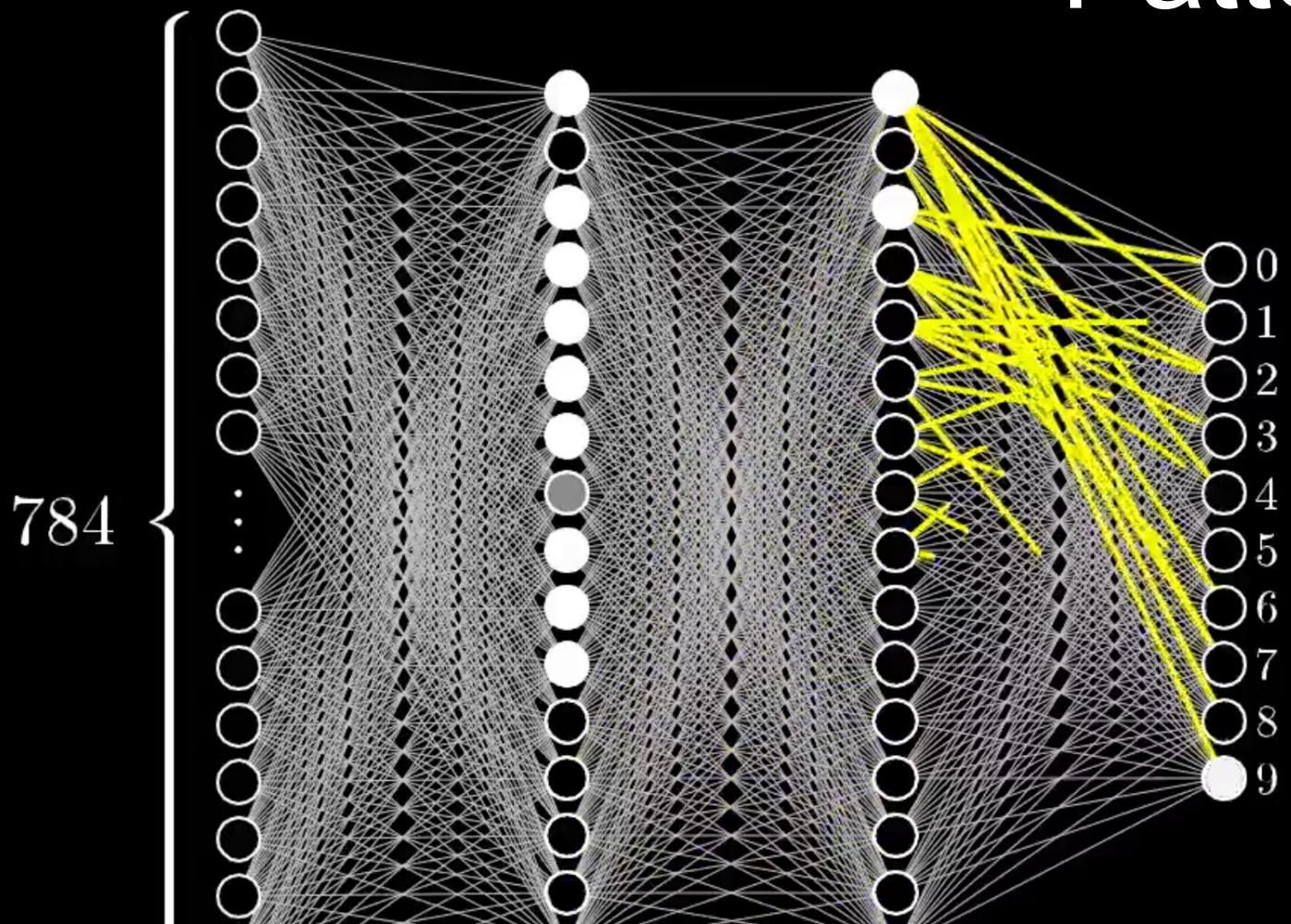
Feature

$$\text{g} = \text{o} + \text{g}$$

$$\text{g} = \text{l} + \text{i} + \text{r}$$

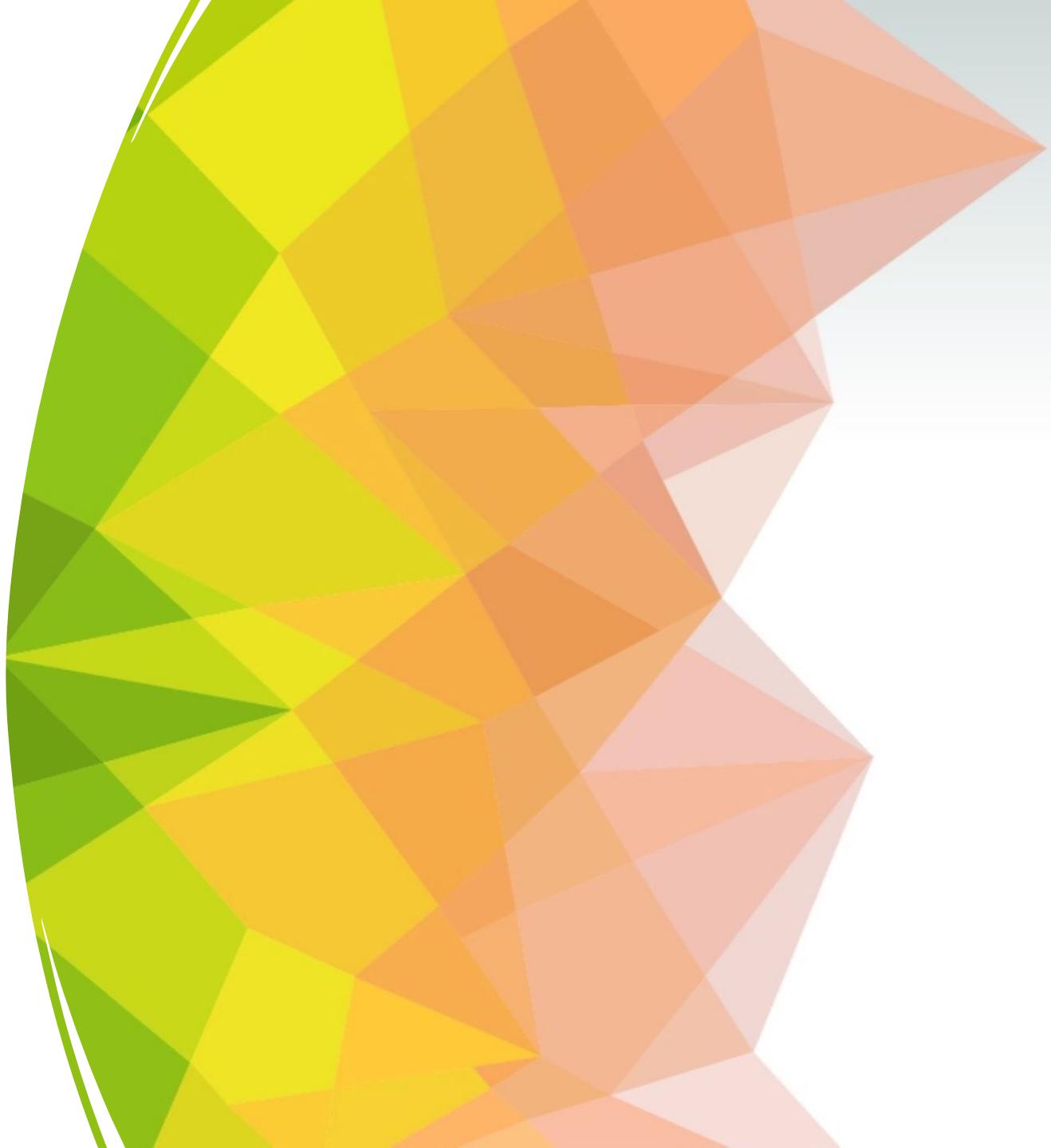


Pattern



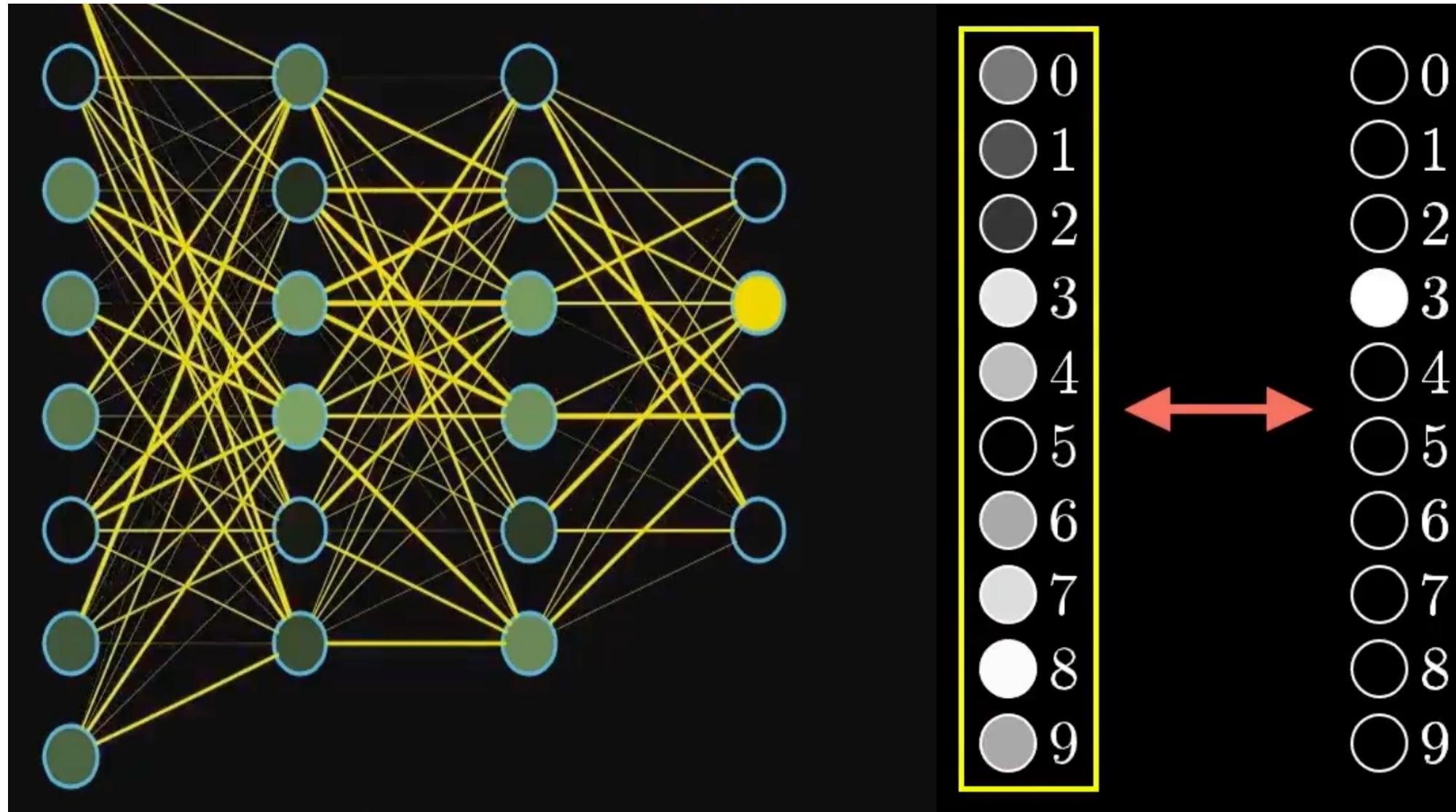
# How To Deep Learning

Supervision→Training



(**1**, 8) (**6**, 6) (**3**, 3) (**7**, 7) (**5**, 5) (**8**, 8) (**0**, 0) (**9**, 9)  
(**1**, 1) (**0**, 0) (**3**, 3) (**1**, 1) (**2**, 2) (**8**, 8) (**3**, 3) (**3**, 3)  
(**6**, 6) (**4**, 4) (**7**, 7) (**2**, 2) (**0**, 0) (**6**, 6) (**2**, 2) (**7**, 7)  
(**9**, 9) (**8**, 8) (**9**, 9) (**9**, 9) (**2**, 2) (**1**, 1) (**1**, 1) (**4**, 4)  
(**4**, 4) (**5**, 5) (**6**, 6) (**4**, 4) (**1**, 1) (**2**, 2) (**5**, 5) (**3**, 3)  
(**9**, 9) (**3**, 3) (**9**, 9) (**0**, 0) (**5**, 5) (**9**, 9) (**6**, 6) (**5**, 5)  
(**7**, 7) (**4**, 4) (**1**, 1) (**3**, 3) (**4**, 4) (**0**, 0) (**4**, 4) (**8**, 8)  
(**0**, 0) (**4**, 4) (**3**, 3) (**6**, 6) (**8**, 8) (**7**, 7) (**6**, 6) (**0**, 0)  
(**1**, 9) (**7**, 7) (**5**, 5) (**7**, 7) (**2**, 2) (**1**, 1) (**1**, 1) (**6**, 6)  
(**8**, 8) (**9**, 9) (**4**, 4) (**1**, 1) (**5**, 5) (**2**, 2) (**3**, 2) (**0**, 0)

Transform a realistic target into a mathematical description : loss



Prediction  $\longleftrightarrow$  Realistic

## L1 loss

也称Mean Absolute Error，简称MAE，计算实际值和预测值之间的绝对差之和的平均值。

表达式如下：

Pred	GT
0.1	0
0.2	0
0.3	1
0.1	0
0.0	0
0.1	0
0.2	0
...	...

$$\text{Loss}(\text{pred}, y) = \sum |y - \text{pred}|$$

y表示标签，pred表示预测值。

## L2 loss

也称为Mean Squared Error，简称MSE，计算实际值和预测值之间的平方差的平均值。

表达式如下：

$$\text{Loss}(\text{pred}, y) = \sum (y - \text{pred})^2$$

应用场合：对大部分回归问题，pytorch默认使用L2，即MSE。

## Cross-Entropy

此损失函数计算提供的一组出现次数或随机变量的两个概率分布之间的差异。它用于计算预测值与实际值之间的平均差异的分数。

表达式：

$$\text{loss}(\text{pred}, y) = - \sum y \log \text{pred}$$

应用场景：二分类及多分类。

特性：负对数似然损失不对预测置信度惩罚，与之不同的是，交叉熵惩罚不正确但可信的预测，以及正确但不太可信的预测。

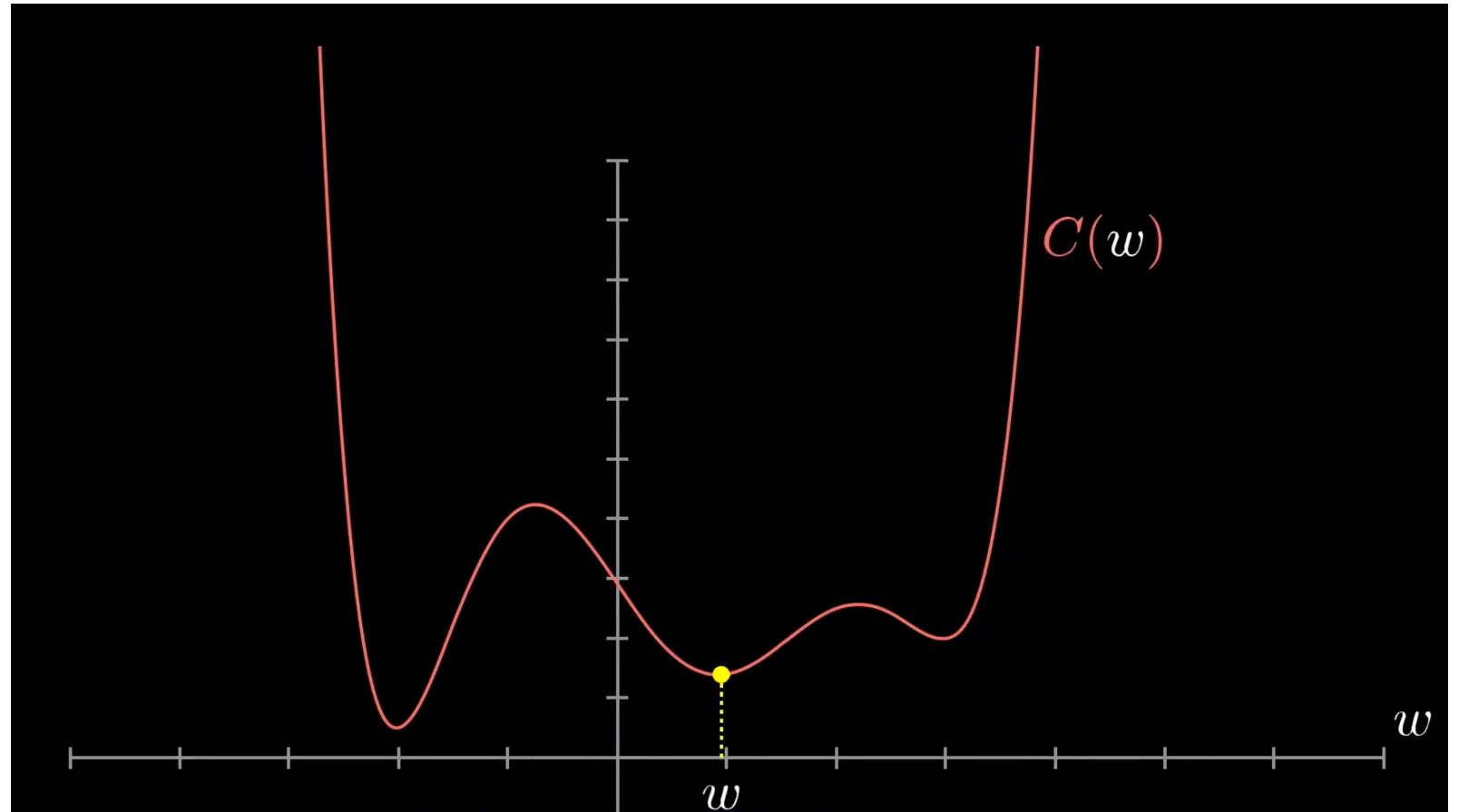
Transform a realistic target into a mathematical description

$L = \text{Optim}(\text{Model}, \text{Data}, \text{Label})$

$l = \text{Loss}(\text{Pred}, \text{GT})$

How to minimize loss?

Gradient

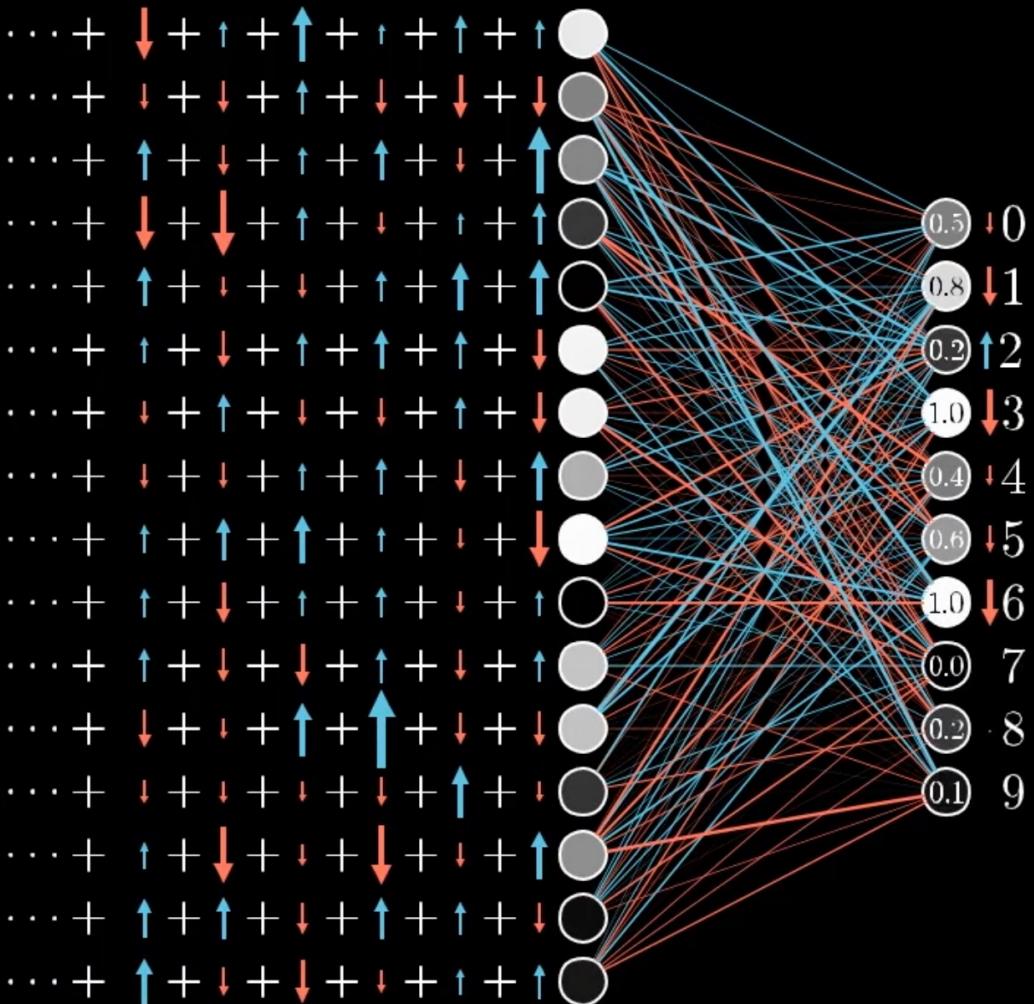




Increase  $b$

Increase  $w_i$   
in proportion to  $a_i$

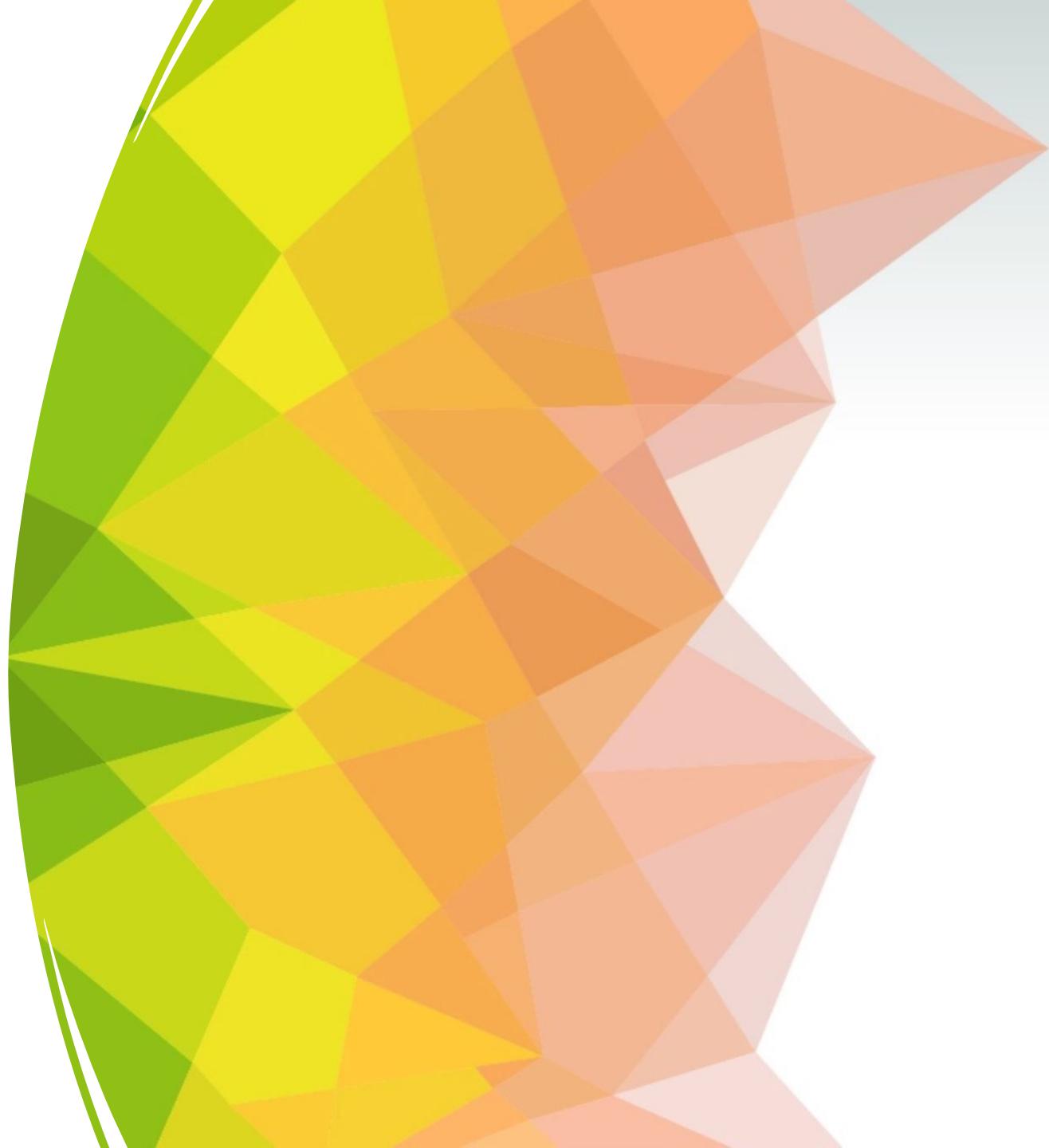
Change  $a_i$   
in proportion to  $w_i$



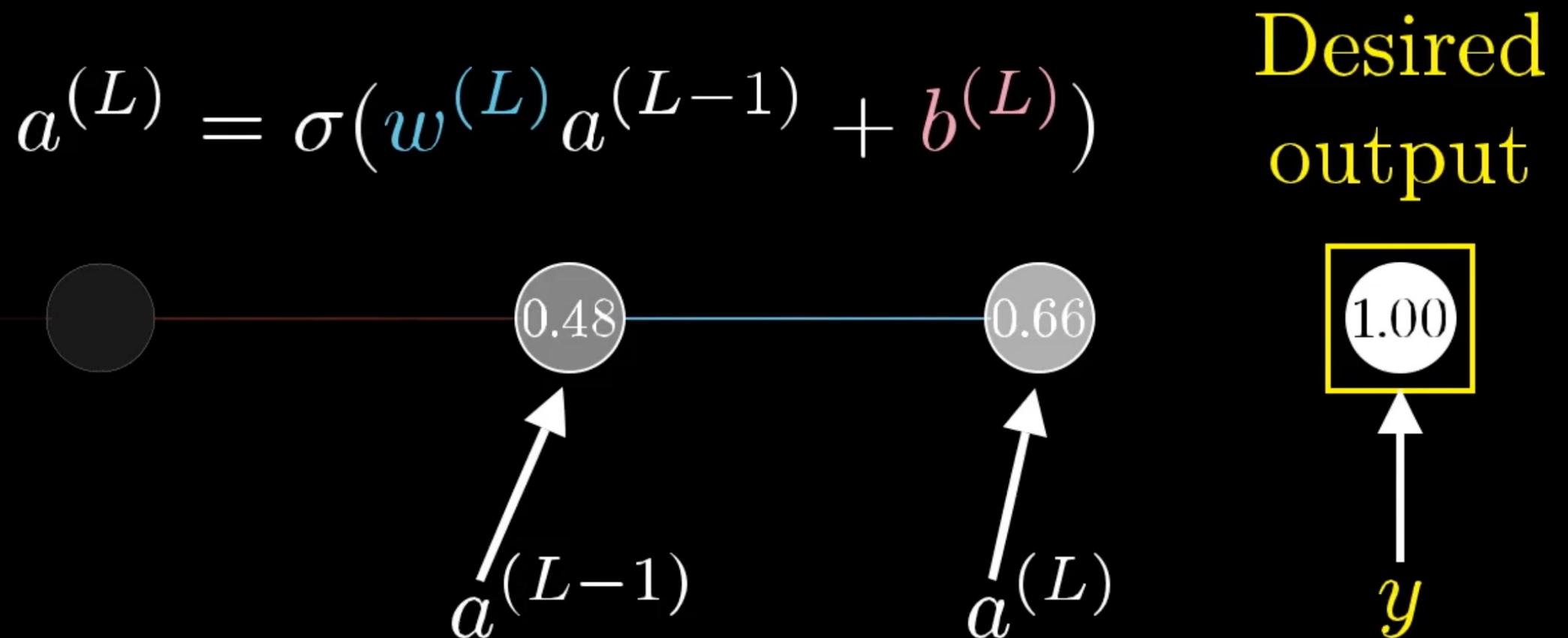
# How To Deep Learning

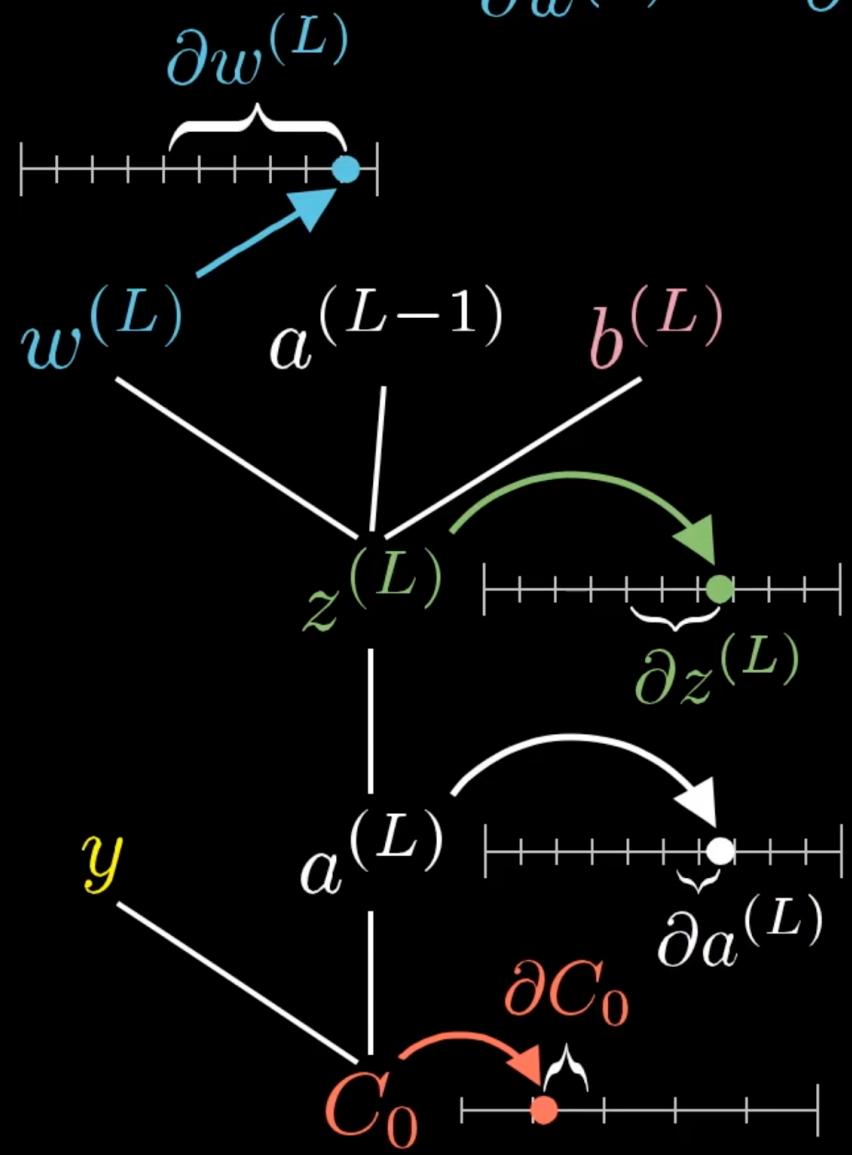
Supervision→Training

Back propagation(BP)



Cost  $\rightarrow C_0(\dots) = (a^{(L)} - y)^2$



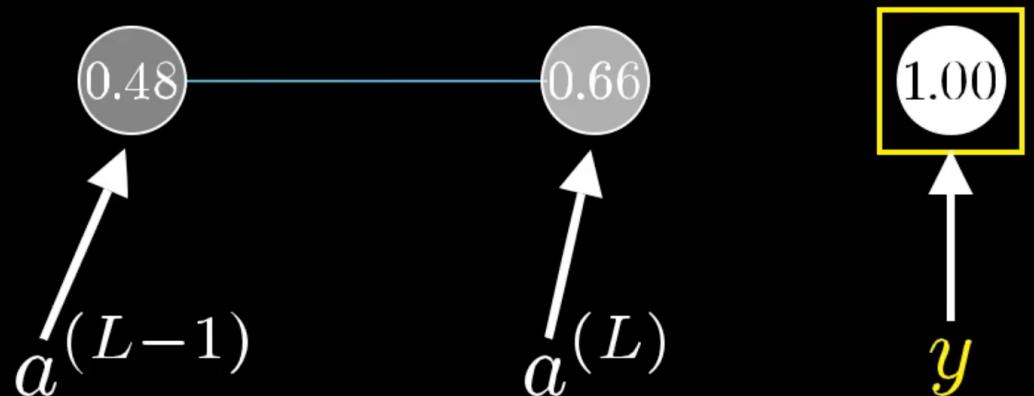


$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \quad C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

Desired output



$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$C_0 = (a^{(L)} - y)^2$$

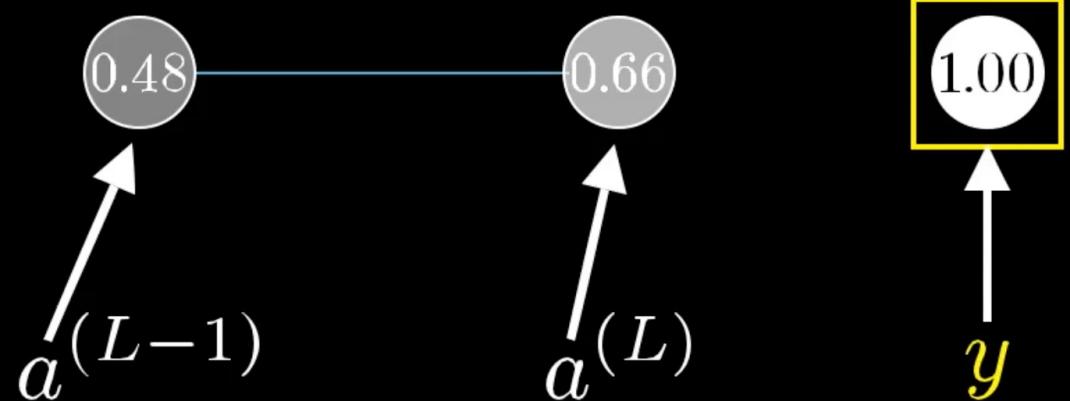
$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$$

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$



$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

Average of all  
training examples

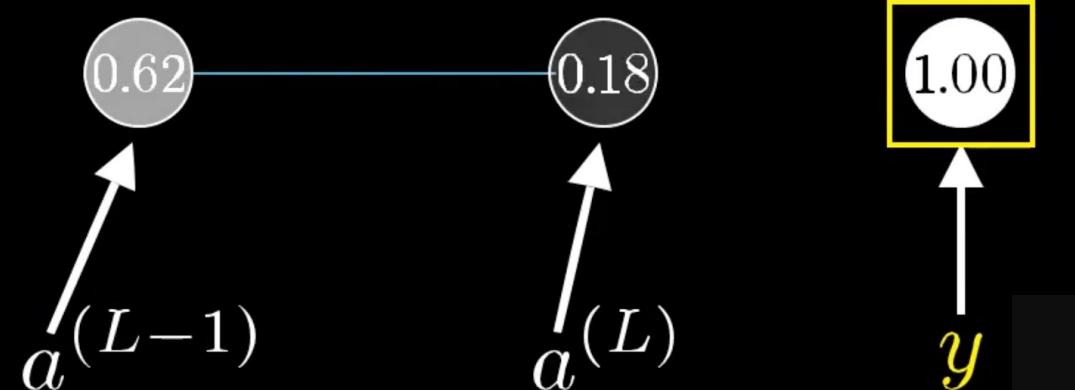
$$\underbrace{\frac{\partial C}{\partial w^{(L)}}}_{= \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$

Derivative of  
full cost function

$$C_0 = (a^{(L)} - y)^2$$

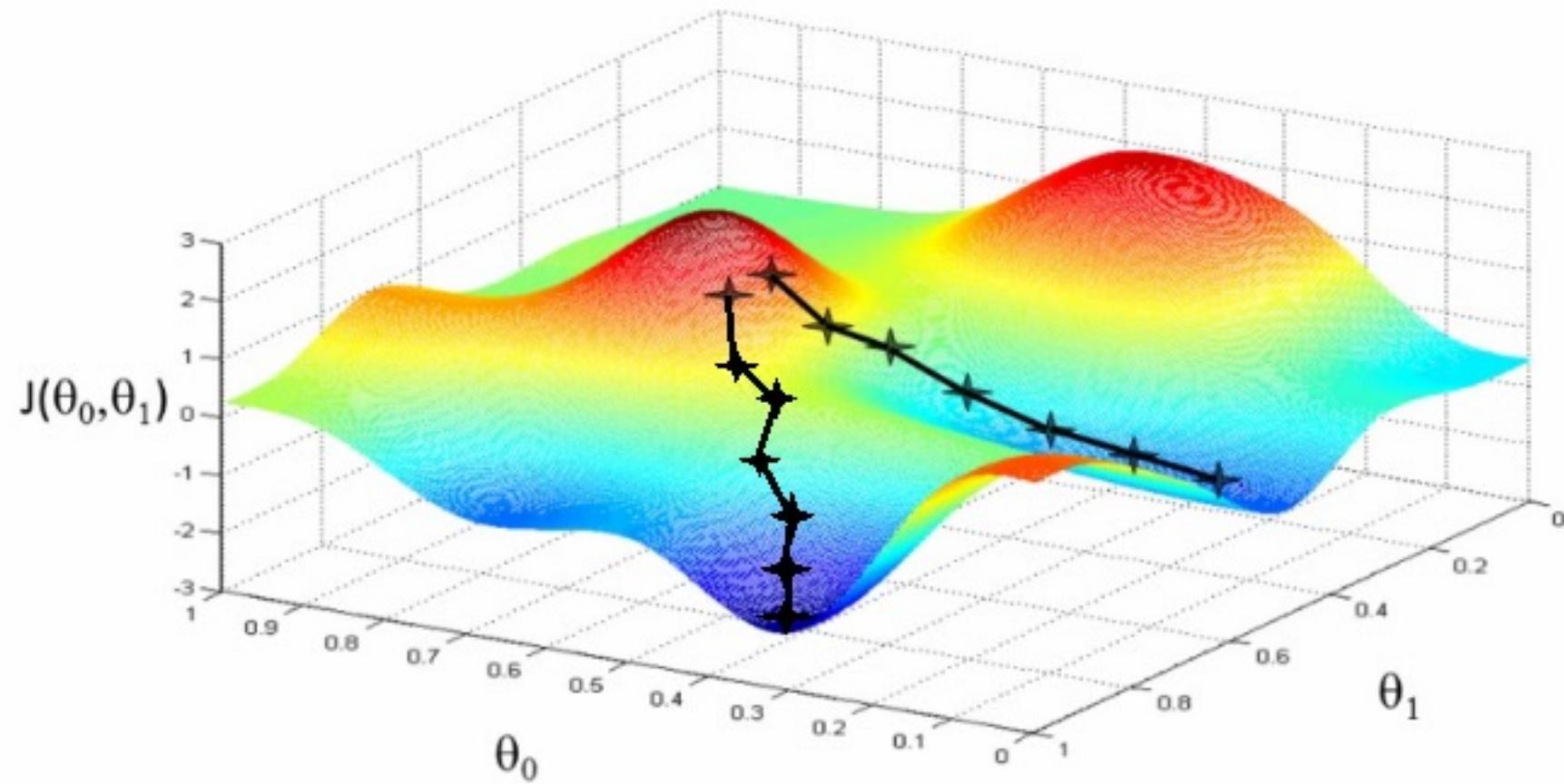
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



Optimizer: how to let loss drop move ?

# Stochastic gradient descent(SGD)



- 梯度下降法

梯度下降使用整个训练数据集来计算梯度，因此它有时也被称为批量梯度下降

下面就以均方误差讲解一下，假设损失函数如下：

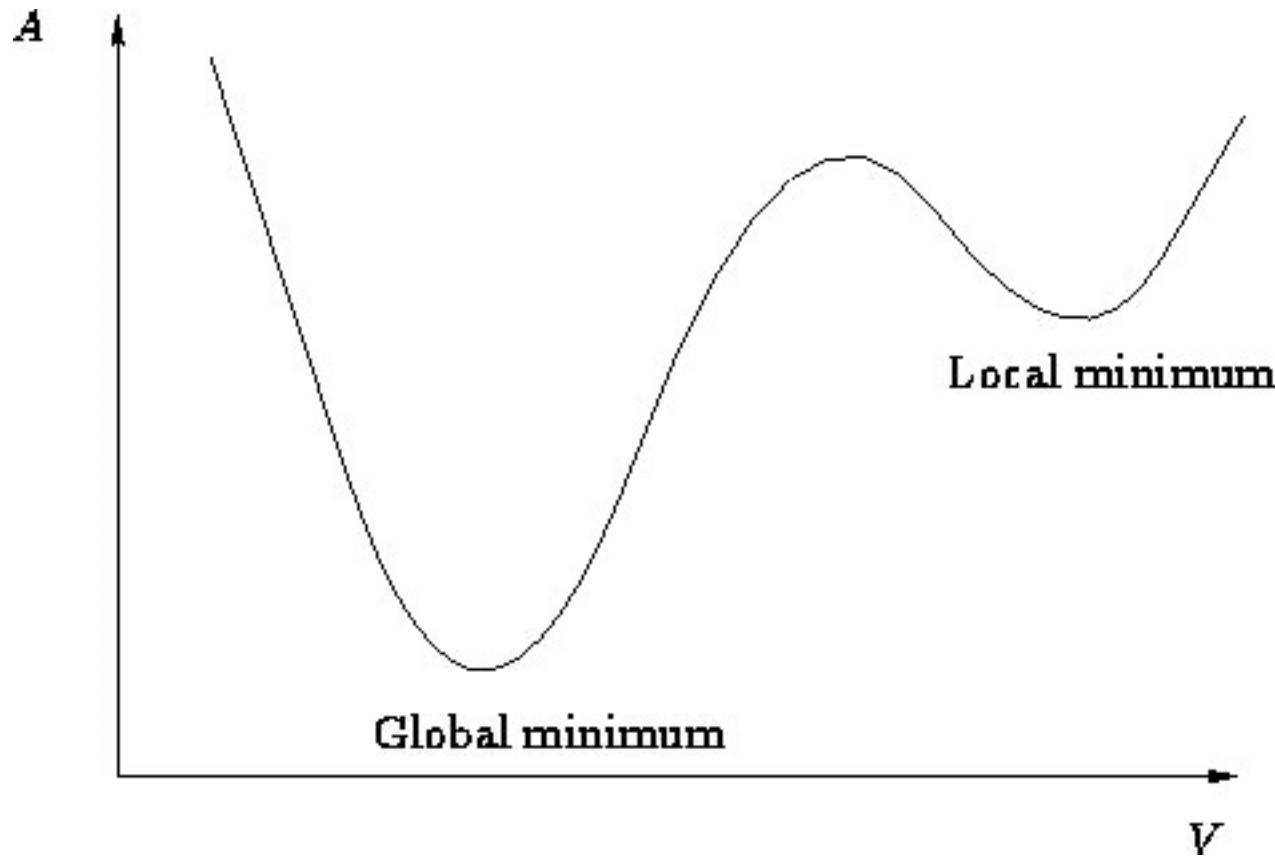
$$J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{m} \sum_m^{j=0} (\hat{y} - y)^2$$

其中  $\hat{y}$  是预测值， $y$  是真实值，那么要最小化上面损失  $J$ ，需要对每个参数  $\theta_0$ 、 $\theta_1$ 、 $\dots$ 、 $\theta_n$  运用梯度下降法：

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n)$$

其中  $\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n)$  是损失函数对参数  $\theta_i$  的偏导数、 $\alpha$  是学习率，也是每一步更新的步长。

# Stochastic gradient descent(SGD)



# Stochastic gradient descent(SGD)

在机器学习\深度学习中，目标函数的损失函数通常取各个样本损失函数的平均，那么假设目标函数为：

$$J(x) = \frac{1}{n} \sum_{i=1}^n J(x_i)$$

其中  $J(x_i)$  是第  $x_i$  个样本的目标函数，那么目标函数在在  $x$  处的梯度为：

$$\nabla J(x) = \frac{1}{n} \nabla \sum_{i=1}^n J(x_i)$$

如果使用梯度下降法(批量梯度下降法)，那么每次迭代过程中都要对  $n$  个样本进行求梯度，所以开销非常大，随机梯度下降的思想就是随机采样一个样本  $J(x_i)$  来更新参数，那么计算开销就从  $\mathcal{O}(n)$  下降到  $\mathcal{O}(1)$ 。

# Dataset partition: Train Val Test

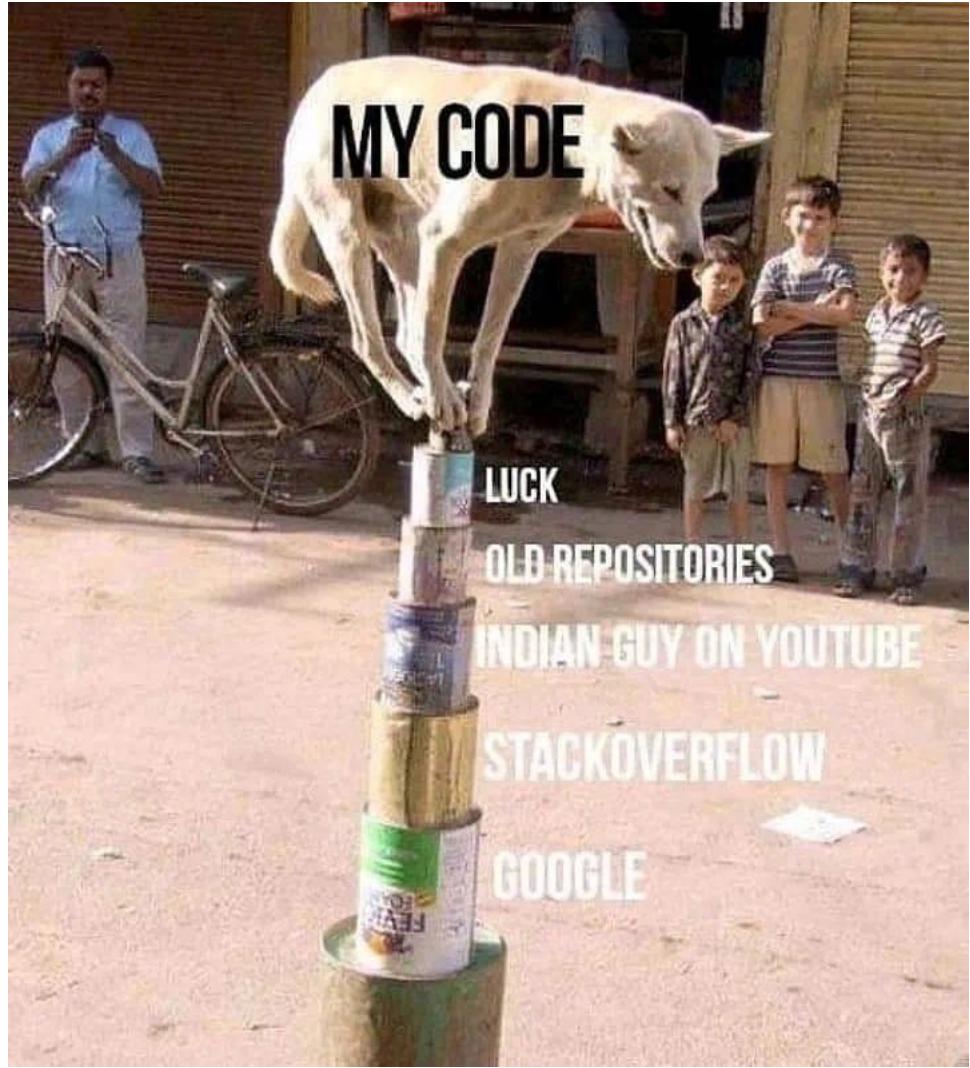
(**8**, 8) (**6**, 6) (**3**, 3)  
(**1**, 1) (**0**, 0) (**3**, 3)  
(**6**, 6) (**4**, 4) (**7**, 7)  
(**9**, 9) (**8**, 8) (**9**, 9)  
(**4**, 4) (**5**, 5) (**6**, 6)  
(**9**, 9) (**3**, 3) (**9**, 9)  
(**7**, 7) (**4**, 4) (**1**, 1)  
(**0**, 0) (**4**, 4) (**3**, 3)  
(**9**, 9) (**7**, 7) (**5**, 5)  
(**8**, 8) (**9**, 9) (**4**, 4)

(**7**, 7) (**5**, 5)  
(**1**, 1) (**2**, 2)  
(**2**, 2) (**0**, 0)  
(**9**, 9) (**2**, 2)  
(**4**, 4) (**1**, 1)  
(**0**, 0) (**5**, 5)  
(**3**, 3) (**4**, 4)  
(**6**, 6) (**8**, 8)  
(**7**, 7) (**2**, 2)  
(**1**, 1) (**5**, 5)

(**8**, 8) (**0**, 0) (**9**, 9)  
(**8**, 8) (**3**, 3) (**3**, 3)  
(**6**, 6) (**2**, 2) (**7**, 7)  
(**1**, 1) (**1**, 1) (**4**, 4)  
(**2**, 2) (**5**, 5) (**3**, 3)  
(**9**, 9) (**6**, 6) (**5**, 5)  
(**0**, 0) (**4**, 4) (**8**, 8)  
(**7**, 7) (**6**, 6) (**0**, 0)  
(**1**, 1) (**1**, 1) (**6**, 6)  
(**2**, 2) (**2**, 2) (**0**, 0)

Generally use random partition by probability, so that each data set can reflect the distribution of the data. Use the training set to train the data, use the validation set to test the effect of training, and save the best model. Using the test set to test the results of the final test model.

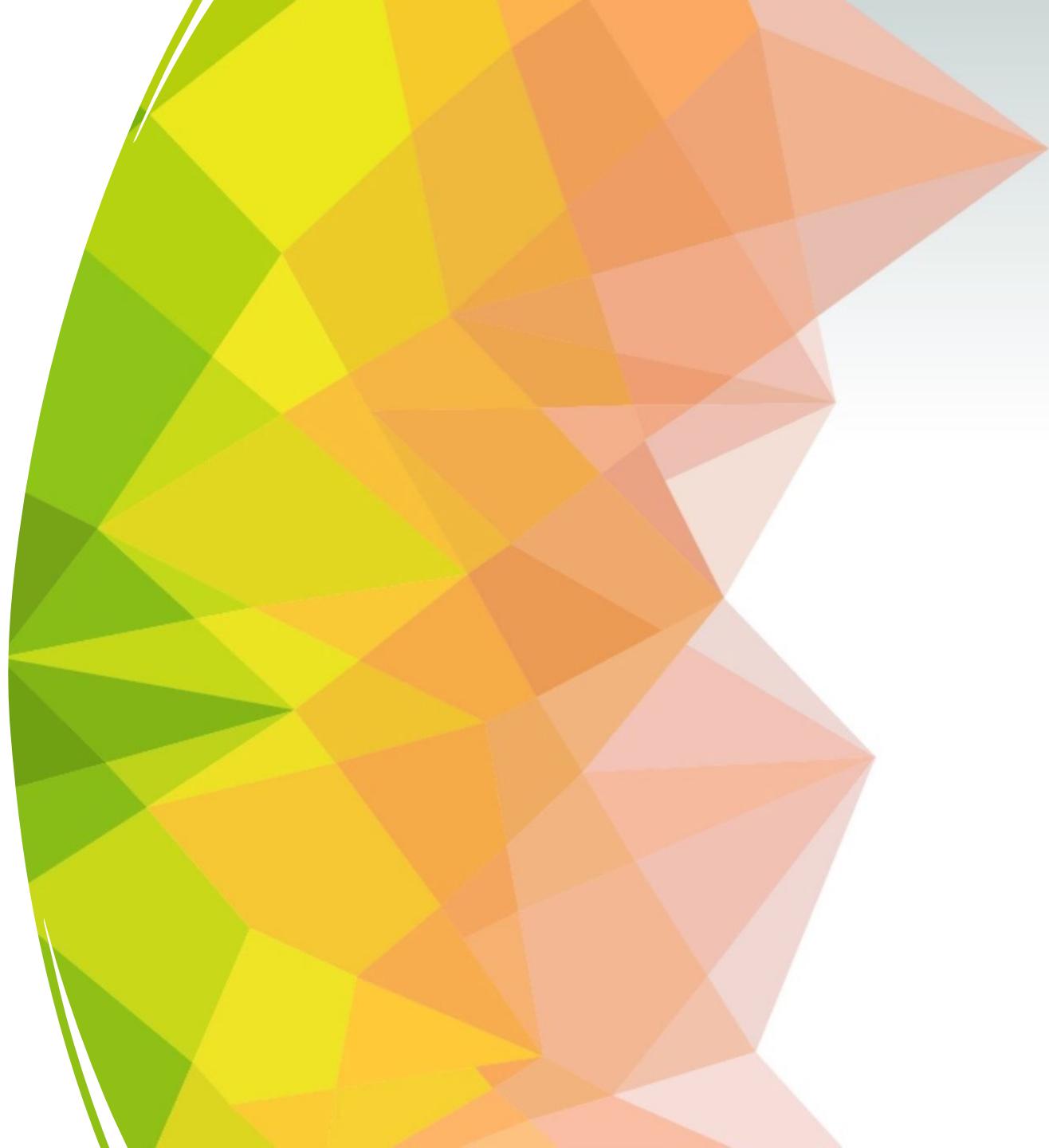
# Artificial→Intelligence



# Code Project ?

Supervision → Training

PyTorch



Dataset (read data+data enhance dispose)

Dataloader (PyTorch's tool, in order to read and process data in multithread)

Model

Iteration

## Dataset (read data+data enhance dipose)

Dataloader (PyTorch's tool, in order to read and process data in multithread)

# Model

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 7, 5)
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size,
        # padding_mode='zeros')
        self.pool = nn.MaxPool2d(2, 2)
        # torch.nn.MaxPool2d(kernel_size, stride=None, padding=0,
        self.conv2 = nn.Conv2d(7, 16, 5)

        self.fc1 = nn.Linear(16 * 5 * 5, 120)

        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))

        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(-1, 16 * 5 * 5)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))

        x = self.fc3(x)
        return x
```

# Iteration:choose loss and optimizer

```
net = Net()

import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

# Iteration: iterate training model

```
def training(trainloader, net, optimizer, PATH):
    for epoch in range(2): # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data # inputs, labels = data[0].to(device), data[1].to(device) # gpu mode

            # zero the parameter gradients
            optimizer.zero_grad()

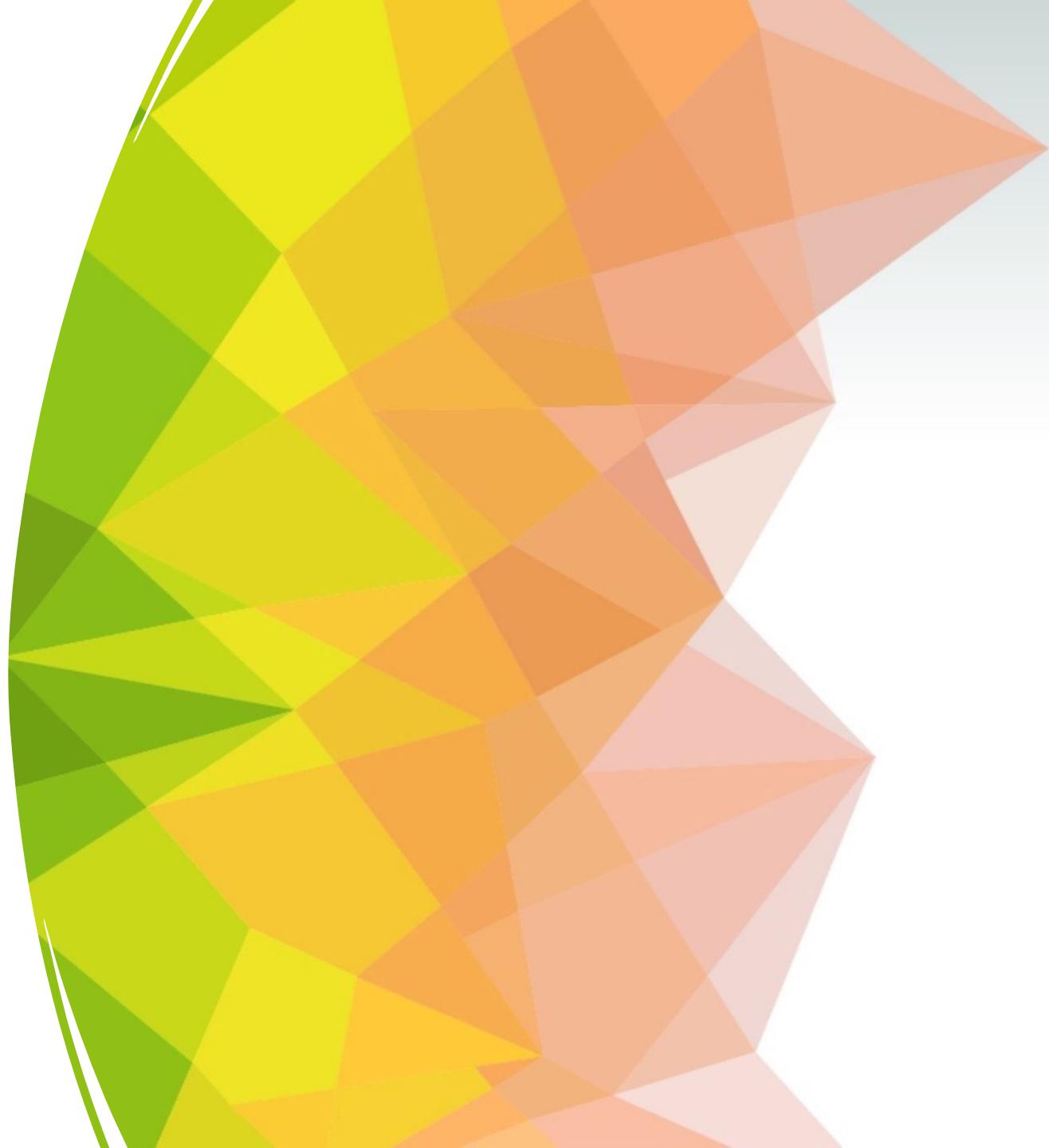
            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999: # print every 2000 mini-batches
                print('[%d, %5d] loss: %.3f' %
                      (epoch + 1, i + 1, running_loss / 2000))
                running_loss = 0.0

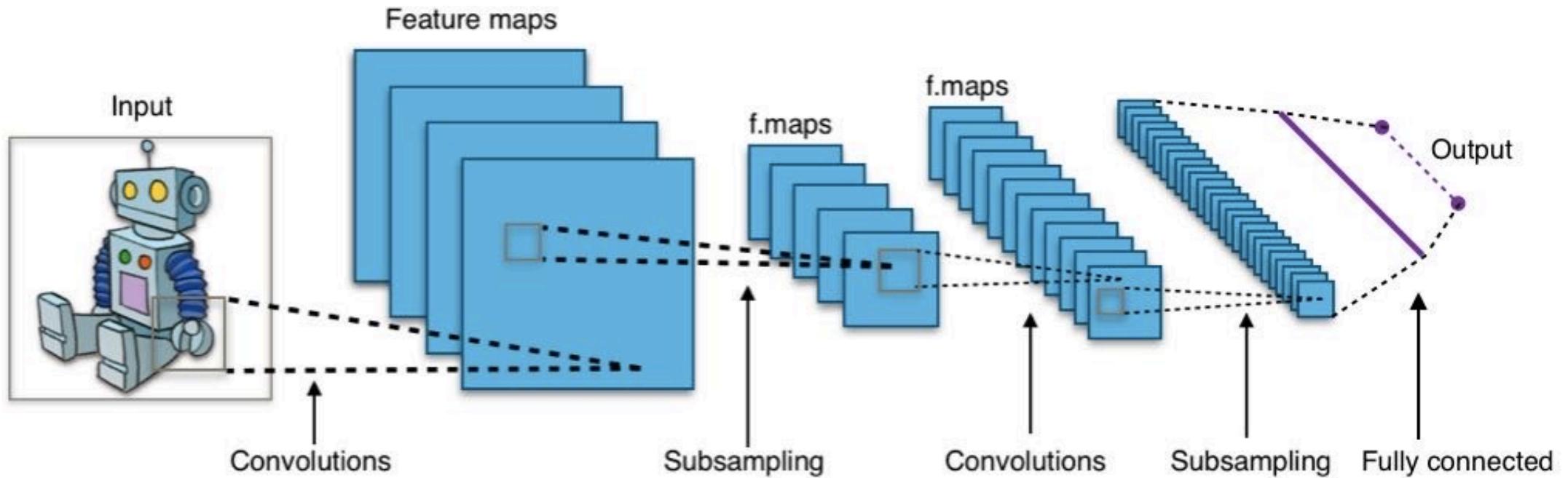
        print('Finished Training')
        # saving model
        torch.save(net.state_dict(), PATH) # 只保存模型参数
```

# How Do Deep Learning Become Stronger

Convolution  
Norm  
Various network structures?



Convolutional neural network (CNN) is a series of multi-layer structures, realizing the excavation of data information, constituting the model that can effectively discovery features and gradually transferring the results. CNN has long been one of the core algorithms in the field of image recognition, which can be used to extract discriminative features of images for classifier learning.

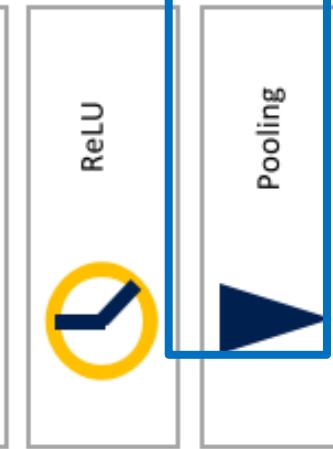
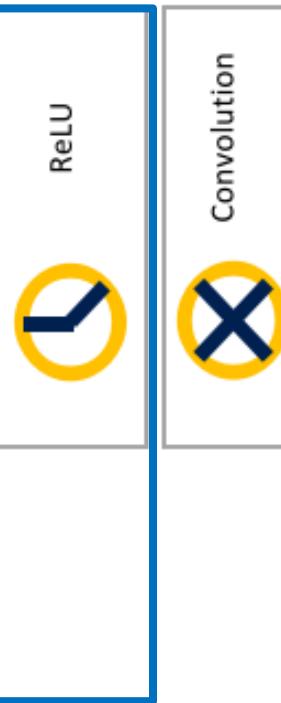


-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1

Convolution layer



Activation layer

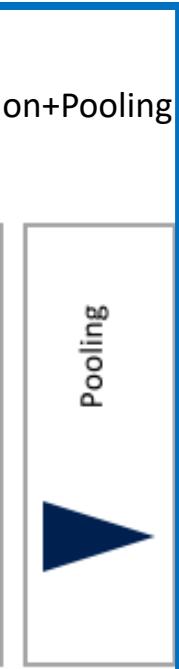
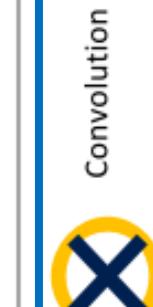


ReLU



Pooling layer

CNP strategy :  
Convolution+Activation+Pooling



1.00	0.55
0.55	1.00
1.00	0.55
0.55	0.55
0.55	1.00
1.00	0.55

# Convolution layer

Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)
x[:, :, 0]	w0[:, :, 0]
0 0 0 0 0 0 0 0 0 1 1 0 2 0 0 2 2 2 2 1 0 0 1 0 0 2 0 0 0 0 1 1 0 0 0 0 1 2 0 0 2 0 0 0 0 0 0 0 0	-1 1 0 0 1 0 0 1 1
x[:, :, 1]	w0[:, :, 1]
0 0 0 0 0 0 0 0 1 0 2 2 0 0 0 0 0 0 2 0 0 0 1 2 1 2 1 0 0 1 0 0 0 0 0 0 1 2 1 1 1 0 0 0 0 0 0 0 0	-1 -1 0 0 0 0 0 -1 0
x[:, :, 2]	w0[:, :, 2]
0 0 0 0 0 0 0 0 2 1 2 0 0 0 0 1 0 0 1 0 0 0 0 2 1 0 1 0 0 0 1 2 2 2 0 0 2 1 0 0 1 0 0 0 0 0 0 0 0	0 0 1 0 1 0 -1 0 1 -1 0 0
	Bias b0 (1x1x1) b0[:, :, 0]
	1

Filter W1 (3x3x3)	Output Volume (3x3x2)
w1[:, :, 0]	o[:, :, 0]
1 1 -1 -1 -1 1 0 -1 1	6 7 5 3 -1 -1 2 -1 4
w1[:, :, 1]	o[:, :, 1]
0 1 0 -1 0 -1 -1 1 0	2 -5 -8 1 -4 -4 0 -5 -5
w1[:, :, 2]	
-1 0 0 -1 0 1 -1 0 0	
Bias b1 (1x1x1) b1[:, :, 0]	0

toggle movement

Learning what (where is parameters)

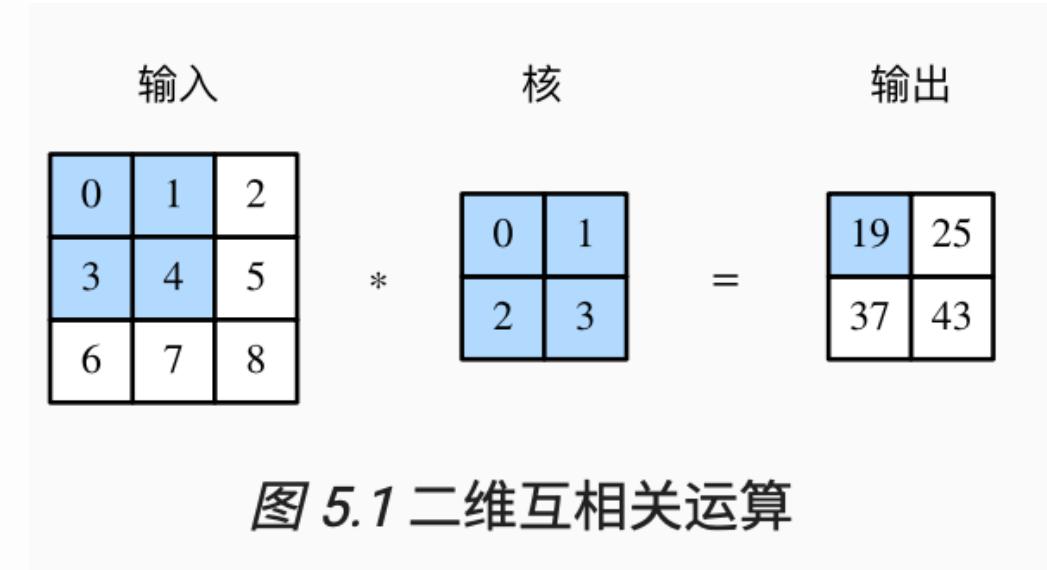
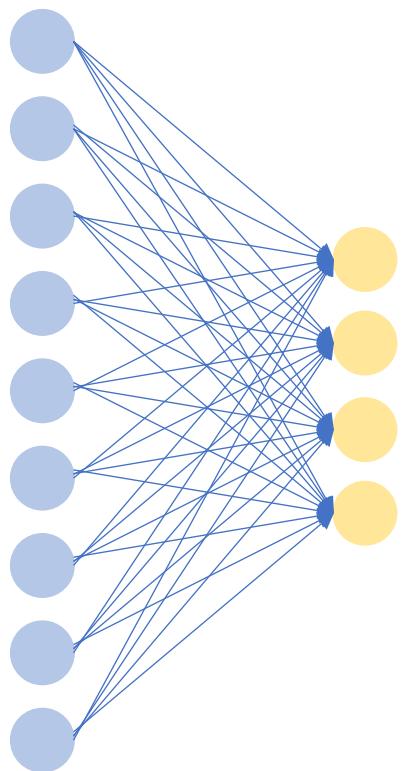


图 5.1 二维互相关运算

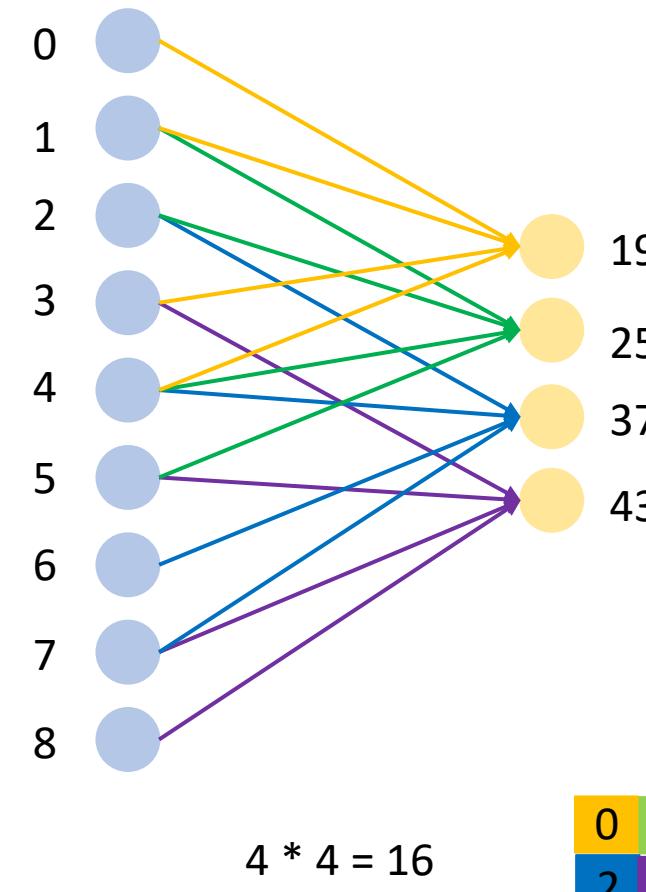
Kernal\_size, Stride, Padding

# What is the neurons of CNN looks like?

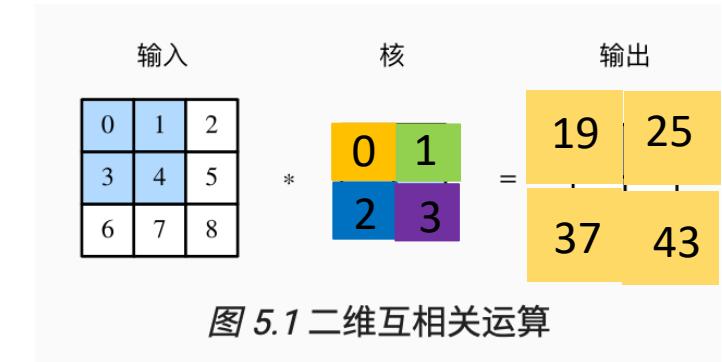


$$9 * 4 = 36$$

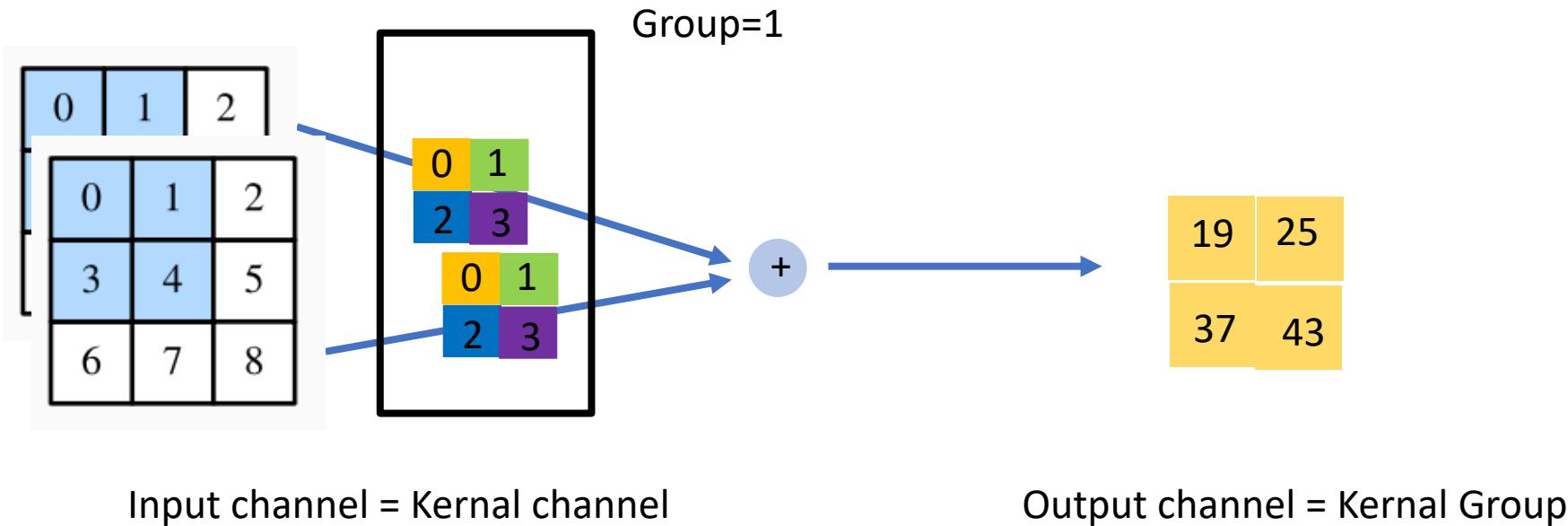
$$F(X) = \sigma(A X + B)$$



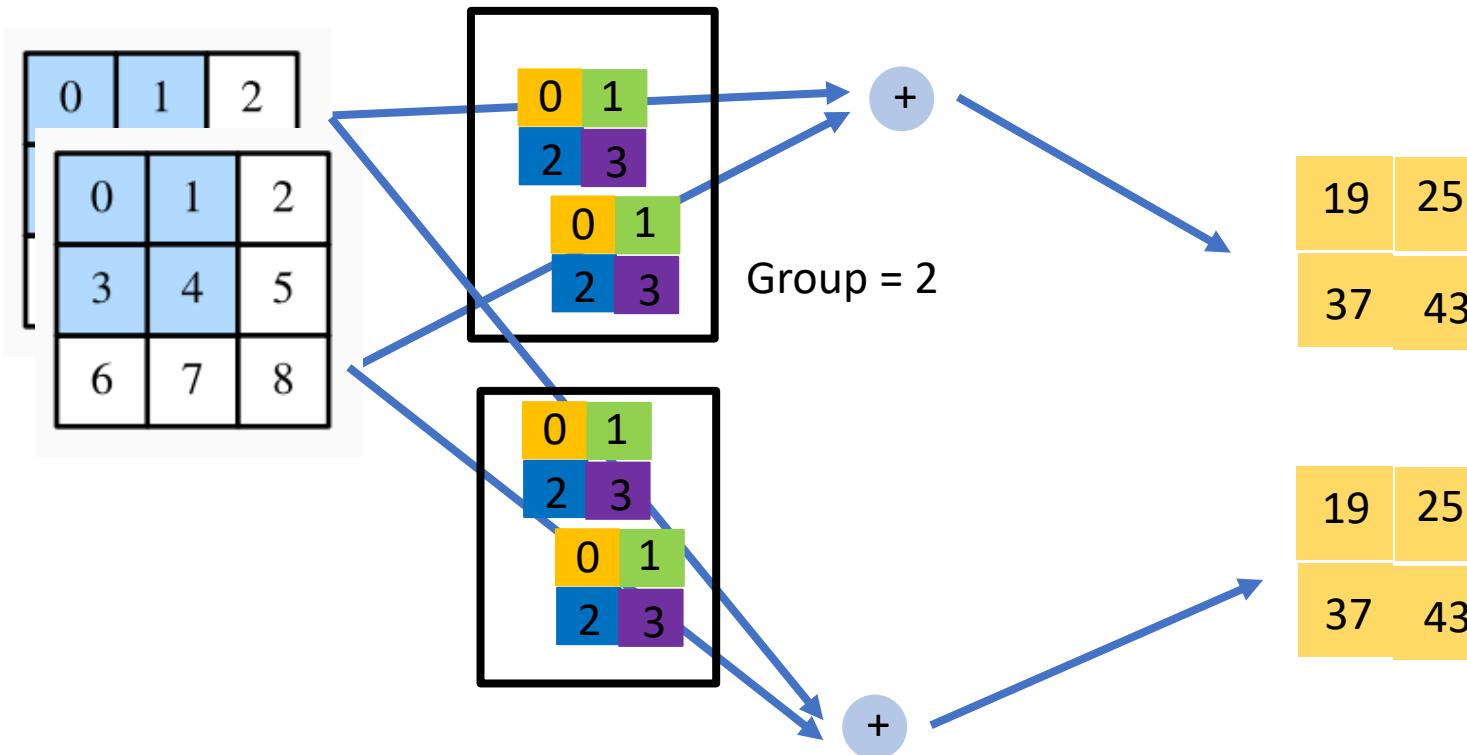
The number of real parameters is only 4



# What is CNN's channel ?



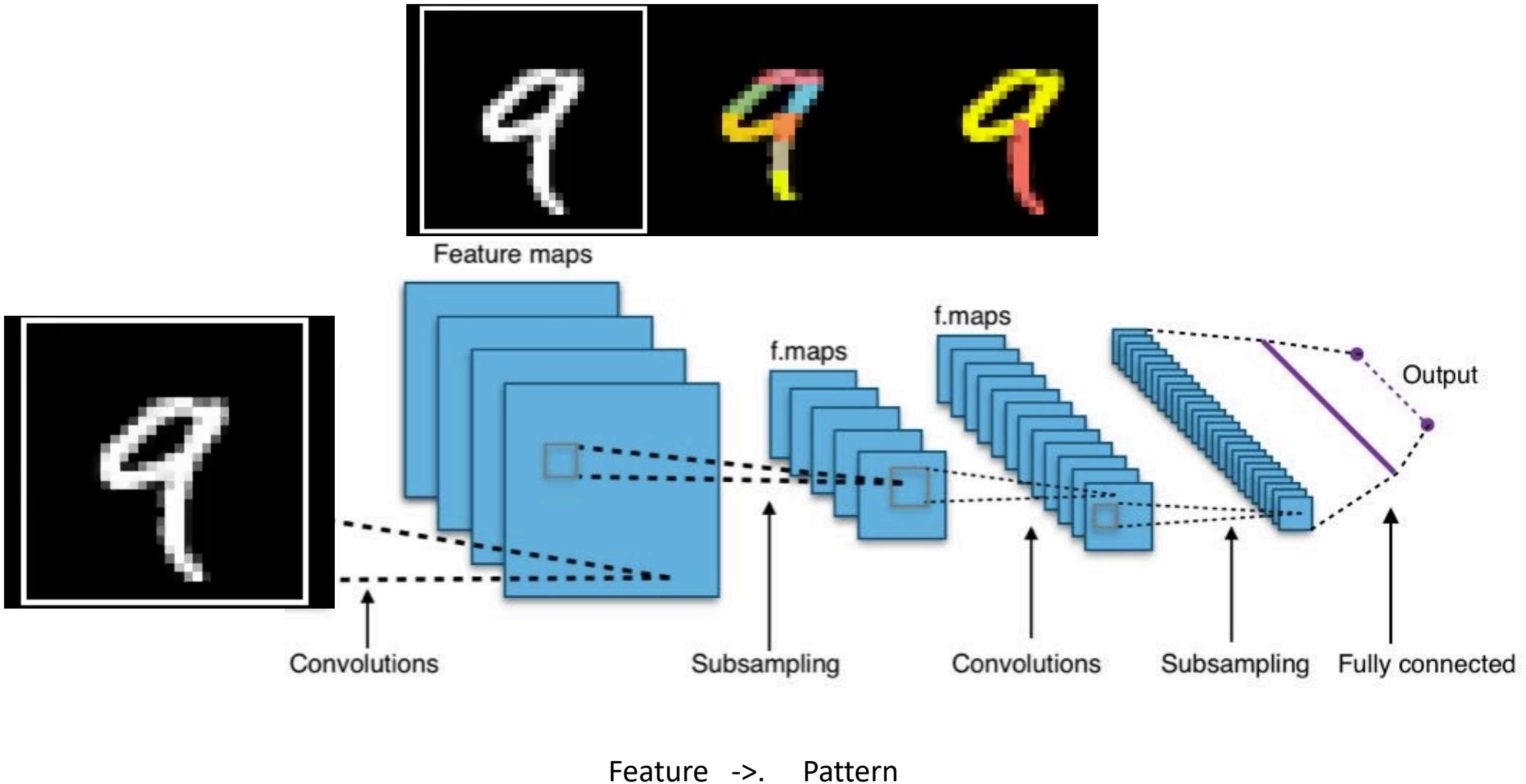
# What is CNN's channel ?



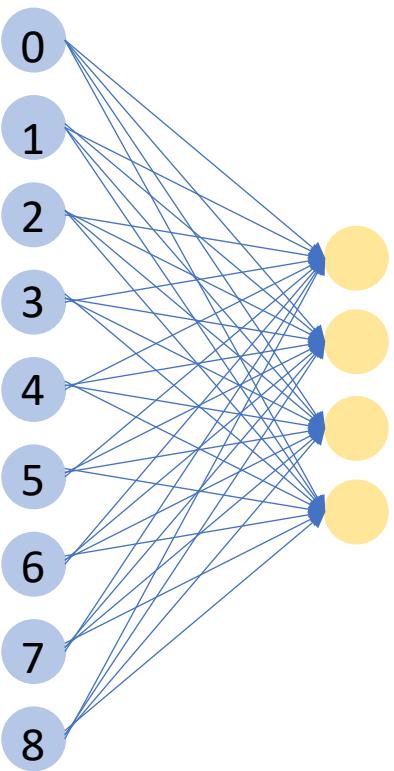
Input channel = Kernel channel

Output channel = Kernel Group

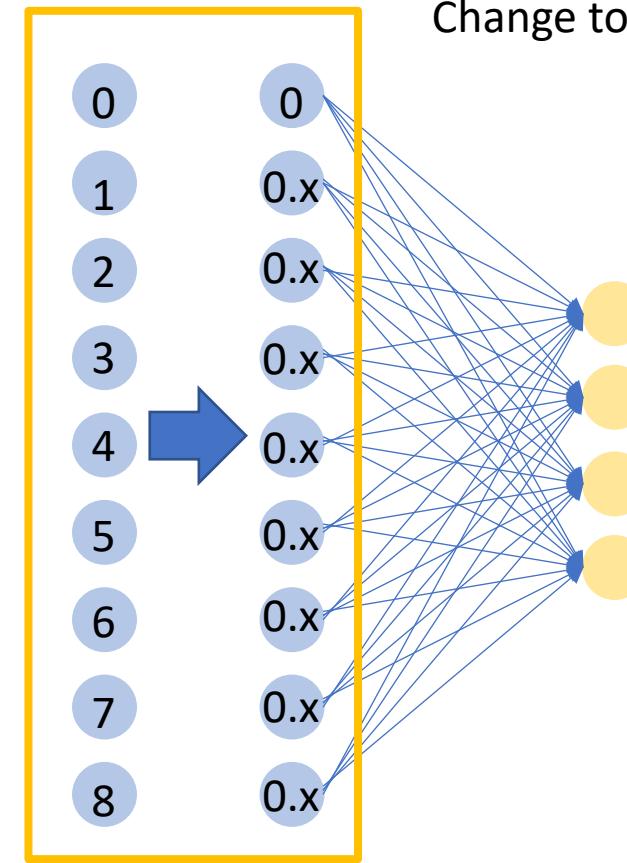
# Receptive field, deep purpose



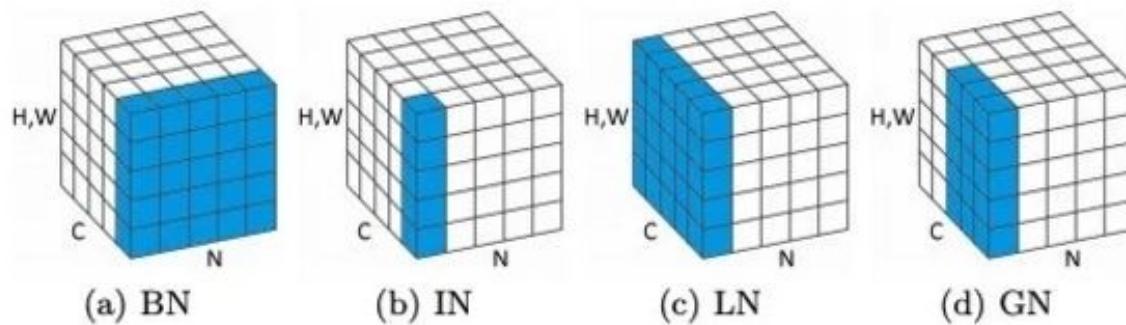
# Norm layer



Date changes



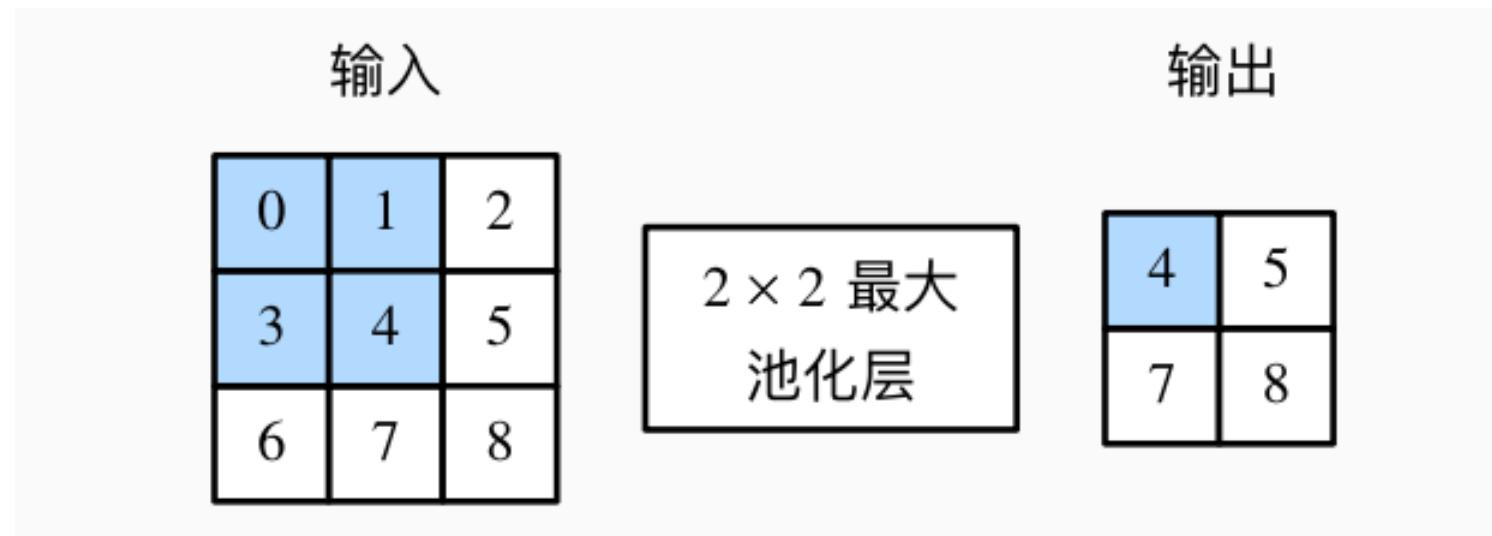
Change to Gaussian distribution



Change to whom?  
Different Norm layer  
(Different Norm Strategies)

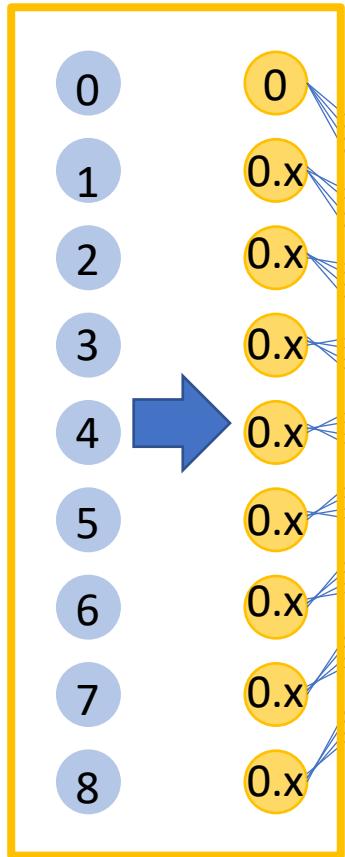
# Pooling layer

Data changes

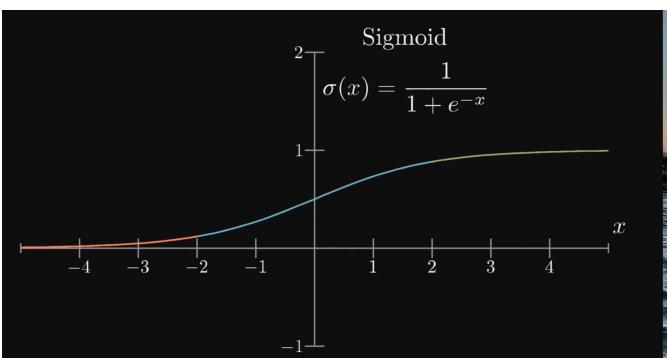


Average pooling/Max pooling

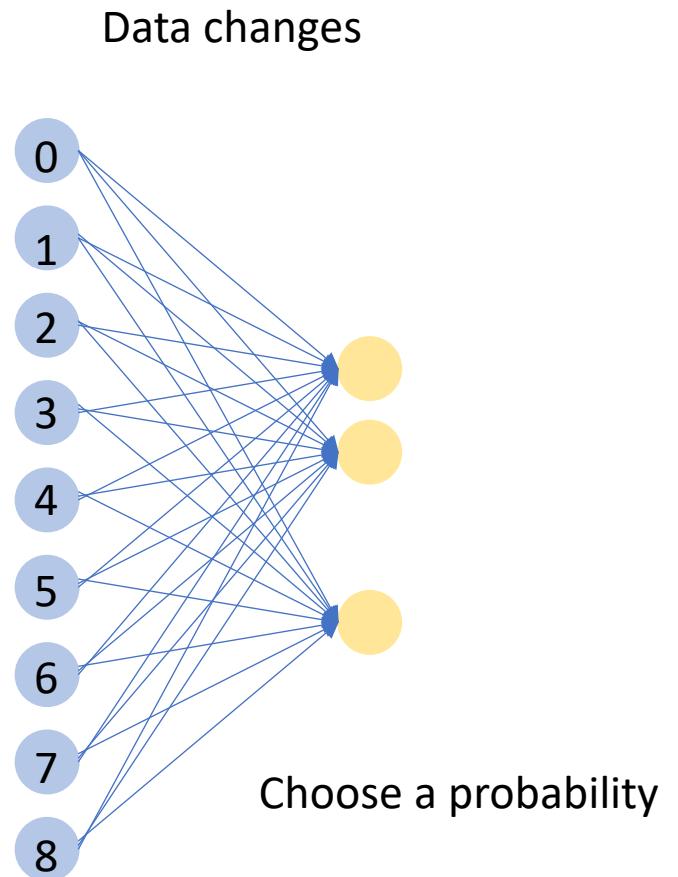
# Activation layer and Dropout



$$F(X) = \sigma(X)$$



Choose a strategy

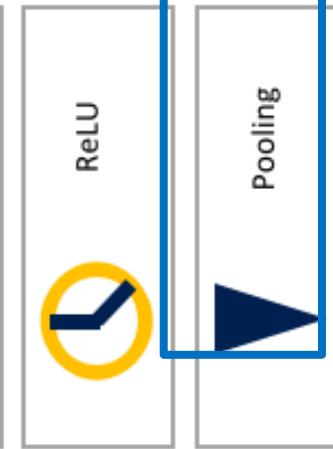
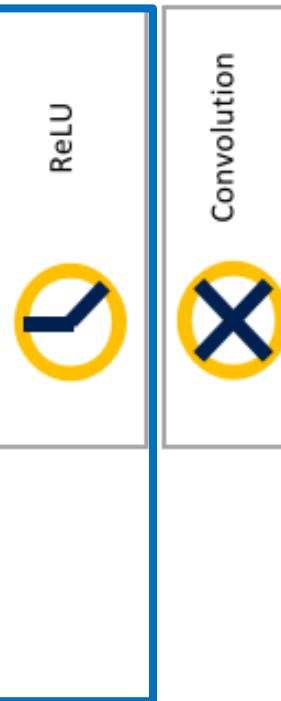


-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1

Convolution layer



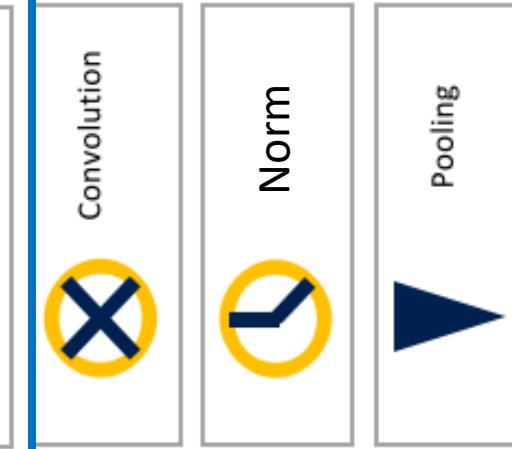
Activation layer



Pooling layer



CNP strategy :  
Convolution+Activation+Pooling



1.00	0.55
0.55	1.00

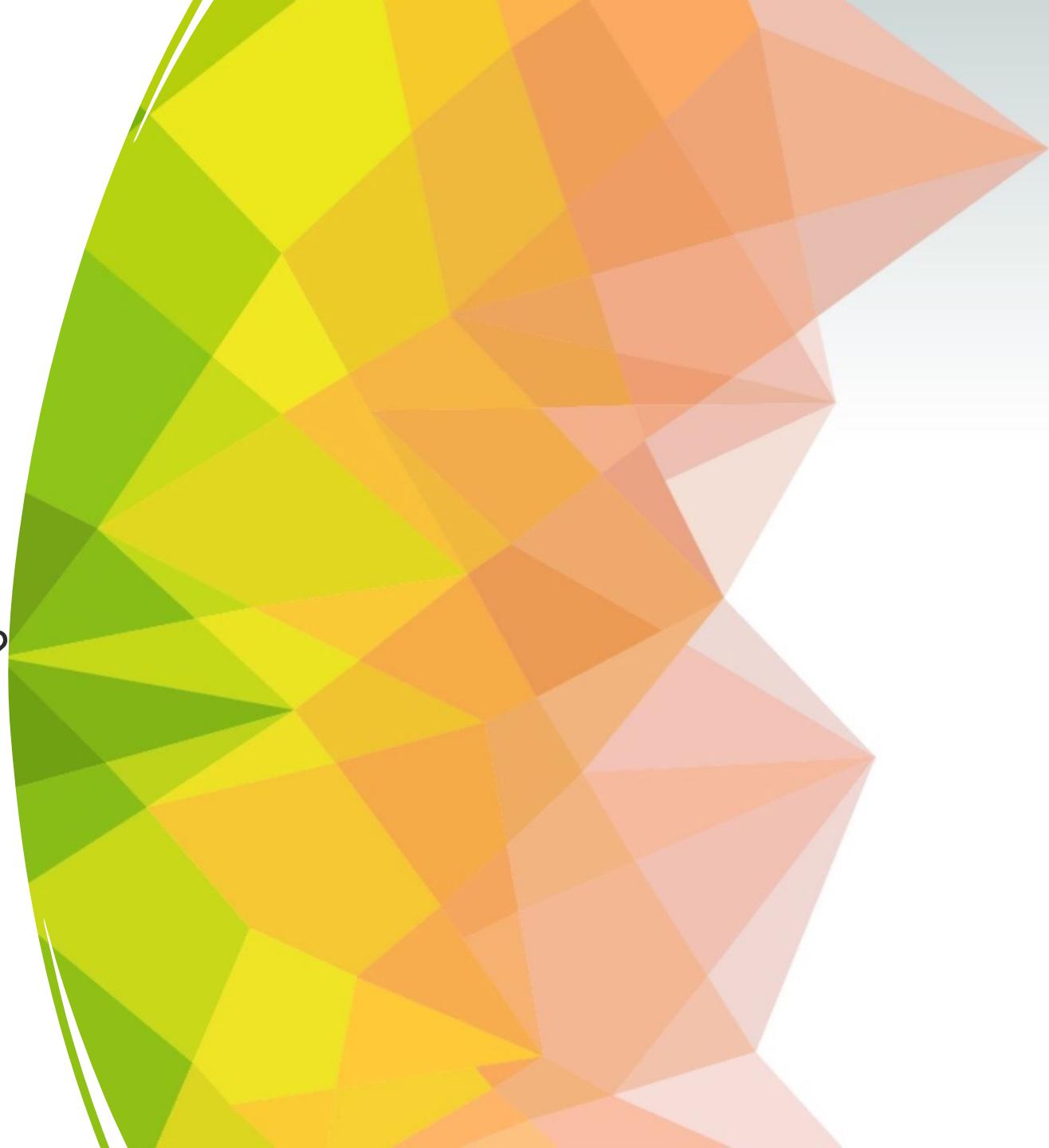
1.00	0.55
0.55	0.55

0.55	1.00
1.00	0.55



# What Is Deep Learning Study?

Various network structures?  
Training/optimization strategy?  
Initializing strategy?  
Excavating new tasks?  
Explanatory research?





我好菜啊

I am very vegetable





Keep  
Wandering

THX !

张天翊 Apr 3rd 2021