



# 3D Point Clouds

## Lecture 5 – Deep Learning with Point Clouds

主讲人 黎嘉信

Aptiv 自动驾驶  
新加坡国立大学 博士  
清华大学 本科





**1. Introduction to Deep Learning**



**2. PointNet**



**3. PointNet++**



## Intended Learning Outcomes

1

Explain the principles of the operations of different layers and training algorithms

2

Compare different neural network architectures

3

Implement popular neural networks

4

Solve problems using neural networks and deep learning techniques

# ARTIFICIAL INTELLIGENCE

IS NOT NEW

## ARTIFICIAL INTELLIGENCE

Any technique which enables computers to mimic human behavior



1950's

1960's

1970's

1980's

## MACHINE LEARNING

AI techniques that give computers the ability to learn without being explicitly programmed to do so



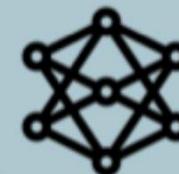
1990's

2000's

2010s

## DEEP LEARNING

A subset of ML which make the computation of multi-layer neural networks feasible



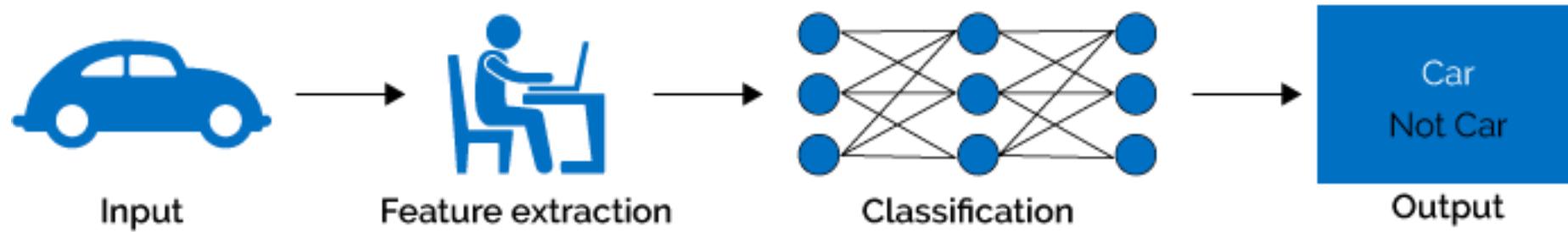
ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. |

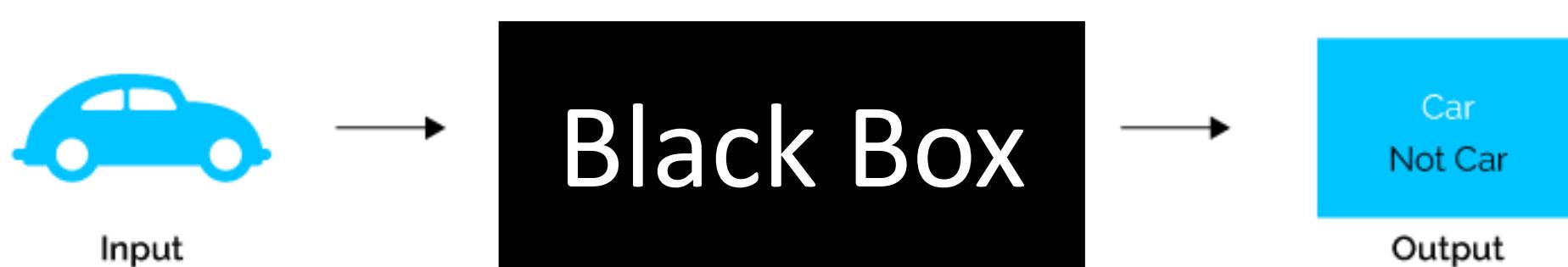
<https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning>



## Machine Learning



## Deep Learning





Everything is an optimization problem.

– Stephen Boyd

We can't really solve most optimization problems.

Deep Learning saves the world because it is a magical optimization solver.

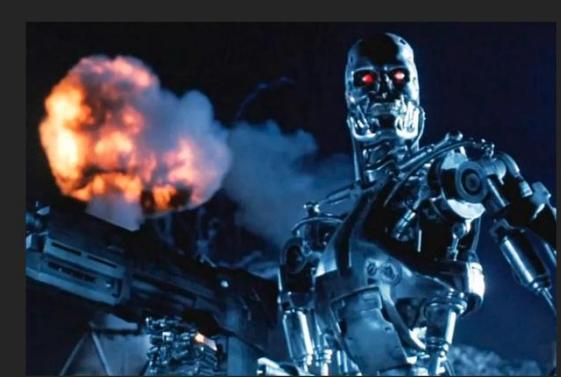
Stephen Boyd and  
Lieven Vandenberghe

# Convex Optimization

CAMBRIDGE



## Another explain of Deep Learning



What society thinks I do



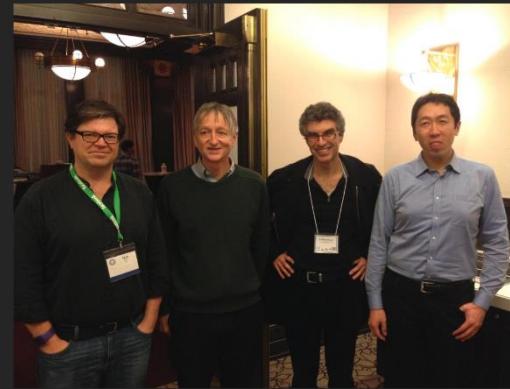
What my friends think I do



What other computer  
scientists think I do



What mathematicians think I do



What I think I do

```
from keras import *
```

What I actually do



- Deep learning is trying to solve one problem

$$\min_x f(x)$$

- Let's start from linear function

$$y = f(x) = w^T x + b$$

- Given a set of  $\{x_i \in \mathbb{R}^n, y_i \in \mathbb{R}\}$
- Solve  $w \in \mathbb{R}^n, b \in \mathbb{R}$



Consider the problem of house price prediction

A dataset consist of  $M$  instance

- Features  $x_i \in \mathbb{R}^n$
- Ground truth (price)  $y_i \in \mathbb{R}$

Unknown parameter  $w \in \mathbb{R}^n, b \in \mathbb{R}$

Denote prediction of one instance

$$\hat{y}_i = w^T x_i + b = \begin{bmatrix} w \\ b \end{bmatrix} [x_i \quad 1]$$

A bit abuse of notation with *homogeneous coordinate*

$$\hat{y}_i = w^T x_i$$

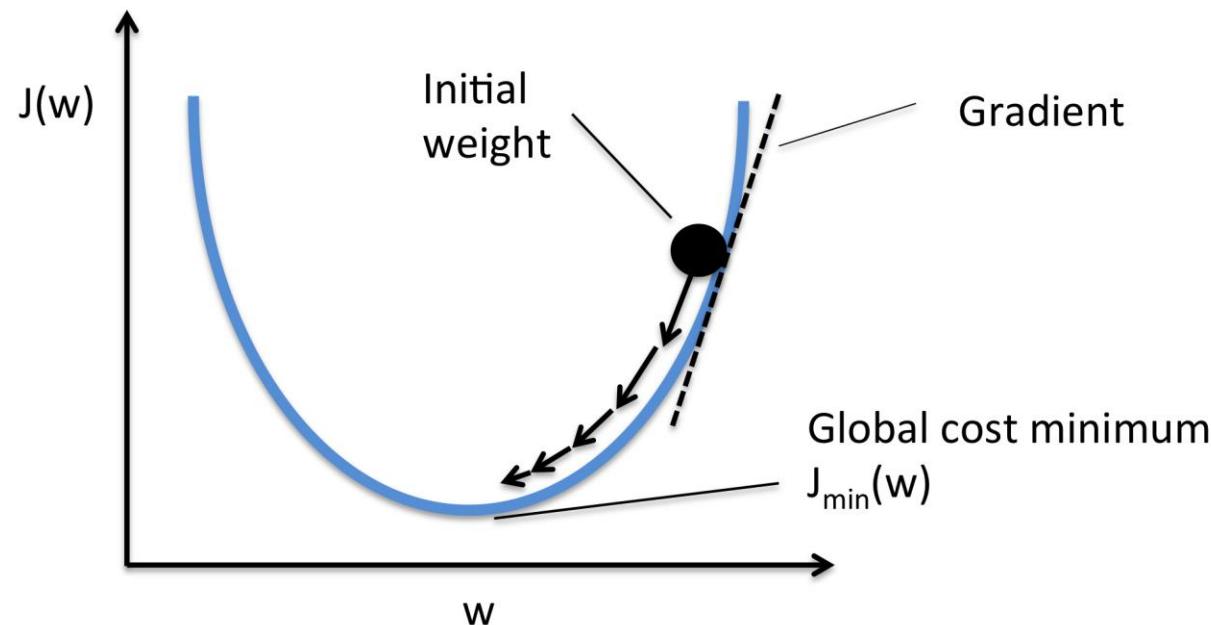
$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ \vdots \\ x_{in} \end{bmatrix} = \begin{bmatrix} size \\ floor \\ age \\ \vdots \\ * * * \end{bmatrix}$$



- It is a linear regression problem

$$w = \arg \min_w \sum_{i=1}^M \frac{1}{2} (w^T x_i - y_i)^2$$

- Typical  $Ax = b$  problem, but can we solve it by *Gradient Descent*?





## Gradient Descent

- For any differentiable function  $F(x): \mathbb{R}^n \rightarrow \mathbb{R}^m$
- The objective is to minimize  $F(x)$

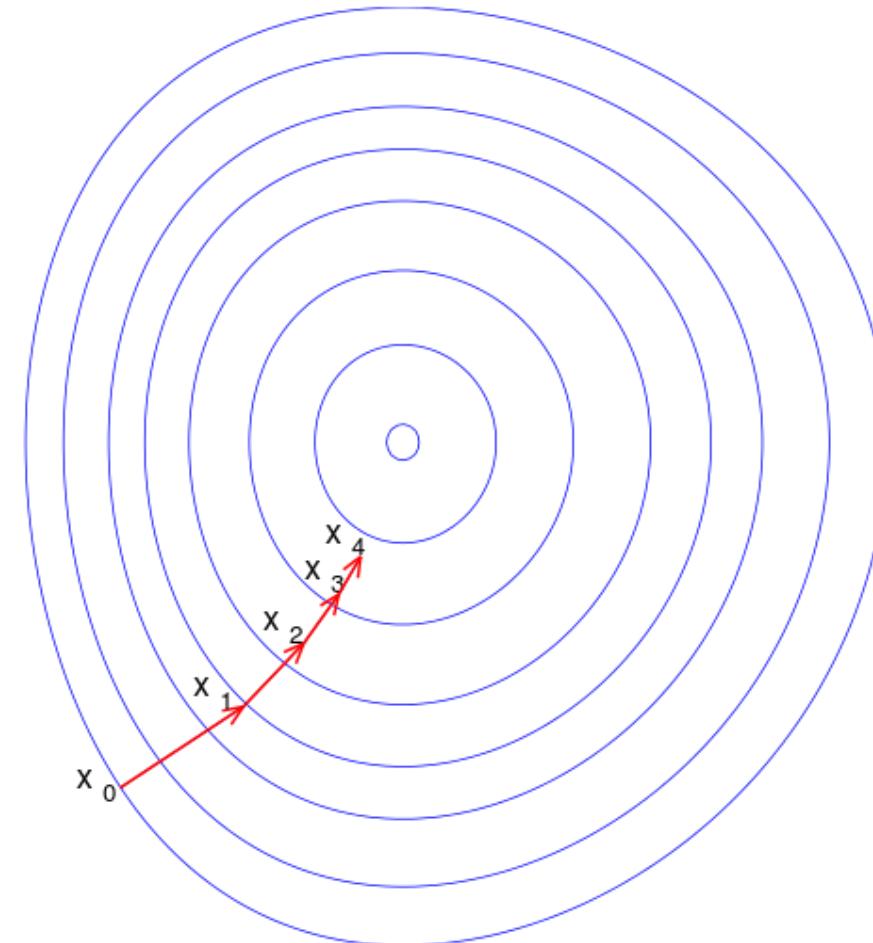
$$x^* = \operatorname{argmin}_x F(x)$$

- At position  $x_n$ ,  $F(x_{n+1}) \leq F(x_n)$  is guaranteed if,

$$x_{n+1} = x_n - \gamma \nabla F(x_n)$$

where  $\nabla F(x_n)$  is the gradient of  $F$  at  $x_n$

- $\gamma$  is the *step size*





### Objectives

$$w = \arg \min_w \sum_{i=1}^M \frac{1}{2} (w^T x_i - y_i)^2$$

### Consider a single instance

$$J(w) = \frac{1}{2} (w^T x - y)^2 = \frac{1}{2} z^2, \quad z = w^T x - y$$

### Gradient descent requires

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial w} = z x = (w^T x - y)x$$

where  $J(w) \in \mathbb{R}$ ,  $z \in \mathbb{R}$ ,  $w \in \mathbb{R}^n$



Constant Rule  $f(x) = c \rightarrow f'(x) = 0$

Constant Multiple Rule  $f(x) = c \cdot g(x) \rightarrow f'(x) = cg'(x)$

Power Rule  $f(x) = x^n \rightarrow f'(x) = nx^{n-1}$

Sum Rule  $f(x) = g(x) + h(x) \rightarrow f'(x) = g'(x) + h'(x)$

Product Rule  $f(x) = g(x)h(x) \rightarrow f'(x) = g'(x)h(x) + g(x)h'(x)$

Quotient Rule  $f(x) = \frac{g(x)}{h(x)} \rightarrow f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$

Chain Rule  $f(x) = g(u), u = h(x) \rightarrow f'(x) = g'(u)h'(x)$



- Given a multivariable function  $f(x, y)$ , and two univariate functions  $x(t), y(t)$ , the **multivariable chain rule** is:

$$\frac{df(\textcolor{blue}{x}(t), \textcolor{red}{y}(t))}{dt} = \frac{\partial f}{\partial \textcolor{blue}{x}} \frac{dx}{dt} + \frac{\partial f}{\partial \textcolor{red}{y}} \frac{dy}{dt}$$

- How to compute  $\frac{\partial f(x)}{\partial x}$  if  $f(x)$  and/or  $x$  is scalar, vector, matrix?



$$\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m, \mathbf{X} \in \mathbb{R}^{n \times m}, \mathbf{Y} \in \mathbb{R}^{m \times n}$$

what are  $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}, \frac{\partial \mathbf{y}}{\partial \mathbf{x}}, \frac{\partial \mathbf{y}}{\partial \mathbf{x}}, \frac{\partial \mathbf{y}}{\partial \mathbf{X}}, \frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$

Types	Scalar	Vector	Matrix
Scalar	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$
Vector	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	
Matrix	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$		

### Denominator layout

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{1 \times m}, \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbb{R}^n$$

### Numerator layout

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^m, \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbb{R}^{1 \times n}$$



$$\overset{\curvearrowright}{\mathbf{y}} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m, \mathbf{X} \in \mathbb{R}^{n \times m}, \mathbf{Y} \in \mathbb{R}^{m \times n}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_m}{\partial x} \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1m}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{n1}} & \frac{\partial y}{\partial x_{n2}} & \cdots & \frac{\partial y}{\partial x_{nm}} \end{bmatrix} \quad \frac{\partial \mathbf{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{21}}{\partial x} & \cdots & \frac{\partial y_{m1}}{\partial x} \\ \frac{\partial y_{12}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{m2}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{1n}}{\partial x} & \frac{\partial y_{2n}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}$$



$\mathbf{y} =$	$\mathbf{a}$	$\mathbf{x}$	$\mathbf{Ax}$	$\mathbf{x}^T \mathbf{A}$	
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} =$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{A}^T$	$\mathbf{A}$	
$\mathbf{y} =$	$a\mathbf{u}$ $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$v\mathbf{u}$ $v = v(\mathbf{x}), \mathbf{u} = \mathbf{u}(\mathbf{x})$	$\mathbf{v} + \mathbf{u}$ $\mathbf{v} = v(\mathbf{x}), \mathbf{u} = \mathbf{u}(\mathbf{x})$	$A\mathbf{u}$ $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$g(\mathbf{u})$ $\mathbf{u} = \mathbf{u}(\mathbf{x})$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} =$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$v \frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}} \mathbf{u}^T$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \mathbf{A}^T$	$\frac{\partial g(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$
$y =$	$a$	$\mathbf{u}^T \mathbf{v}$ $\mathbf{v} = \mathbf{v}(\mathbf{x}), \mathbf{u} = \mathbf{u}(\mathbf{x})$	$g(u)$ $u = u(\mathbf{x})$		
$\frac{\partial y}{\partial \mathbf{x}} =$	$\mathbf{0}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \mathbf{v} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}} \mathbf{u}$	$\frac{\partial g(u)}{\partial u} \frac{\partial u}{\partial \mathbf{x}}$		



### Objectives

$$w = \arg \min_w \sum_{i=1}^M \frac{1}{2} (w^T x_i - y_i)^2$$

### Consider a single instance

$$J(w) = \frac{1}{2} (w^T x - y)^2 = \frac{1}{2} z^2, \quad z = w^T x - y$$

### Gradient descent requires

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial w} = \cancel{z} \cancel{x} = (w^T x - y)x$$

where  $J(w) \in \mathbb{R}$ ,  $z \in \mathbb{R}$ ,  $w \in \mathbb{R}^n$

### Update $w$

$$w_{n+1} = w_n - \lambda \frac{\partial J(w)}{\partial w} \Big|_{w=w_n} = w_n - \lambda (w_n^T x - y)x$$



• What we solve just now, is

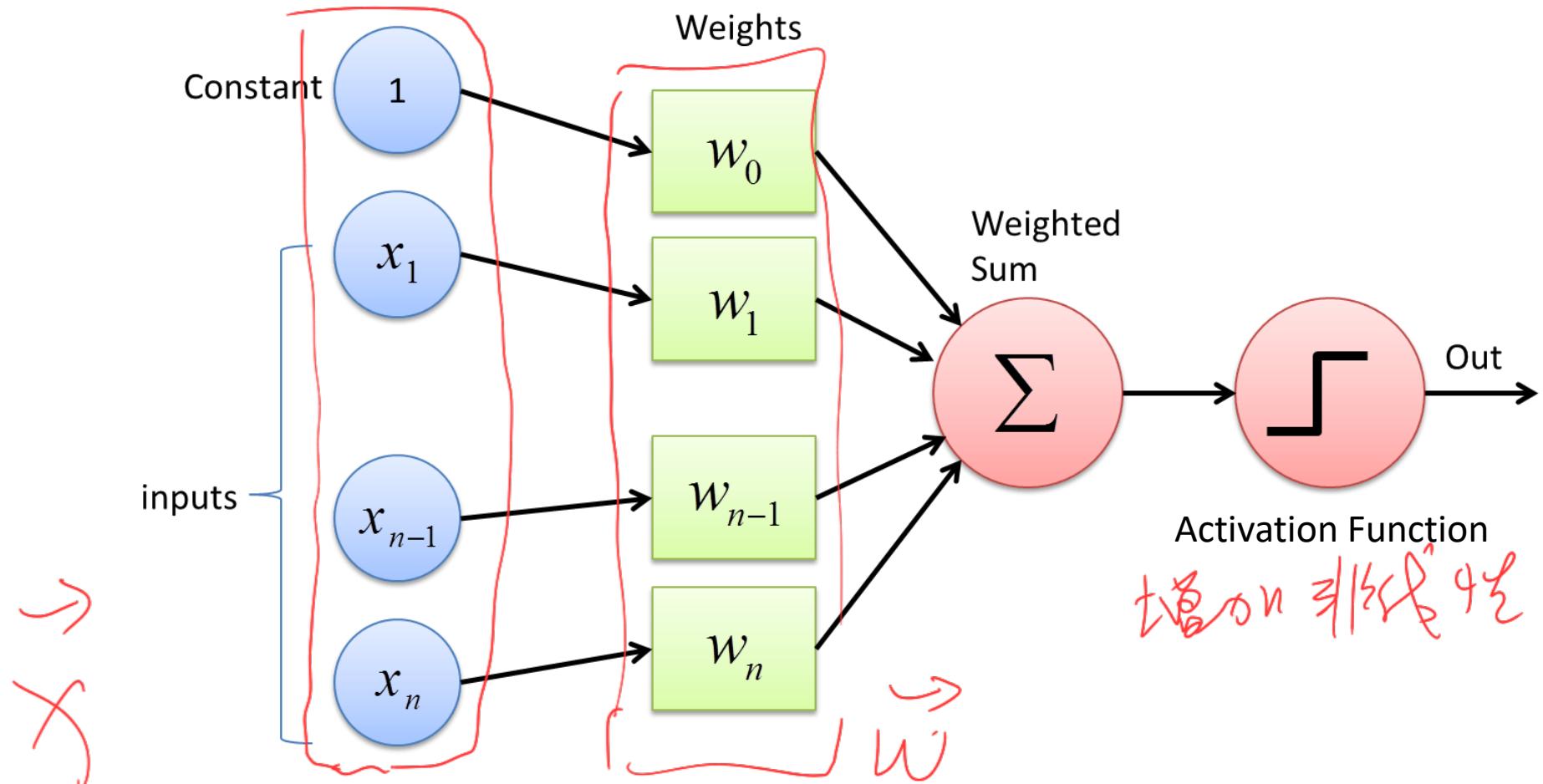
$$w = \arg \min_w \sum_{i=1}^M \frac{1}{2} (w^T x_i - y_i)^2$$

• The is the very basic Neural Network – Perceptron

- $w$  is the trainable parameter
- $x_i$  is the data point,  $y_i$  is the label
- Invented in 1968
- Frank Rosenblatt, Cornell Aeronautical Laboratory



## Perceptron



The most simple Neural Network (NN): A Perceptron:  $y = w^T x$



## Train a Perceptron

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

and a perceptron

$$y = w^T x$$

Modify weight  $w$  such that  $\hat{y}$  gets ‘closer’ to  $y$

↑  
perceptron  
parameter

↑  
perceptron  
output

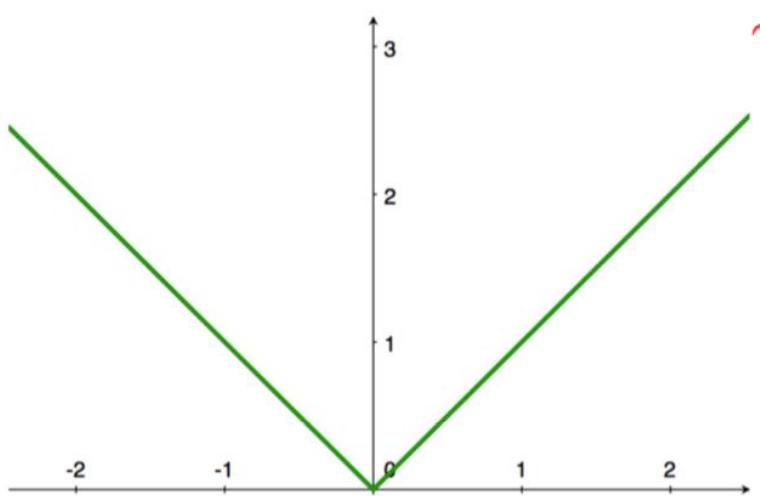
what does  
*this mean?*

↑  
true  
label



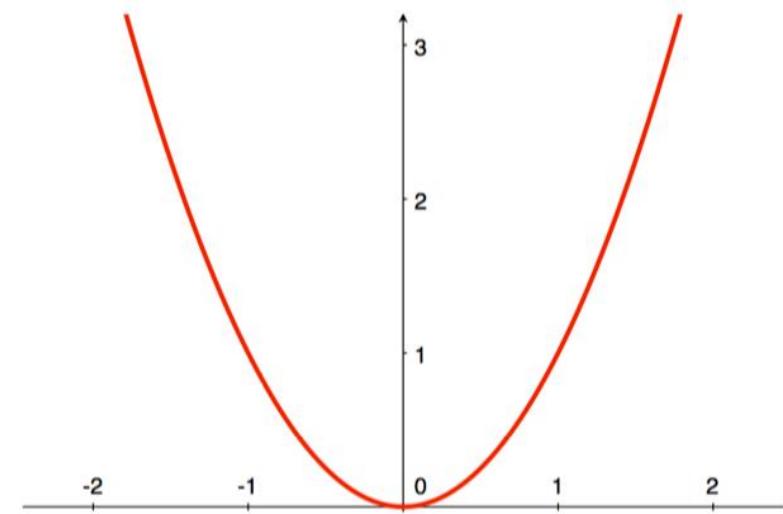
## L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



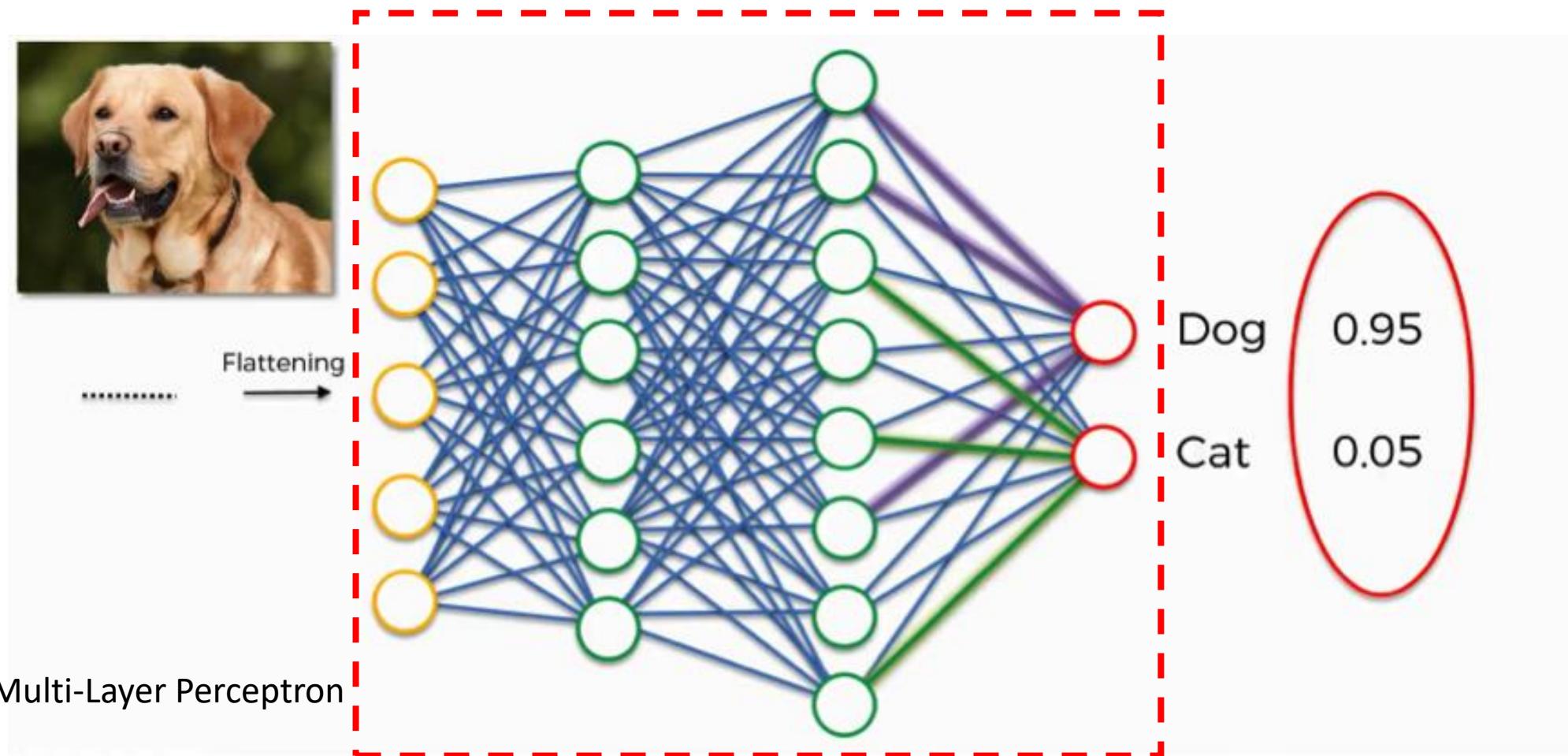
## L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$





## Loss – Classification – Cross Entropy Loss





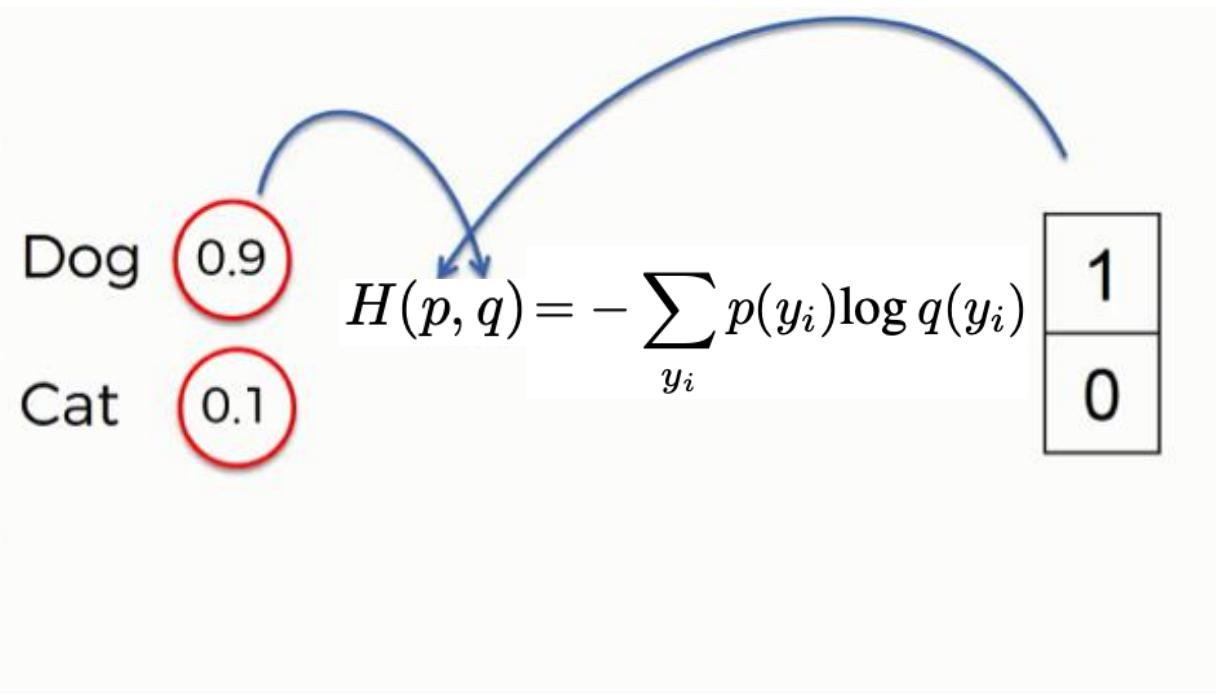
## Loss – Classification – Cross Entropy Loss

$-p(y_i=1) \log q(y_i)$  因为  $p(y_i)$  只有  $y_i=1$  时才为 1

### Cross Entropy Loss

$$H(p, q) = - \sum_{y_i} p(y_i) \log q(y_i), \quad \sum_{y_i} p(y_i) = 1, \sum_{y_i} q(y_i) = 1$$

- $p(y_i = 1)$  is the **ground-truth** probability of category  $i$
- $q(y_i = 1)$  is the **predicted** probability of category  $i$





Cross Entropy Loss → Negative Log Softmax

把 score 轉成 probability

$$H(p, q) = - \sum_{y_i} p(y_i) \log q(y_i) \leftrightarrow L = H(p, q) = -\log \frac{e^{s_{i^*}}}{\sum_j e^{s_j}}, \text{ where } p(y_{i^*}) = 1$$



dog

3.2

Unnormalized probabilities

24.5

cat

5.1

exp

164.0

normalize

0.13

$$L = -\log \frac{e^{3.2}}{e^{3.2} + e^{5.1} + e^{-1.7}}$$

frog

-1.7

0.18

0.87

$$= -\log \frac{24.5}{24.5 + 164.0 + 0.18}$$

Network output **score**:

Unnormalized log probabilities

Soft Max

0.00

$$\begin{aligned} &= -\log 0.13 \\ &= 0.89 \end{aligned}$$

Probabilities



## Multi-Layer Perceptron (MLP)

- How many Perceptrons?

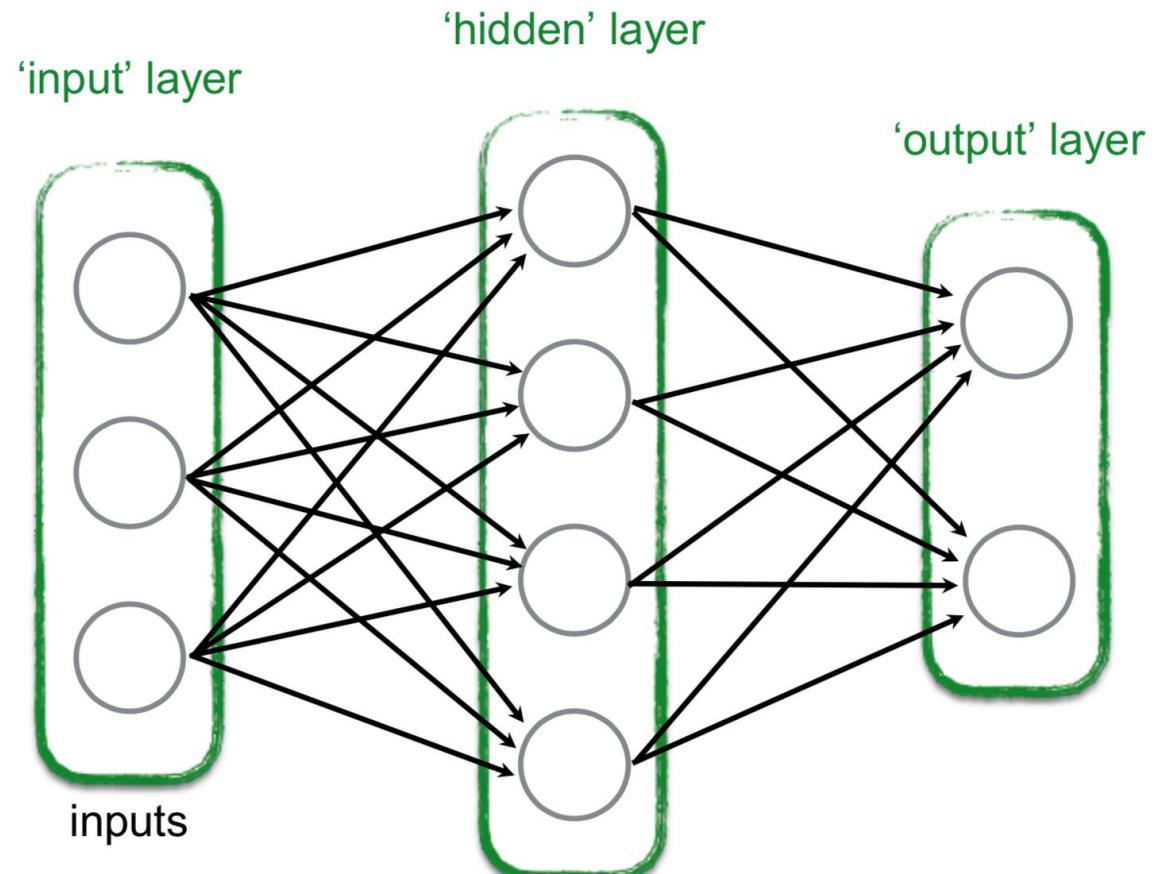
$$4 + 2 = 6$$

- How many trainable parameters?

$$3 \times 4 + 4 \times 2 = 20$$

- Output  $y = w_2^T (w_1^T x) = w'^T x$

- It is **linear** no matter how many layers are in the MLP!





$$XOR(0,0) = XOR(1,1) = 0$$

$$XOR(0,1) = XOR(1,0) = 1$$

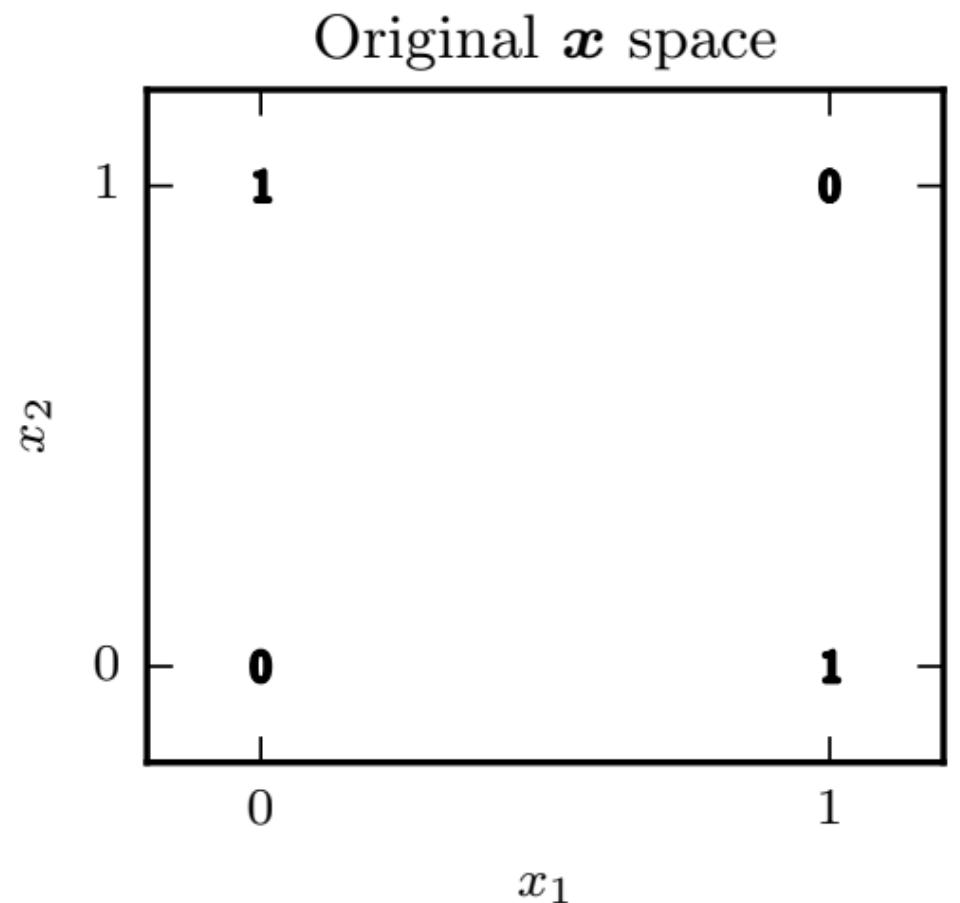
Train a linear MLP  $y = w^T[x_1, x_2]$  so that:

$$w^T[0,0] = w^T[1,1] < 0$$

$$w^T[0,1] = w^T[1,0] > 0$$

NOT Possible, because  $w^T x = 0$  is a hyperplane

How? Make it non-linear

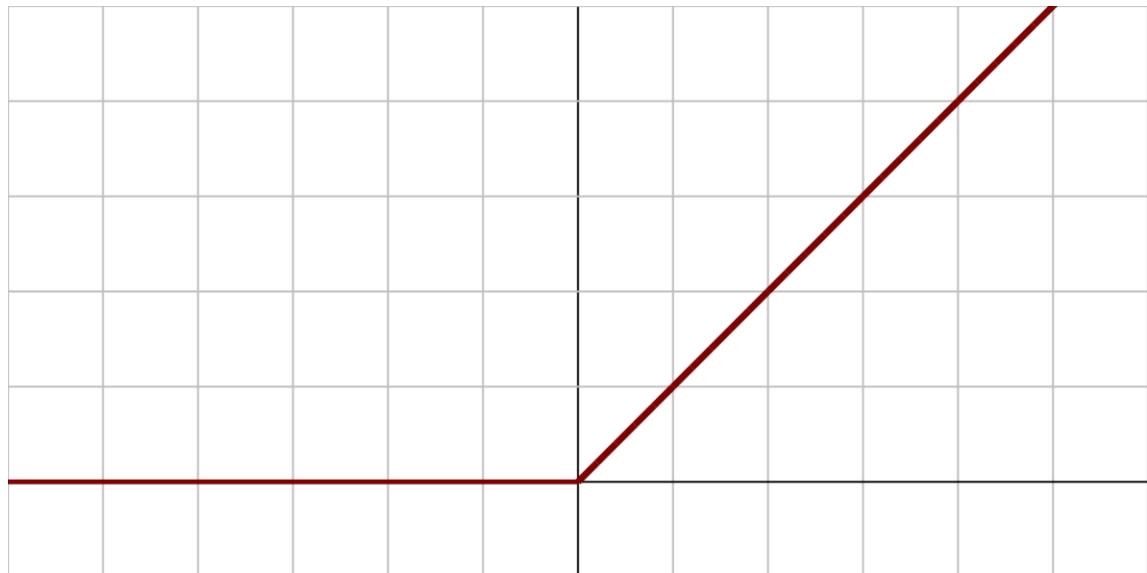




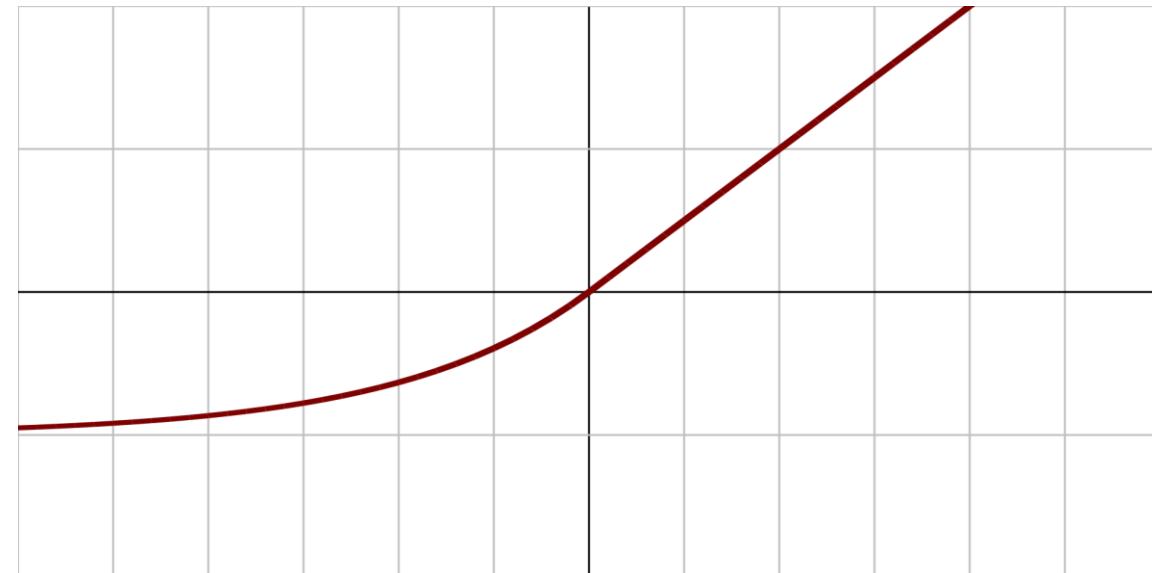
## MLP – Activation Function

- Add non-linearity by activation function  $g(\cdot)$
- A perception becomes  $y = g(f(x)) = g(w^T x)$

Rectified Linear Unit (ReLU)  $f(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{cases}$



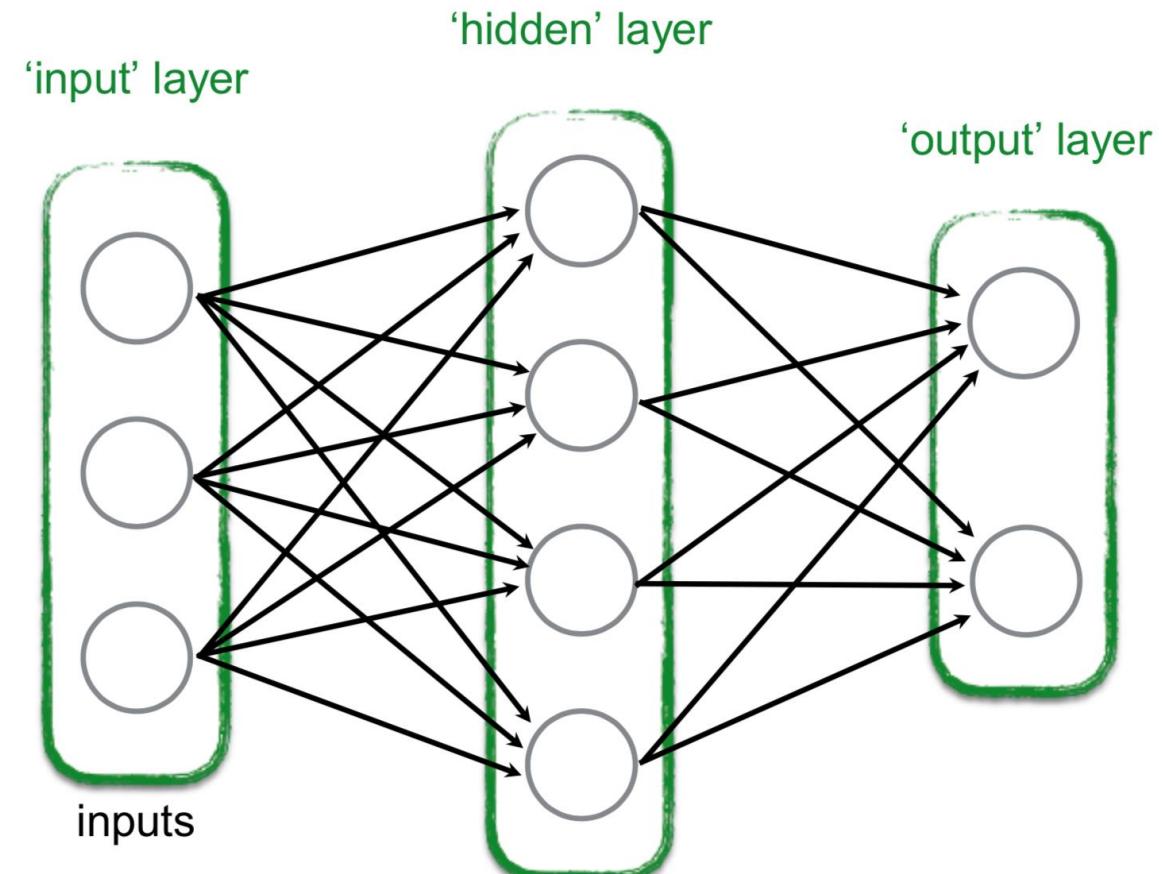
Exponential Linear Unit (ELU)  $f(x) = \begin{cases} \alpha(e^x) - 1, & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{cases}$





## Multi-Layer Perceptron (MLP) Fully Connected Network (FC)

- A MLP with activation &  $\geq 1$  hidden layers
- Is able to simulate **ANY** function  $f(x)$ , where  $x$  is the input
- There is proof for this, not shown in this lecture.



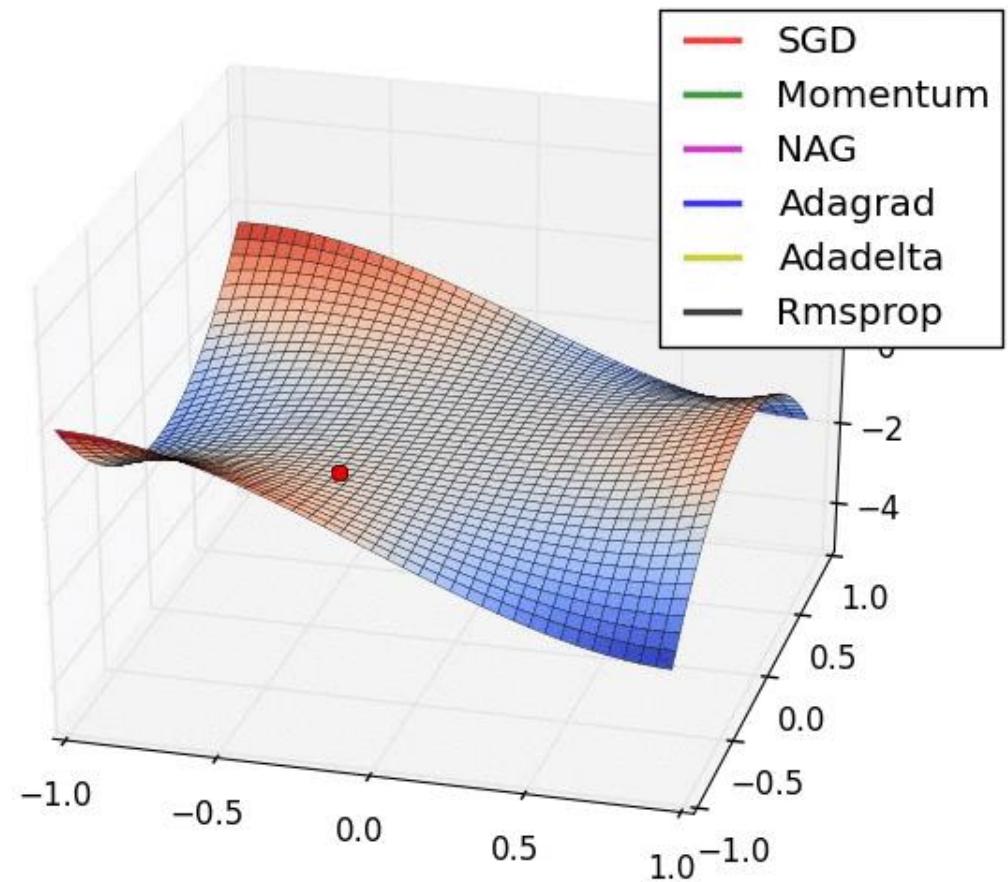


- Same, gradient descent.

$$w_{n+1} = w_n - \lambda \frac{\partial L}{\partial w} \Big|_{w=w_n}, \forall w$$

- What is **Back Propagation**?
- An efficient way of gradient descent.
- What are optimizers like SGD (Stochastic Gradient Descent), Adam?
- Improvements over gradient descent, still gradient descent.

### Gradient Descent Example





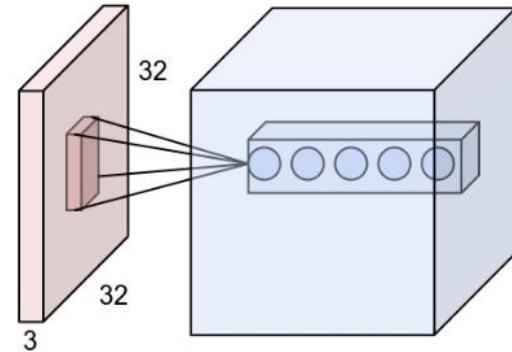
## What is CNN?

- Apply convolution on 1D/2D/3D

## Why CNN?

- Features can be extracted in a **local neighborhood**.

(We will give detailed answers later, after showing what is CNN)



Input image



Convolution  
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

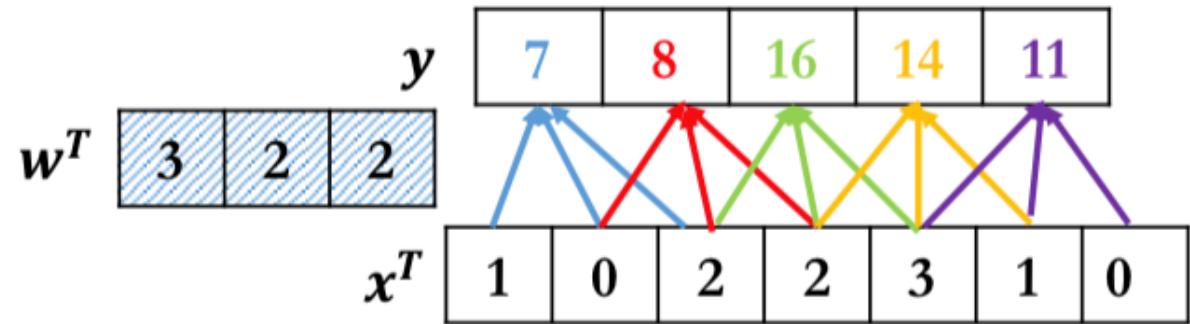
Feature map





## 1D Convolution

$$y_t = \sum_{i=0}^{k-1} w_i \cdot x_{t+i}$$



•  $w$  is the kernel / filter

- Length  $k$
- They are the unknown/trainable parameters

•  $x$  is the input

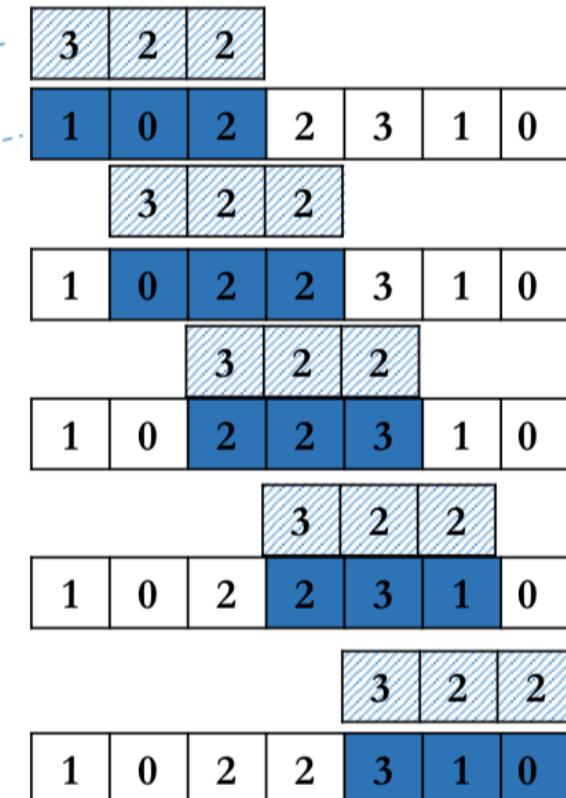
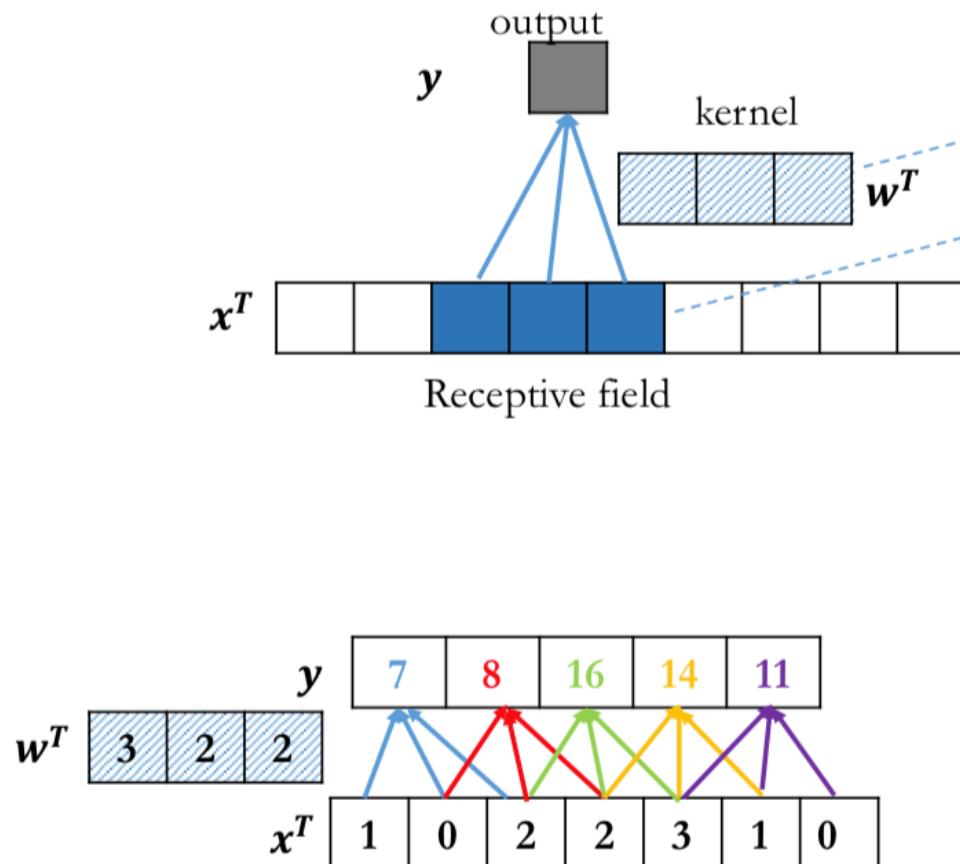
- Length  $n$
- Receptive field:  $[x_t, x_{t+1}, \dots, x_{t+k-1}]$
- Each receptive field generates one output value  $y_t$

•  $y$  is the output

- Length  $o$



# 1D Convolution



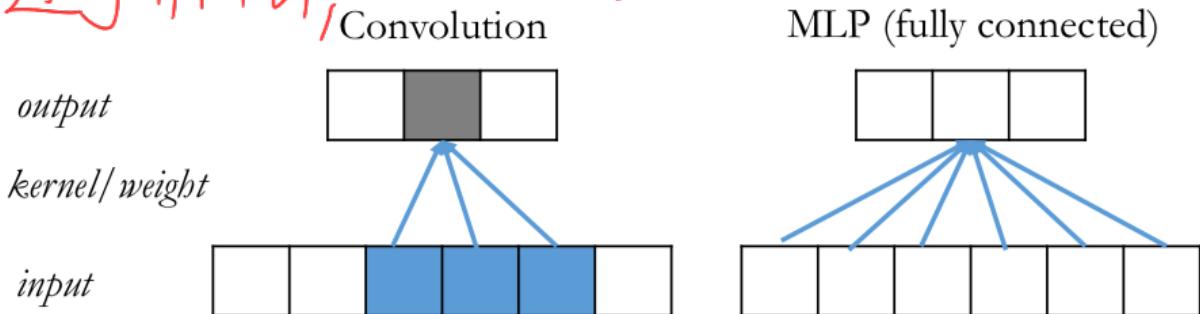


## Why CNN ?

CNN是稀疏的，MLP过于稠密，容易过拟合

### 1. Sparse connection vs. Dense

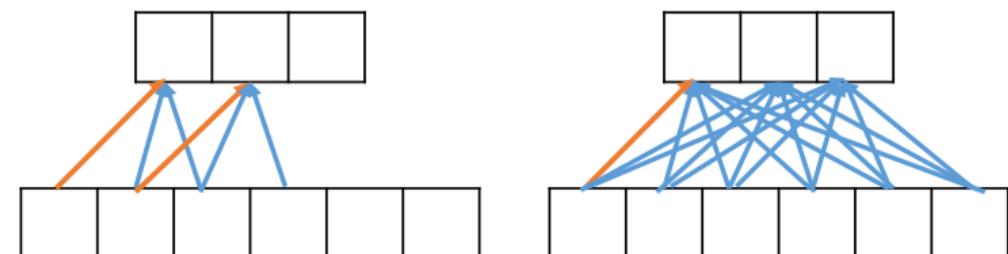
- Fewer parameters. 3 vs. 18
- Less overfitting



### 2. Weight sharing vs. Unique weights

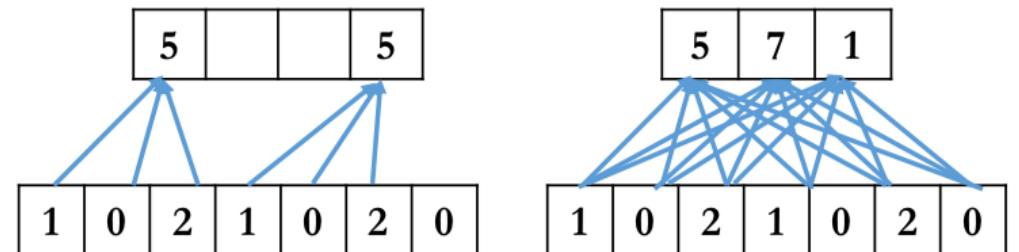
- Less overfitting

CNN 权重可共享



### 3. Local invariant vs. Local variant

- Features should not depend on the location within the image
- Make the same prediction no matter where is the object in the image





- ## How to determine edge values?

# Padding!

-  Given padding  $p$ , what is the output length  $o$ ?

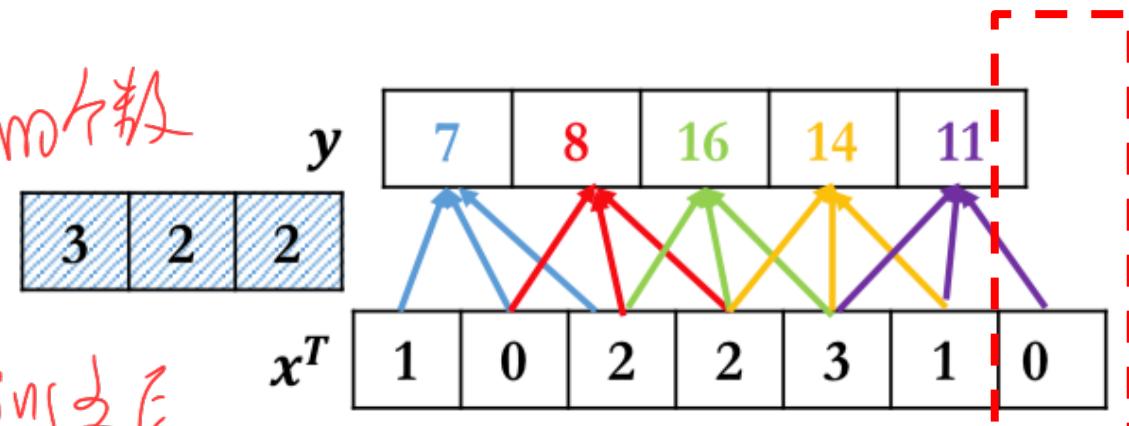
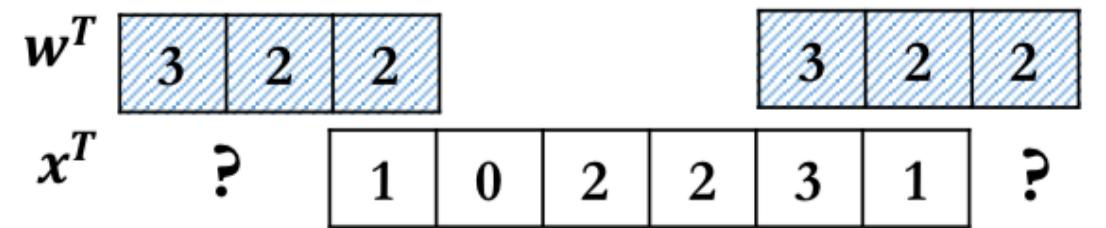
- kernel size  $k$
  - Input length  $n$

## Without padding:

$$o = n - k + 1$$

## With padding:

$$o = (n + p) - k + 1$$



## 拉锯战的个数

→ 有 padding 之後

no 風格繪出個樣



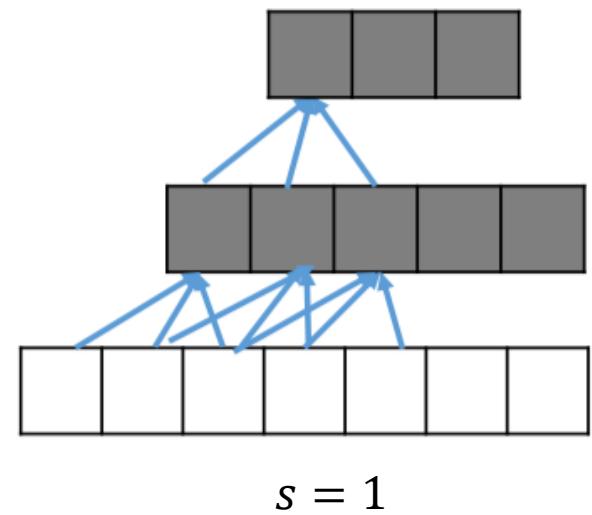
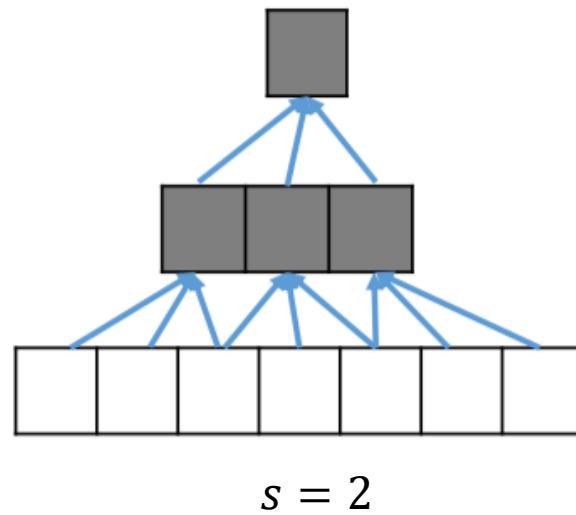
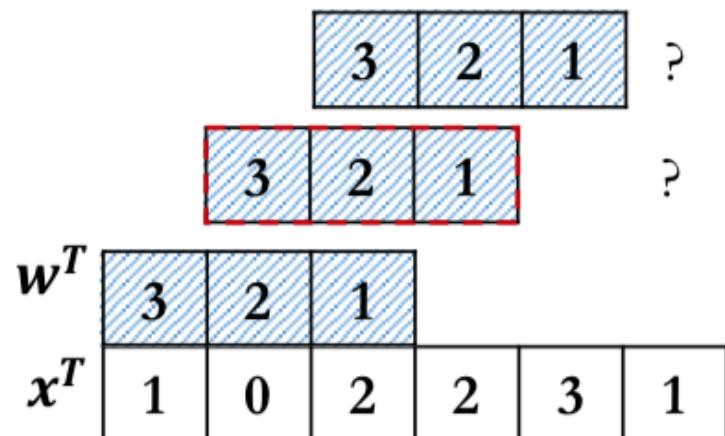
## 1D Convolution - Stride

❖ How many steps to move for the next receptive field?

- Stride! Skip some elements when  $s > 1$

❖ Why?

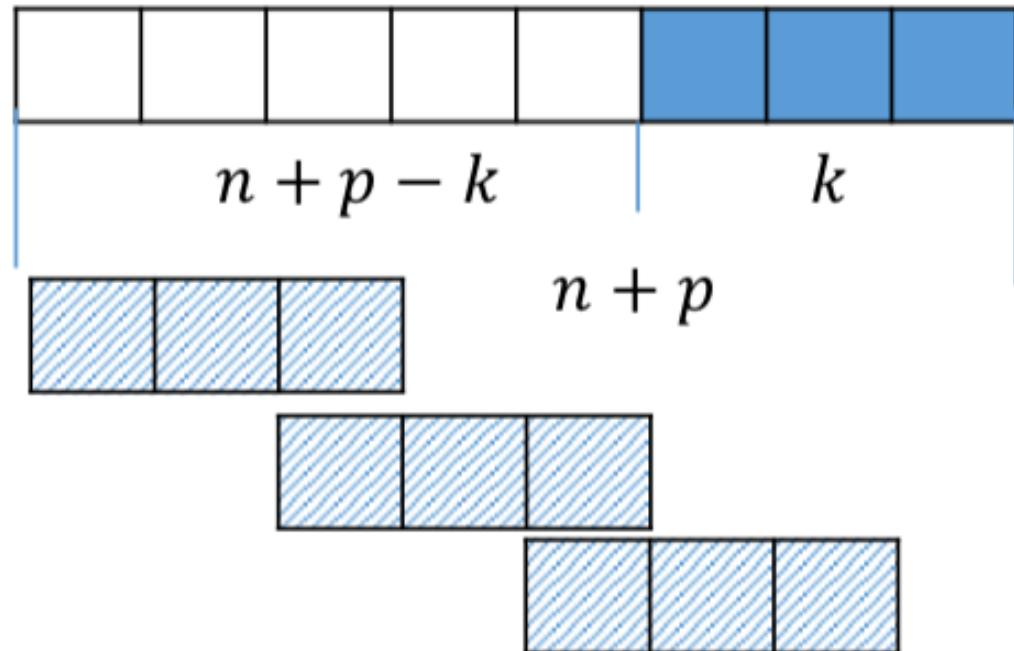
- Less compute
- Increase receptive field





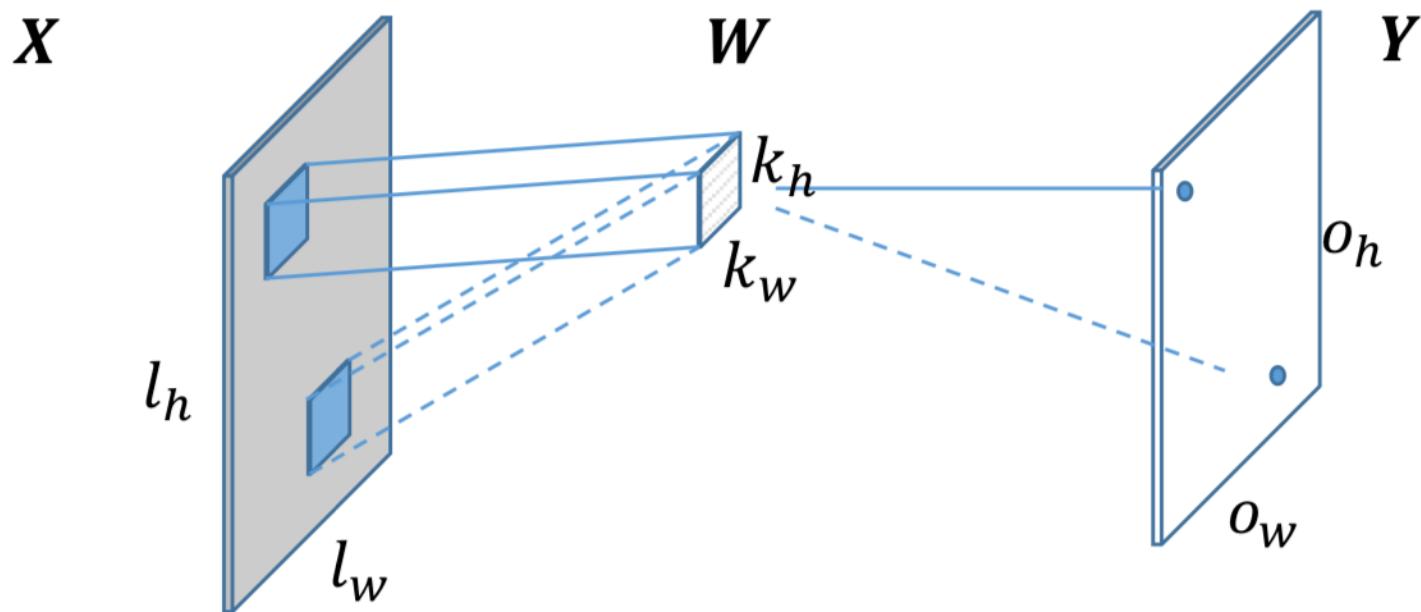
What is the output length  $o$ ?

- kernel size  $k$
  - Input length  $n$
  - Padding  $p$
  - Stride  $s$
- 
- $$o = \left\lfloor \frac{n+p-k}{s} \right\rfloor + 1$$





## Same but everything 2D

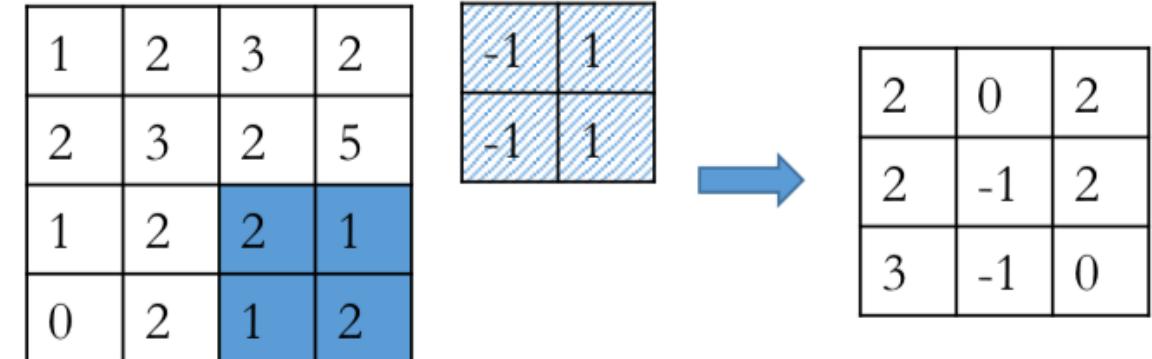
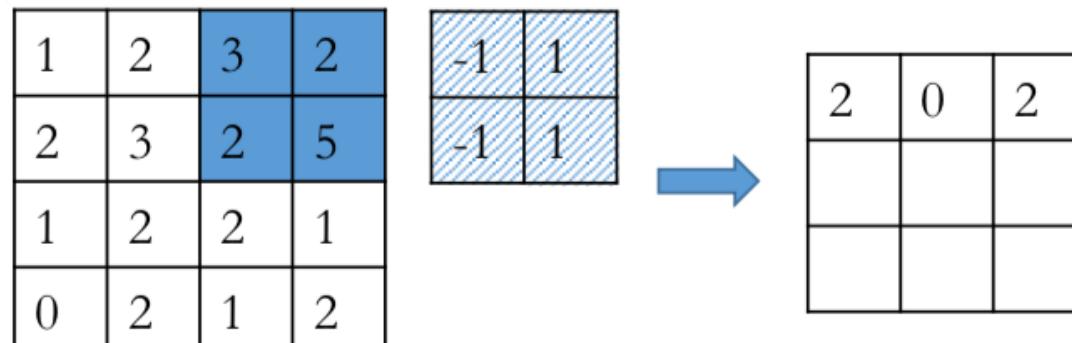
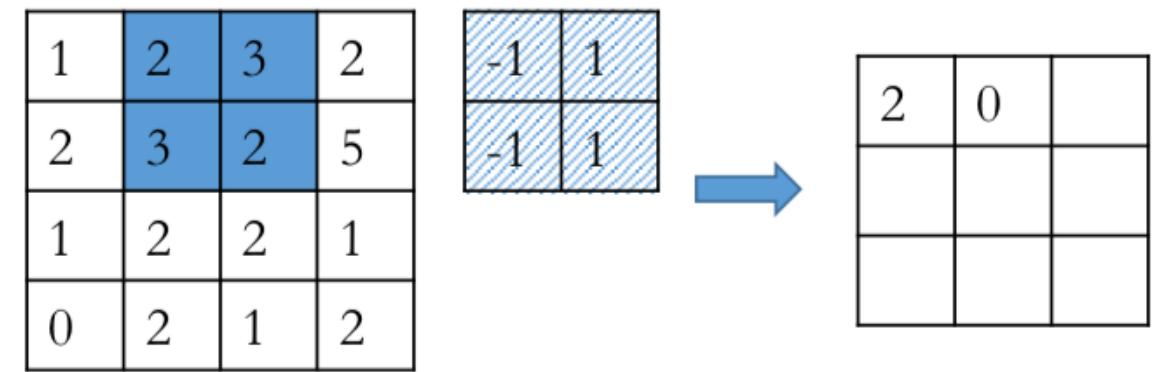
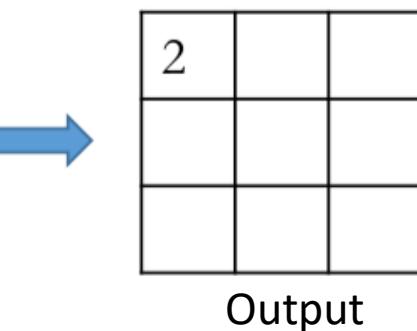
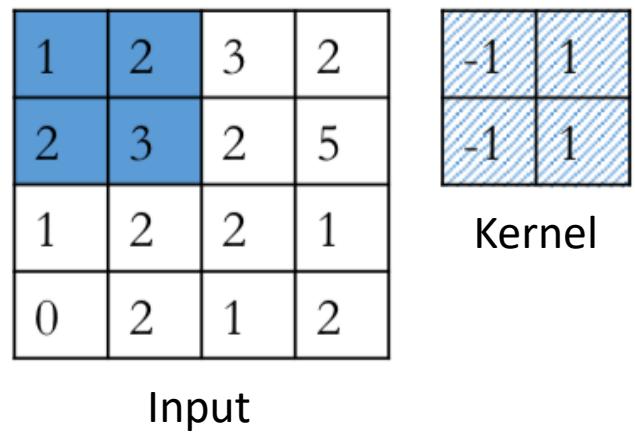


$$Y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} X_{i*s_h+a, j*s_w+b} \times W_{a,b}$$



## 2D Convolution

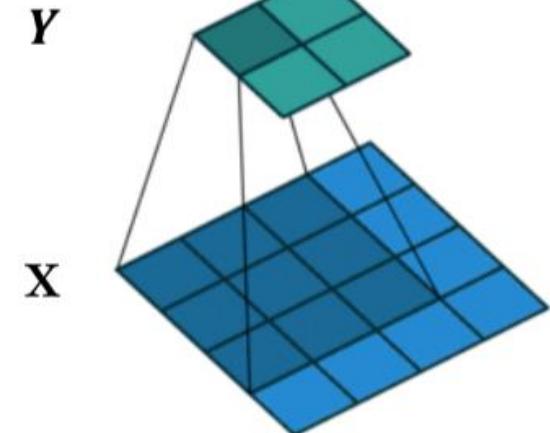
$$Y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} X_{i*s_h+a, j*s_w+b} \times W_{a,b}$$





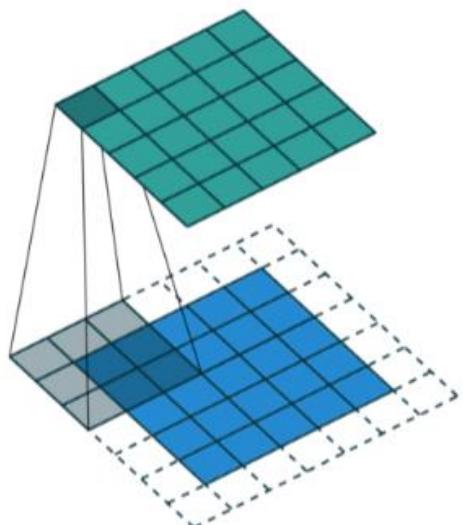
## 2D Convolution

kernel size  $k$   
Padding  $p$   
Stride  $s$



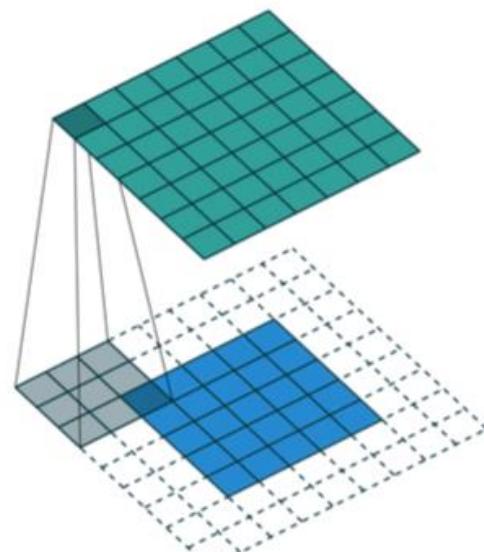
$k=3, p=0, s=1$  (Valid)

Valid: No padding

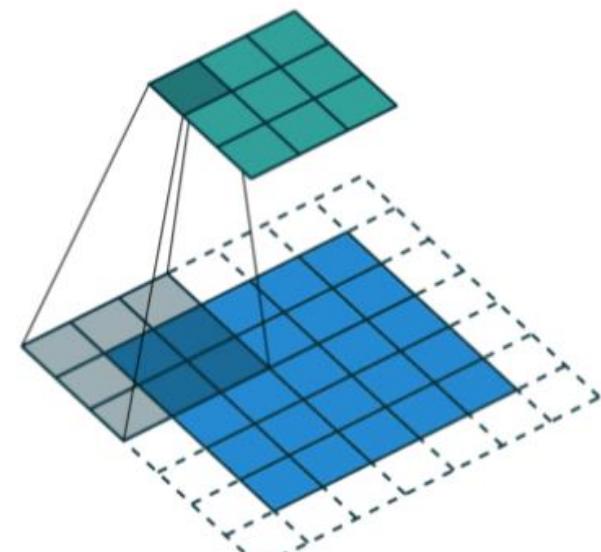


$k=3, p=2, s=1$  (Same)

Same: Add padding so  
input size = output size



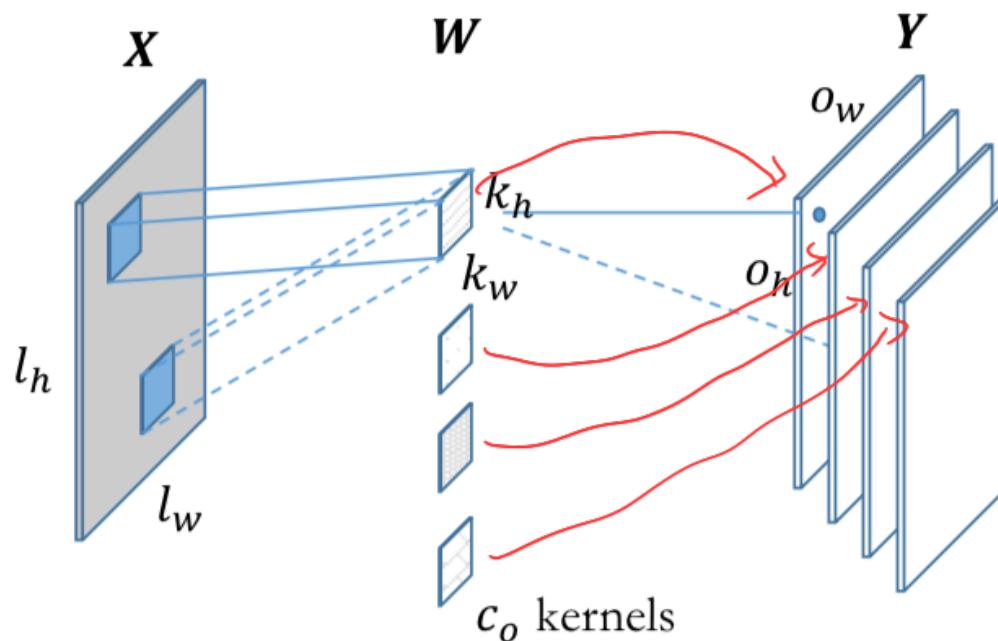
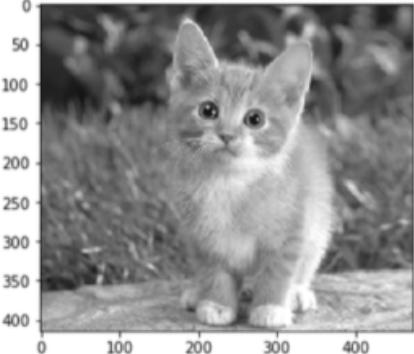
$k=3, p=4, s=1$  (Full)



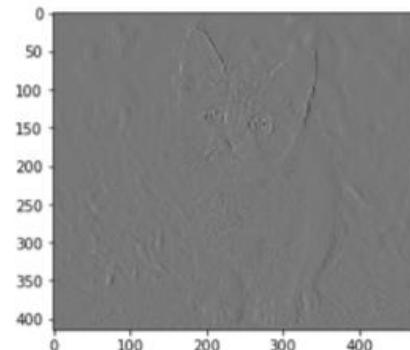
$k=3, p=2, s=2$



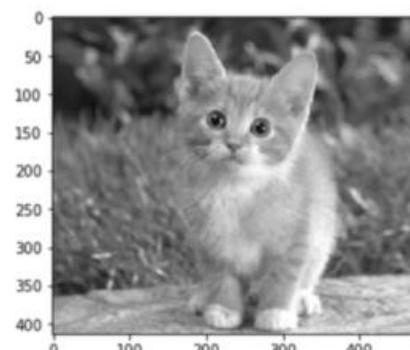
### ◆ Multiple features → Multiple kernels



$$\begin{matrix} -1 & 1 \\ -1 & 1 \end{matrix}$$



$$\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$



不同 kernel 产生不同的特征, 然后叠加

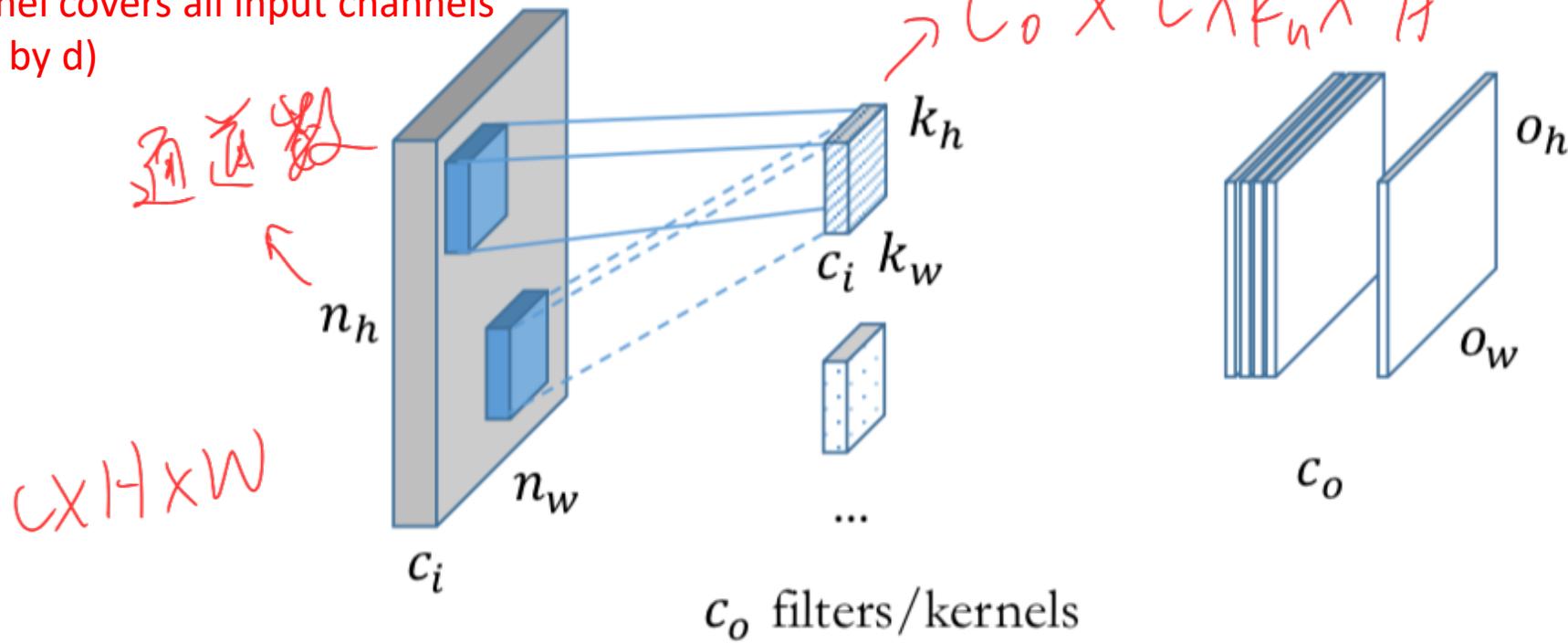


## 2D Convolution – Multiple Inputs & Kernels

$$Y_{l,i,j} = \sum_{d=0}^{c_i-1} \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} X_{d,i+a,j+b} \times W_{l,d,a,b} + b_l, l \in [0, c_o)$$

2D convolution across the plane (indexed by a, b)

Each kernel covers all input channels  
(indexed by d)





## 2D Convolution - Pooling

池化层

- Aggregate information in each receptive field

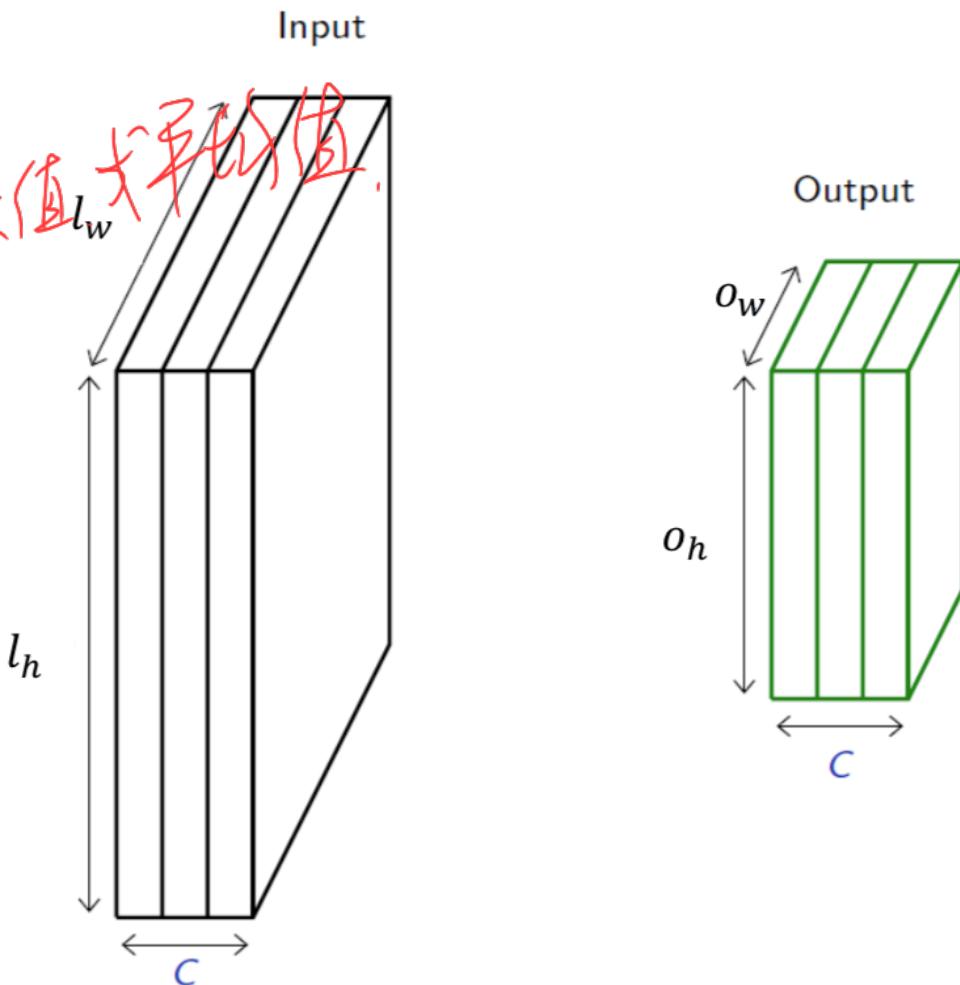
- Max
- Average

- No trainable parameter

- # input channels = # output channels

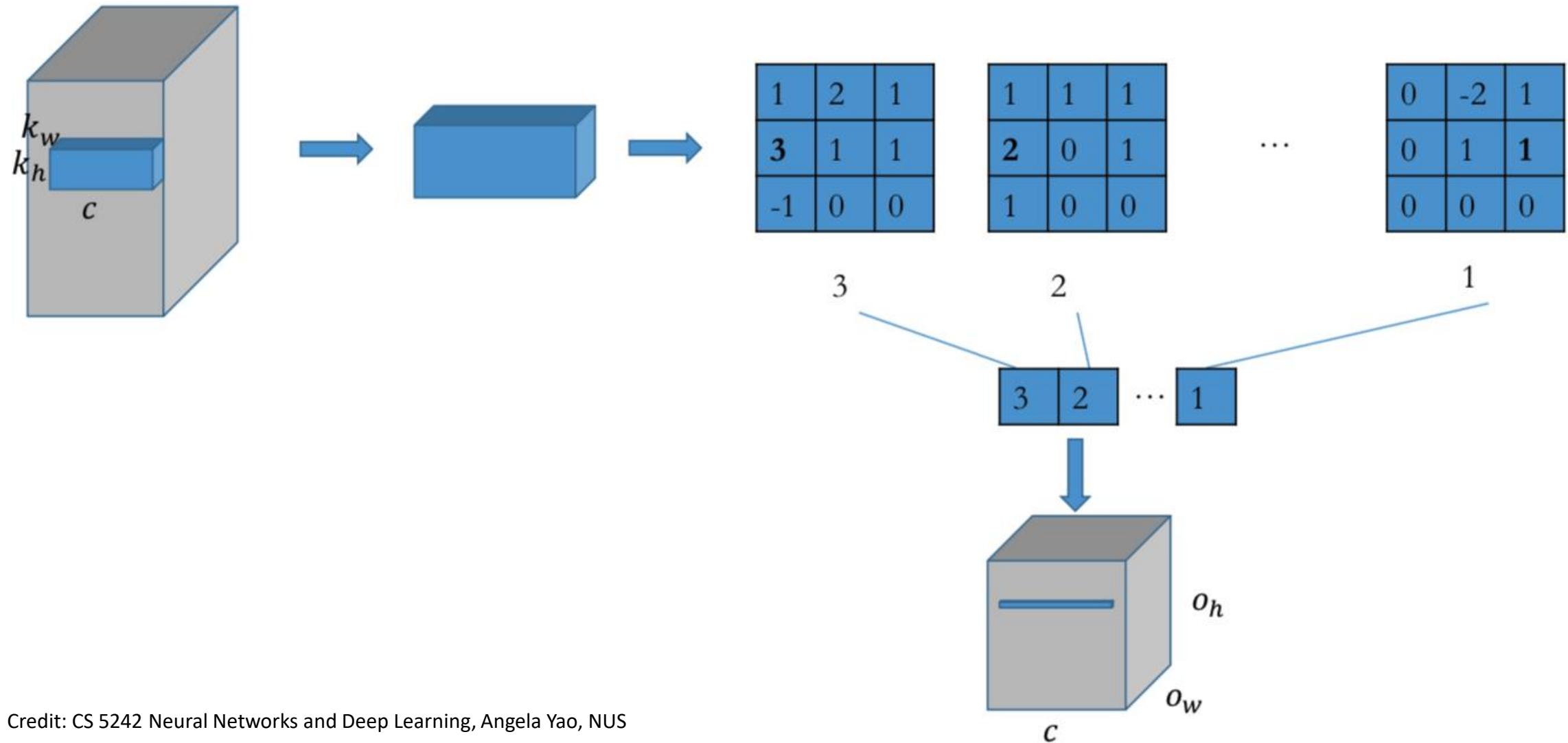
- $c_i = c_o$

- Same padding/stride method





## 2D Convolution – Max Pooling





- There are  $c_o$  kernels, they share same stride and padding, NOT parameter
- Parameter size  $c_o \times k_h \times k_w$
- Output shape

- $(c_o, o_h, o_w) = (c_o, \left\lfloor \frac{n_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{n_w + p_w - k_w}{s_w} \right\rfloor + 1)$

- Computation cost

- $O((c_o \times k_h \times k_w) \times (o_h \times o_w))$

### 1D Convolution Multiple inputs/kernels

- Same as 2D
- Set  $k_h$  or  $k_w$  to be 1.



- ◆ Natural extension of 2D Convolution

- ◆ Blue: input

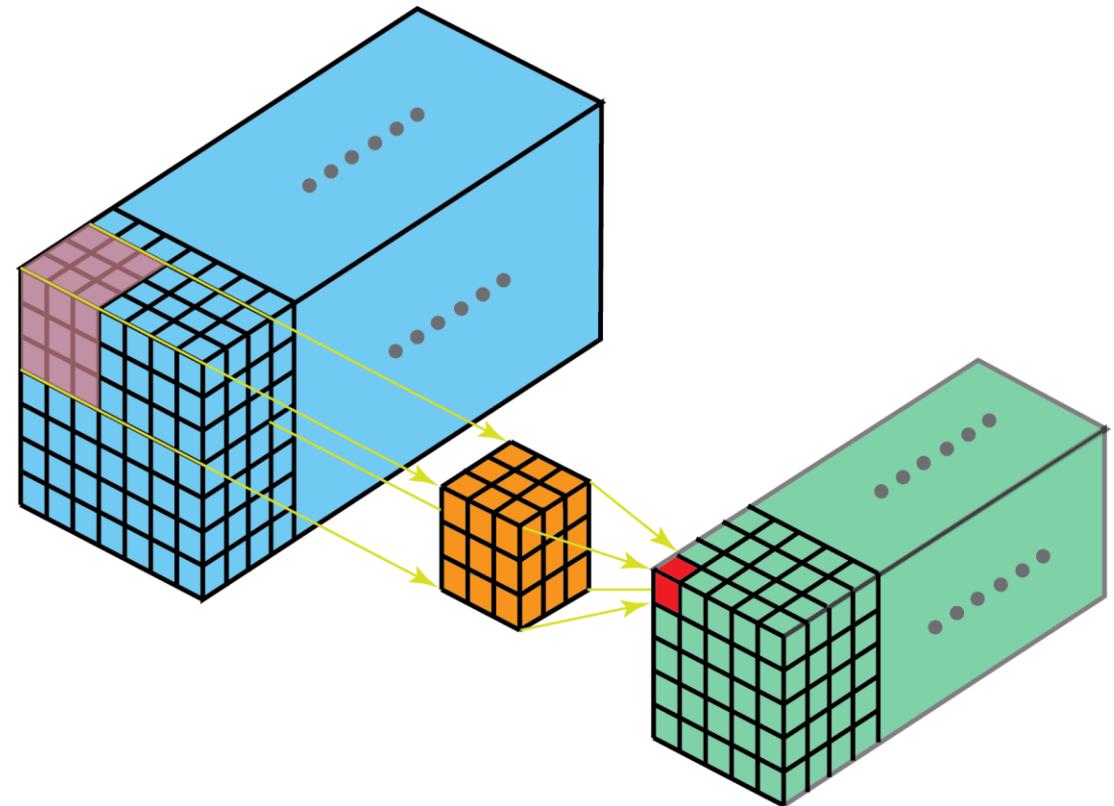
- Each small cube contains  $d$  features/channels

- ◆ Orange: kernel

- Each small cube contains  $d$  weights

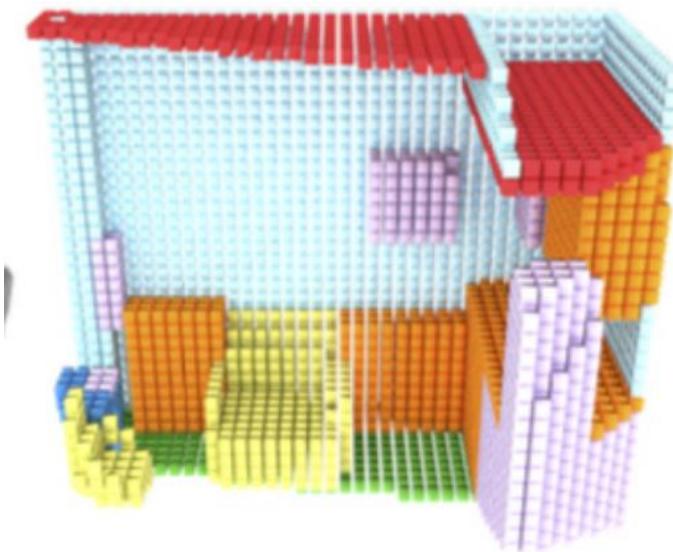
- ◆ Green: output

- Each small cube is a scalar.
  - There are  $o$  green blocks





- 3D convolution *直接3D卷积*
- Multi-view projection onto images + 2D convolution *将3D点投影到多张图片上*
- Simply run 1D/2D convolution or even MLP on point cloud



3D shape model  
rendered with  
different virtual cameras



2D rendered  
images

x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4



### Content of each grid cell

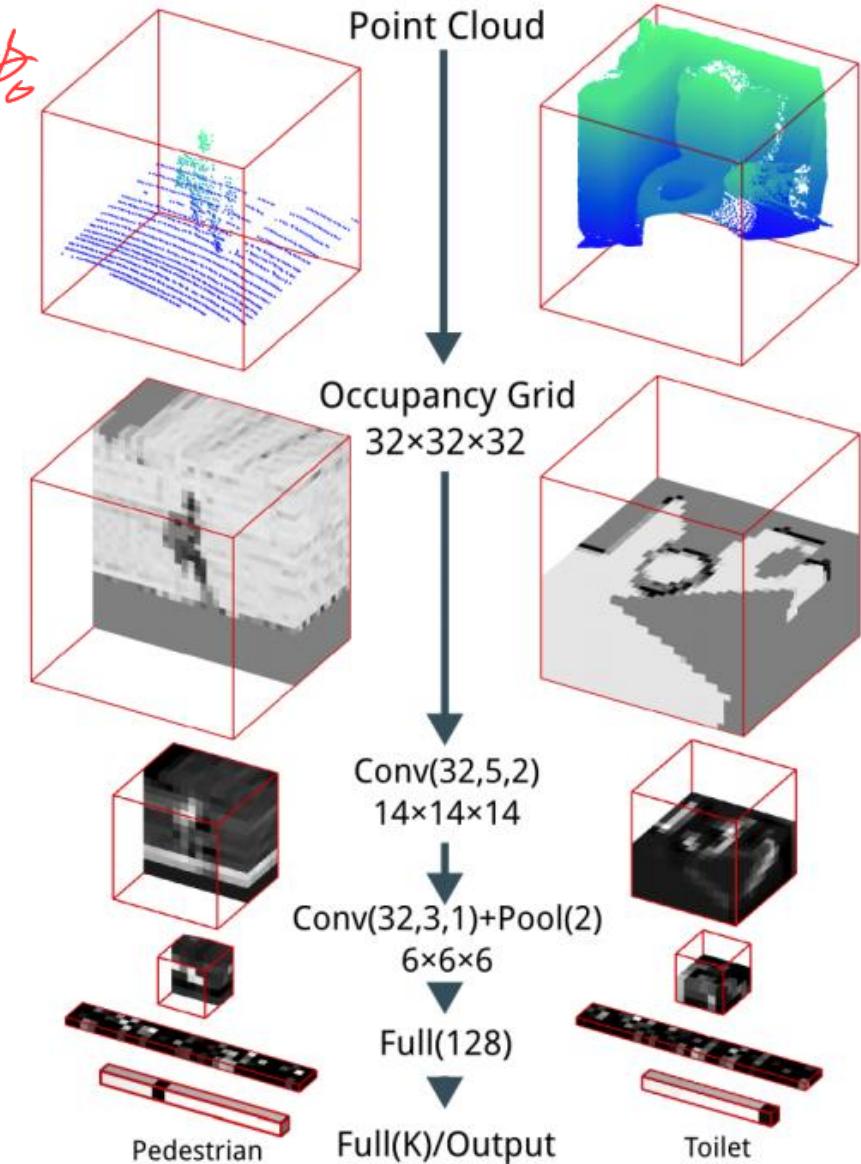
- Binary
- Number of points
- Probability
- etc.

点云、纹理、物体类特征  
格子里面可以是

### Accuracy on ModelNet40: 83%

### Conv( $o$ , $k$ , $s$ )

- $o$ : number of kernels
- $k$ : size of kernel (same for x/y/z)
- $s$ : stride (same for x/y/z)

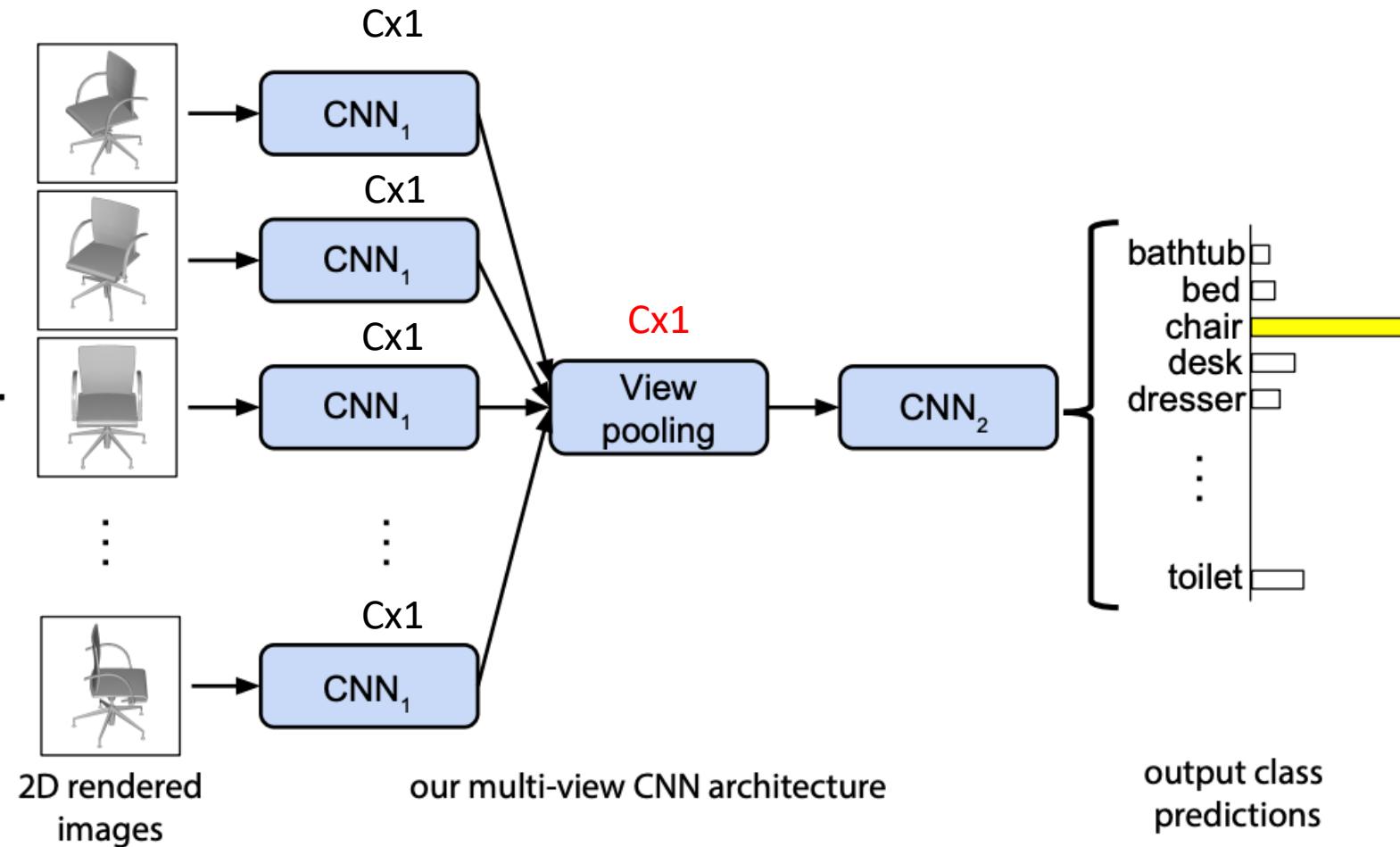




ModelNet40 Accuracy 90.1%

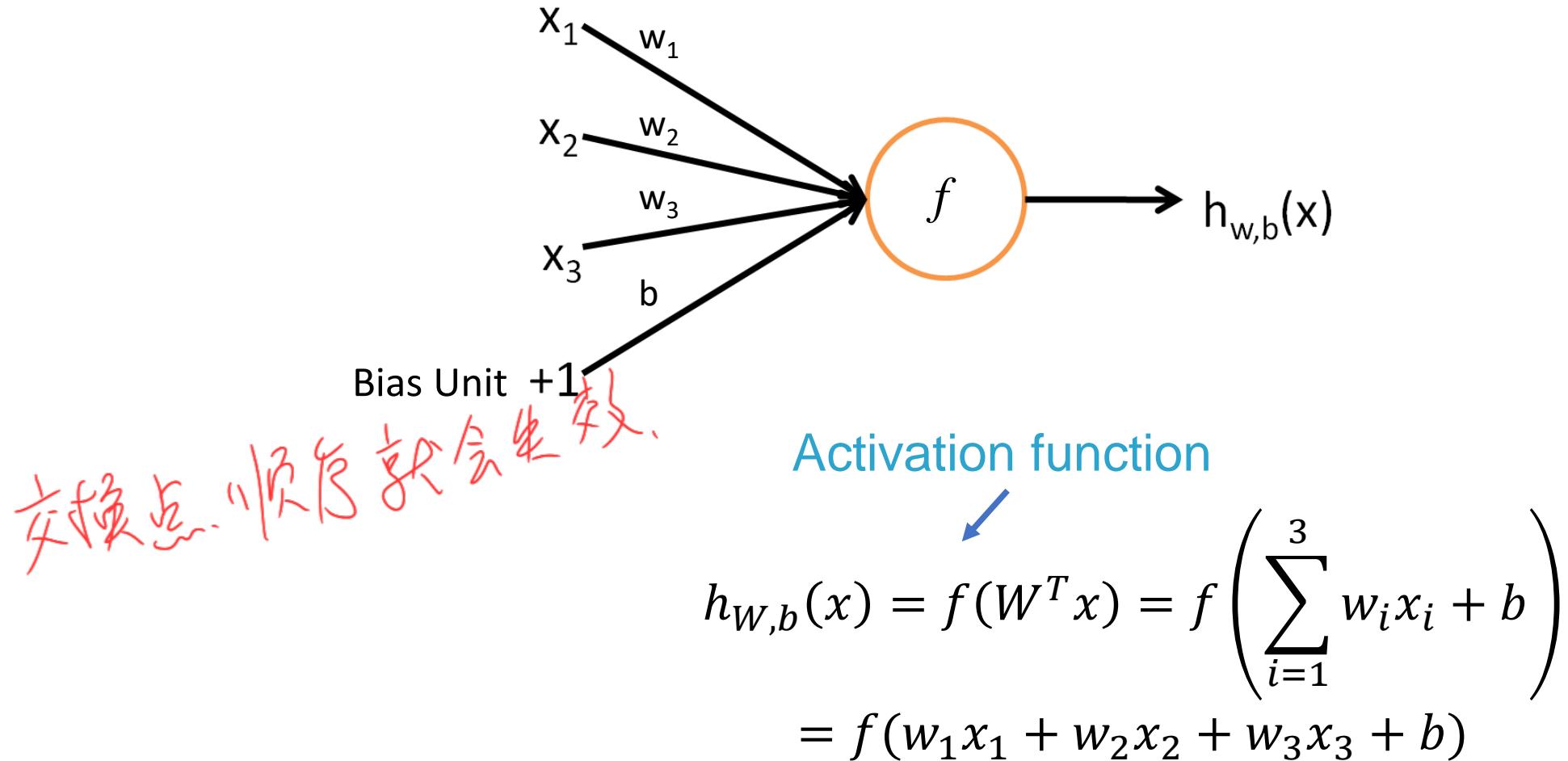


3D shape model  
rendered with  
different virtual cameras



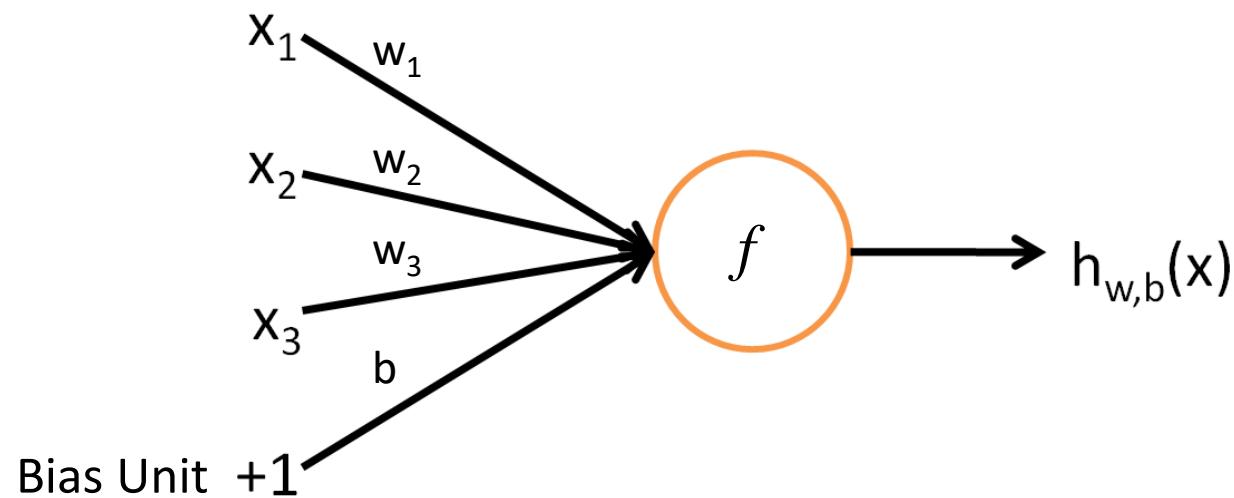


## MLP for Point Cloud?





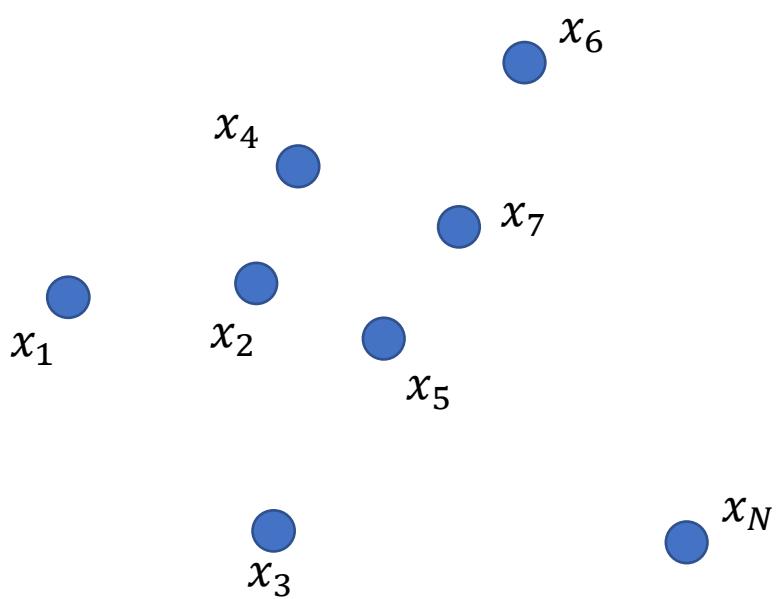
Not Permutation Invariant!



$$f(w_1x_3 + w_2x_1 + w_3x_2 + b) \neq h_{W,b}(x)$$



## Enumerate All Permutation?



$$X_1 = [x_1^T, x_2^T, \dots, x_N^T]^T$$

$$X_2 = [x_2^T, x_1^T, \dots, x_N^T]^T$$

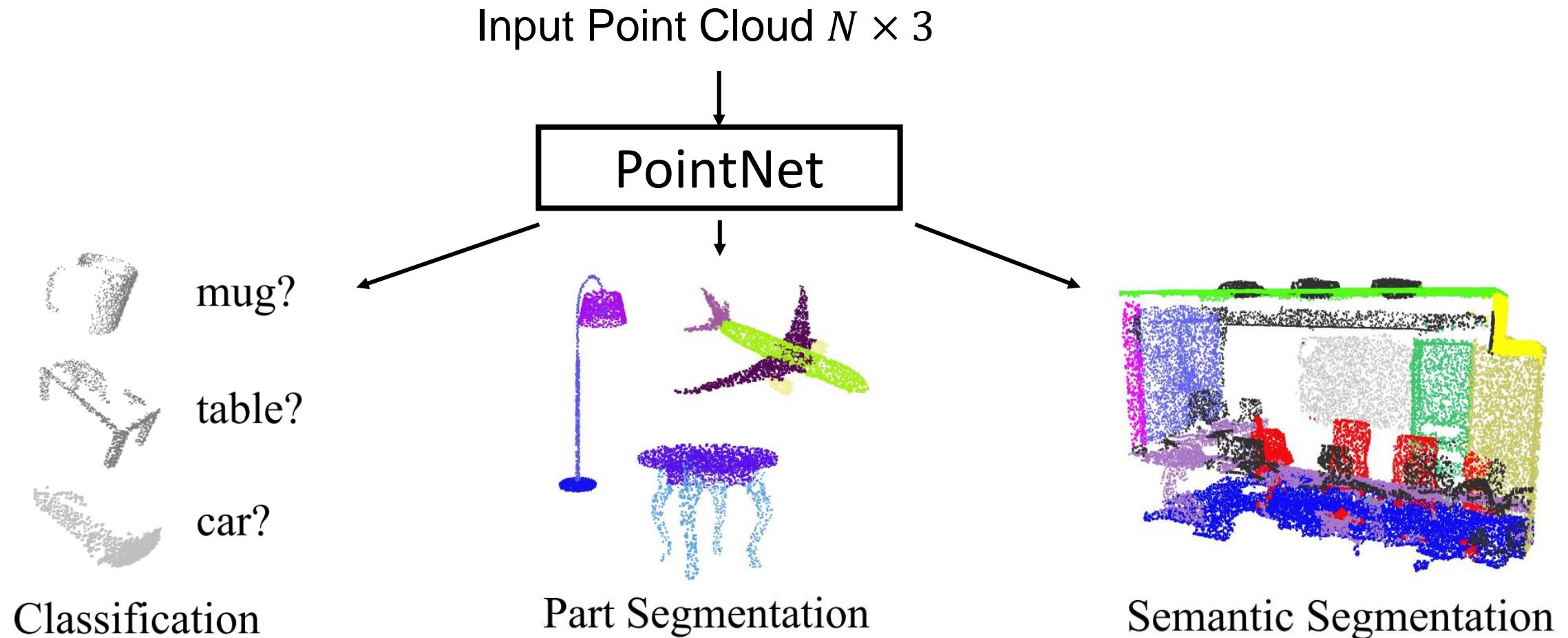
:

:

$$X_{N!} = [x_i^T, x_j^T, \dots, x_n^T]^T$$

**$N!$  possibilities**

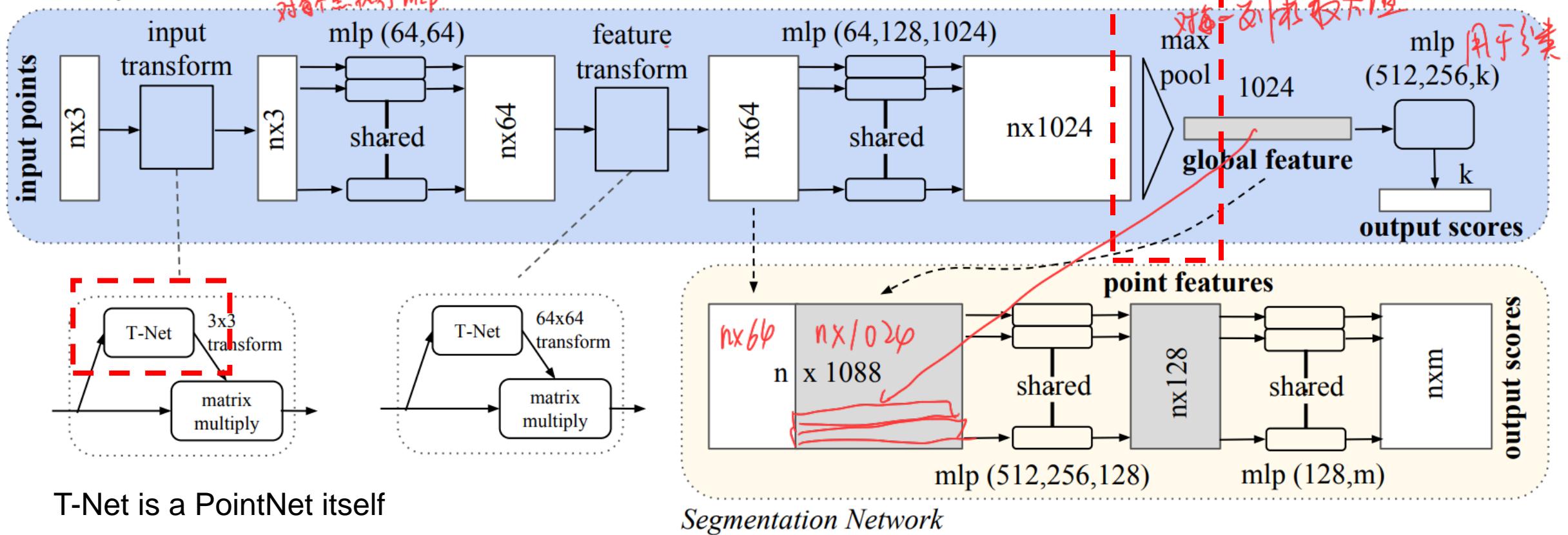
打乱顺序训练，  
并不去大意认同。





# PointNet Architecture

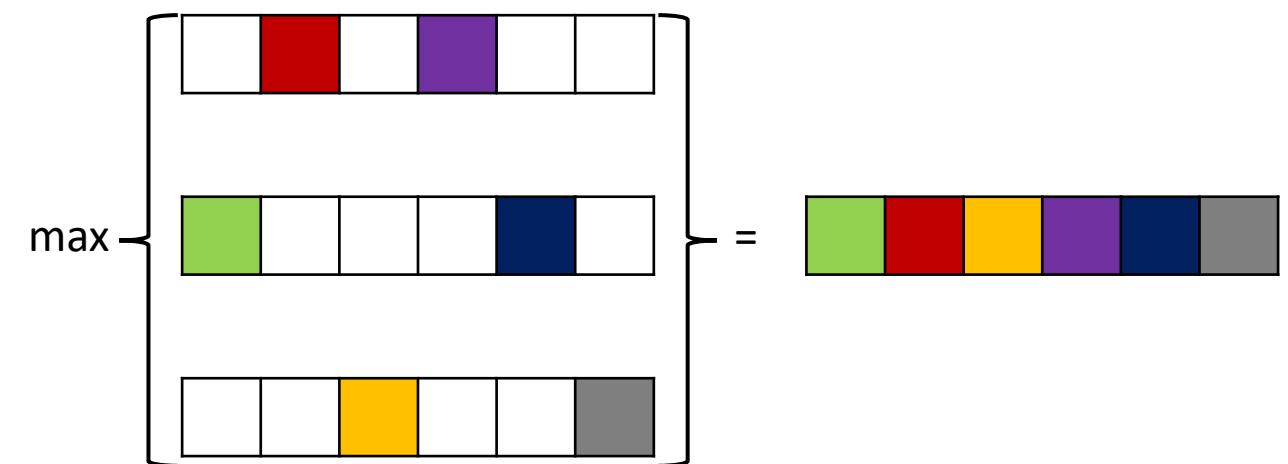
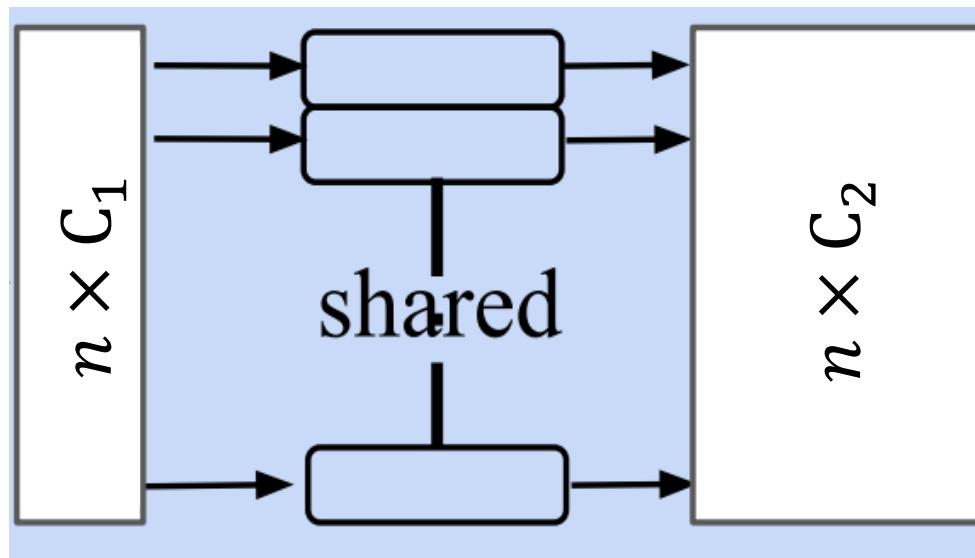
## Classification Network



T-Net is a PointNet itself



- Process each point (feature) **independently**  $n \times C_1 \rightarrow n \times C_2$
- Use **Max/Average** to pool the features  $n \times C \rightarrow 1 \times C$





## Proof – PointNet is able to simulate any function on the point cloud

### Some concepts:

- Input points  $S = \{x_1, \dots, x_n\}, x_i \in \mathbb{R}^m, x_i \in [0, 1]$ .
  - Denote the space of  $S$  as  $\chi$ , i.e.,  $S \in \chi$
- A continuous function  $f: \chi \rightarrow \mathbb{R}$ 
  - This is the “any function” we want to simulate
- *MAX* function: takes  $n$  vectors, give element-wise maximum

→ 1|2-1b



- Given continuous  $f: \chi \rightarrow \mathbb{R}$  *(连续函数)* ~~无间断~~
- $\forall \epsilon > 0, \exists h: \mathbb{R}^m \rightarrow \mathbb{R}^{m'}, \text{and } \gamma: \mathbb{R}^n \rightarrow \mathbb{R}$
- s.t.  $\forall S \in \chi, e.g. S = \{x_1, \dots, x_n\}, x_i \in \mathbb{R}^m$ 
$$\left| f(S) - \gamma \left( MAX(h(x_1), \dots, h(x_n)) \right) \right| < \epsilon$$

MLP for global feature                                  Shared MLP



$$\left| f(S) - \gamma \left( \text{MAX}(h(x_1), \dots, h(x_n)) \right) \right| < \epsilon$$

•  $h(\cdot)$  maps  $x_i$  to some deterministic position of a huge vector

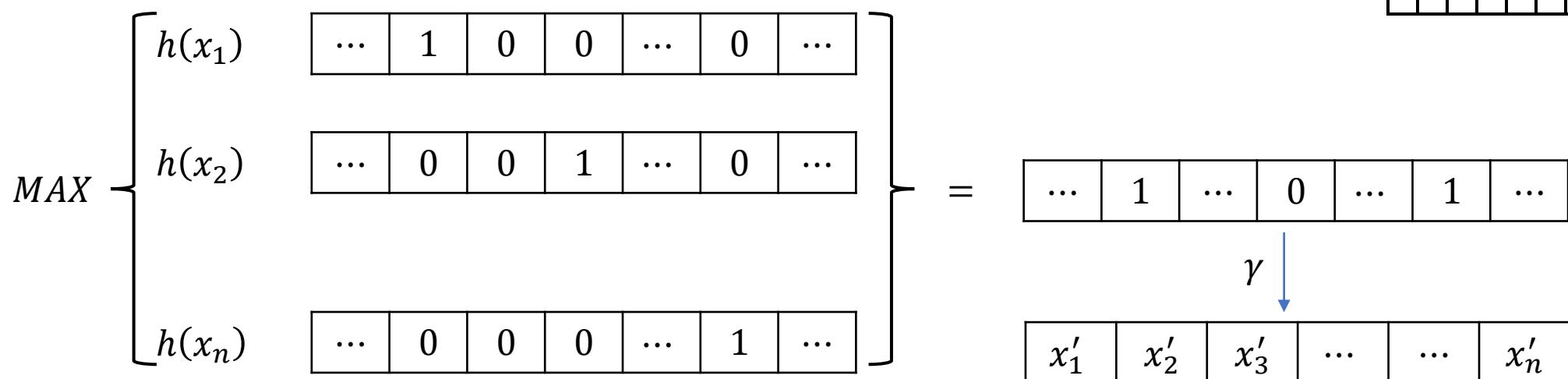
- By **Voxel Grid DownSampling**

•  $\text{MAX}(h(x_1), \dots, h(x_n))$  simply builds a voxel grid representation.

- There will be lots of 0 elements because of empty cells in voxel grid.

•  $\gamma(\cdot) = \text{reconstruct the points} + f(\cdot)$

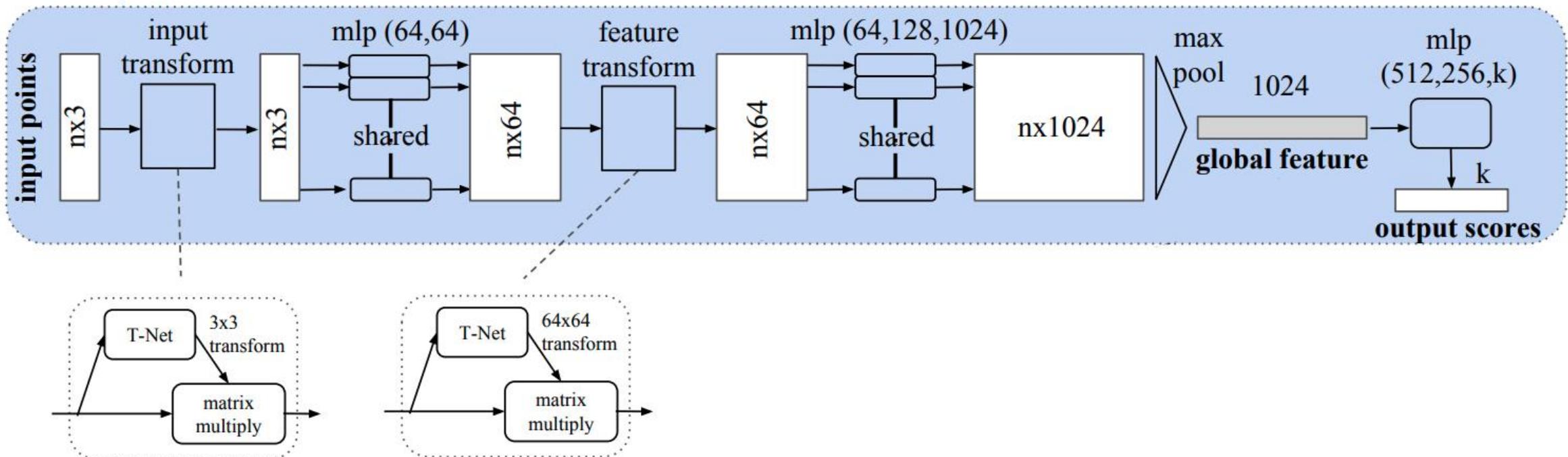
• Done





## PointNet – Classification

*Classification Network*





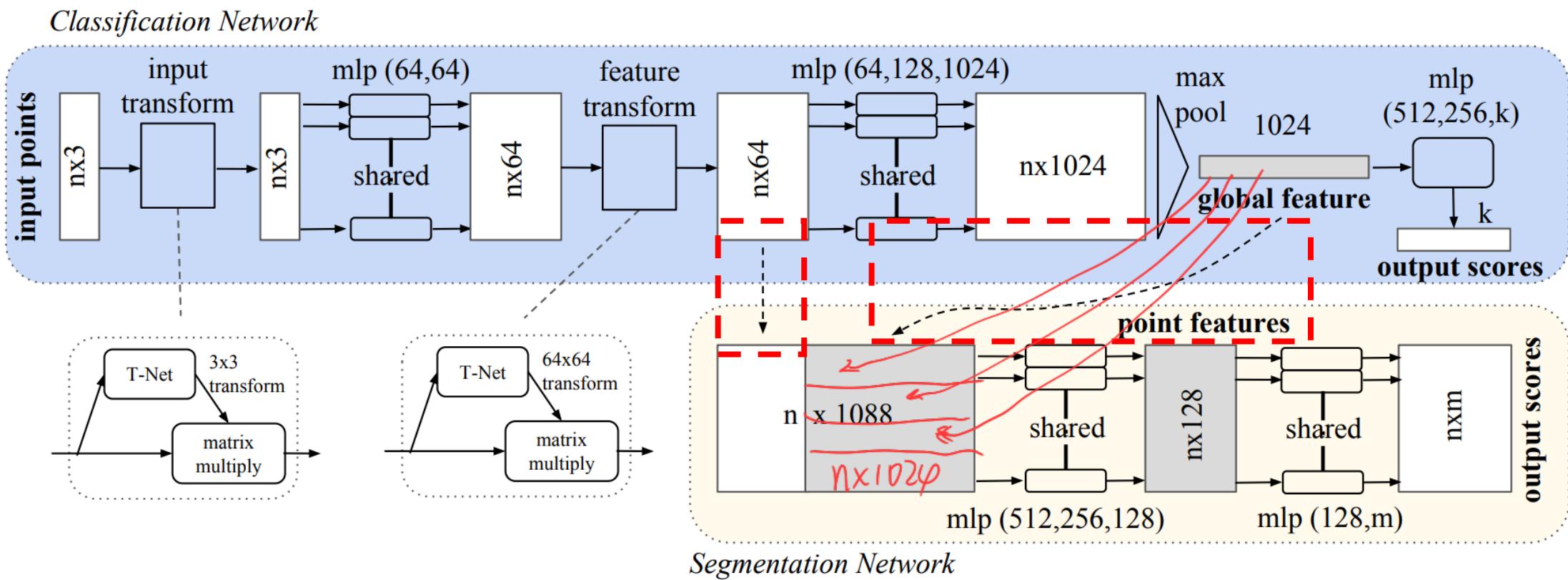
	input	#views	accuracy avg. class	accuracy overall
SPH [11]	mesh	-	68.2	-
3DShapeNets [28]	volume	1	77.3	84.7
VoxNet [17]	volume	12	83.0	85.9
Subvolume [18]	volume	20	86.0	<b>89.2</b>
LFD [28]	image	10	75.5	-
MVCNN [23]	image	80	<b>90.1</b>	-
Ours baseline	point	-	72.6	77.4
Ours PointNet	point	1	86.2	<b>89.2</b>

Table 1. **Classification results on ModelNet40.** Our net achieves state-of-the-art among deep nets on 3D input.



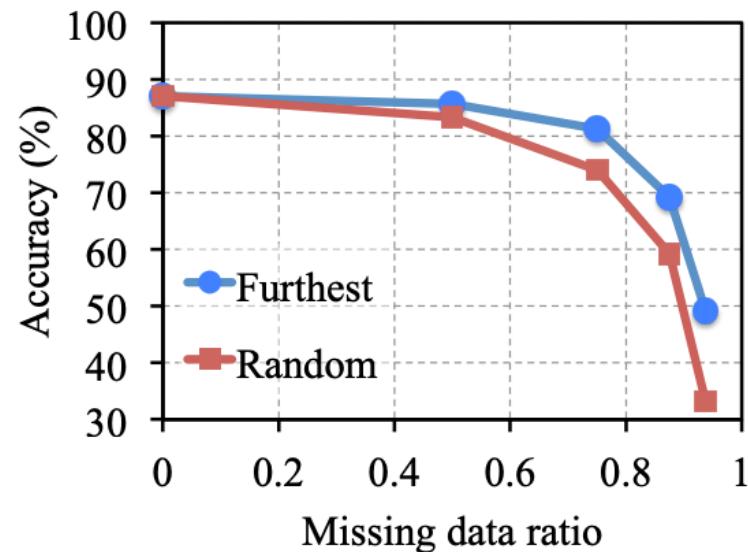
## Segmentation is **per-point** classification

- MLP on per-point feature, instead of global feature
- How to get per-point feature?

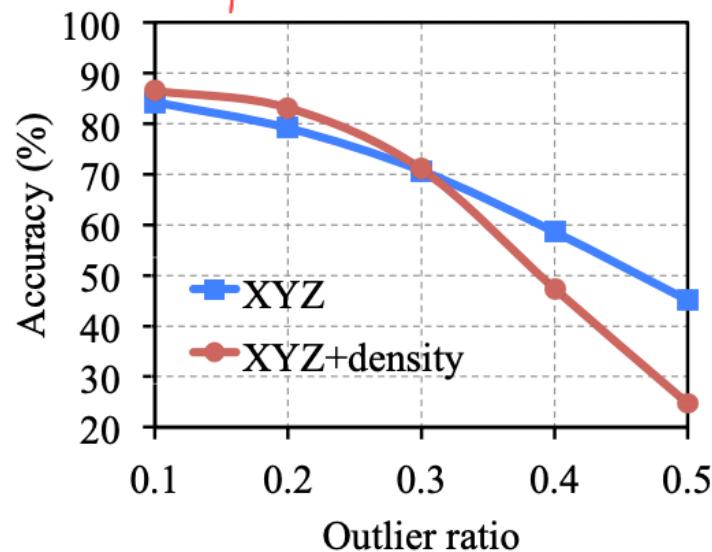




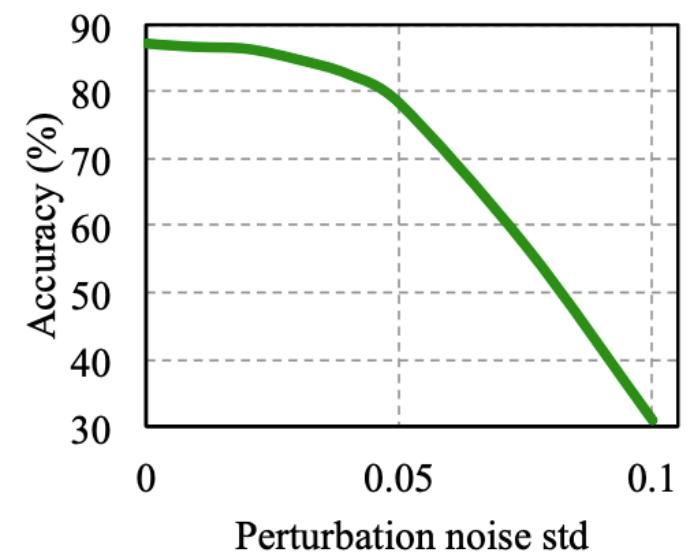
## PointNet – Robustness



Left: Downsample points by random sampling / FPS (furthest point sampling)



Middle: Insert uniformly sampled points.



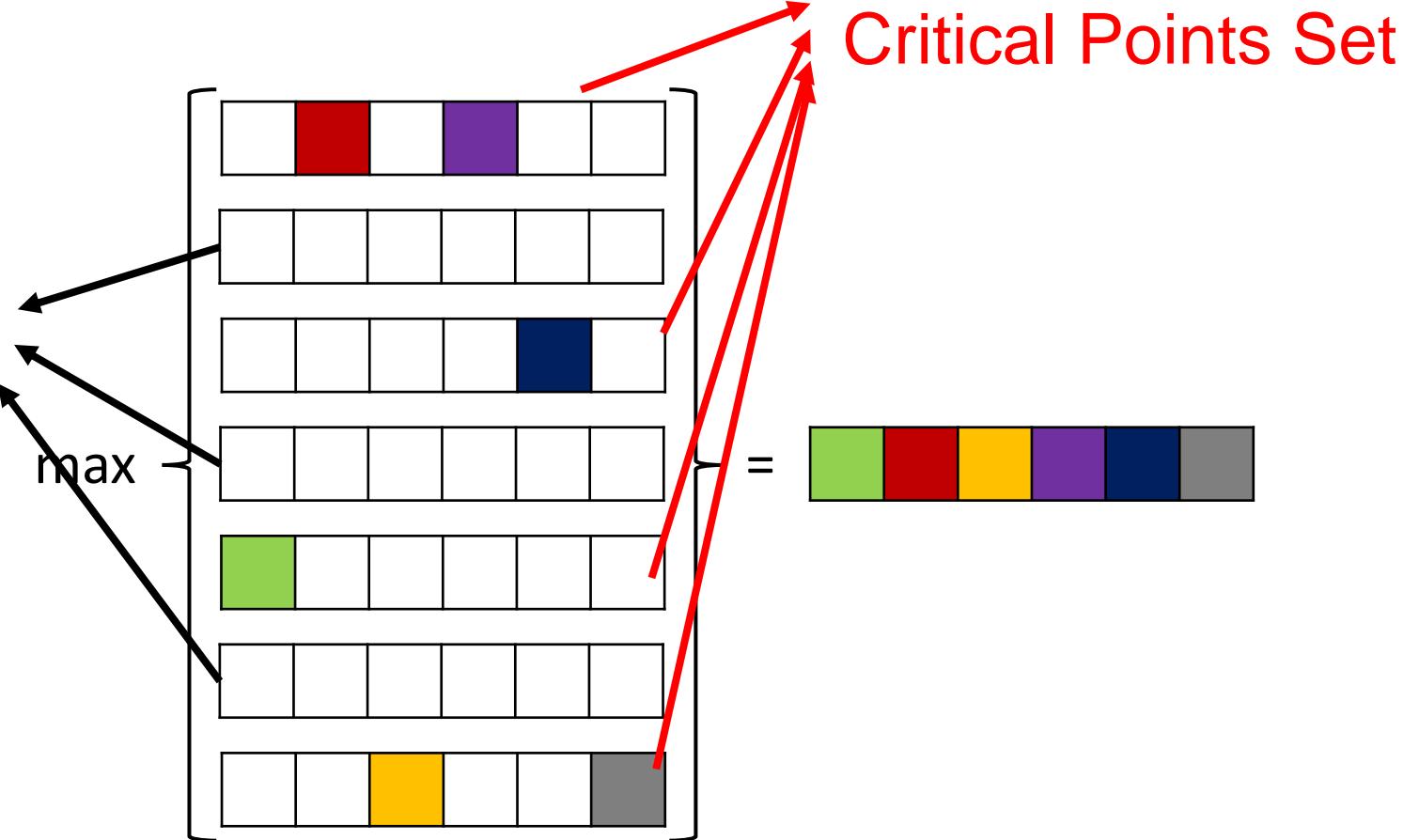
Right: Add Gaussian noise to input points



## Critical Points Set & Upper Bound Shape

### Upper Bound Shape

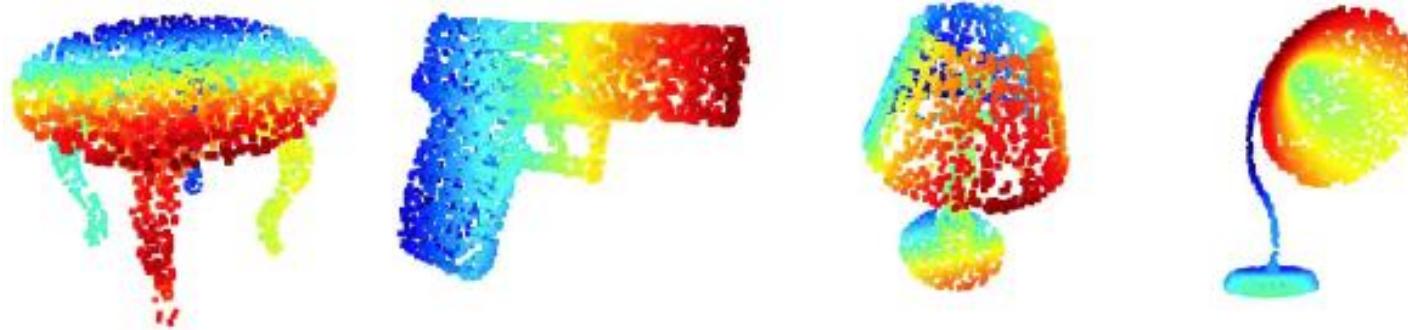
- Add those “useless” points



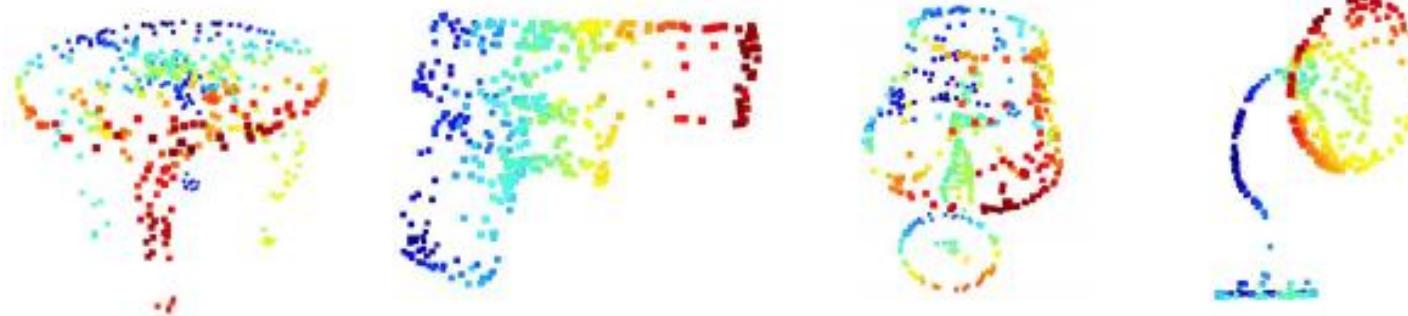


## Critical Points Set & Upper Bound Shape

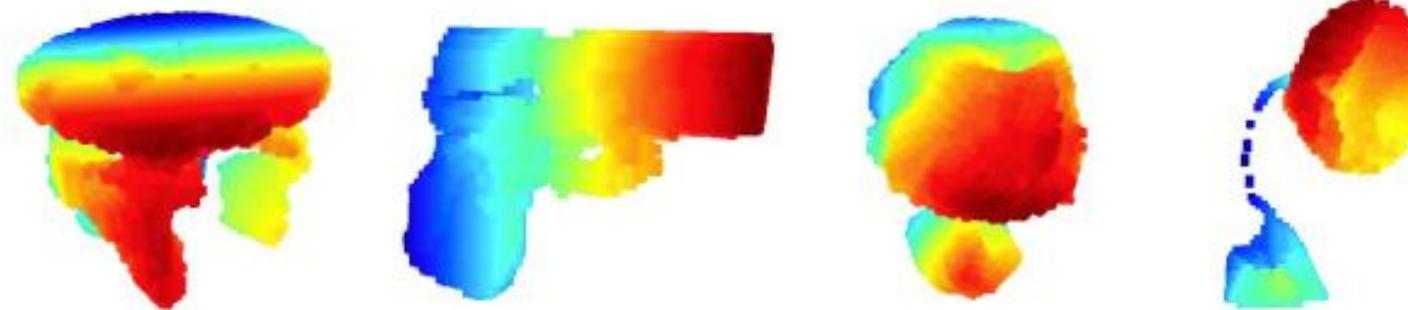
Original Shape



Critical Point Sets



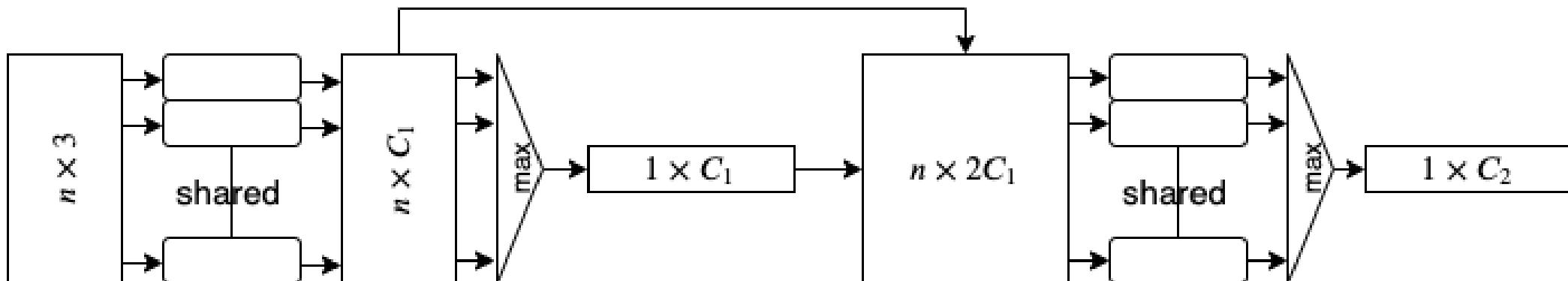
Upper Bound Shape





### • Voxel Feature Encoding (VFE)

- Proposed in VoxelNet, a 3D object detection in point cloud, Lecture 6
- Two PN in a row
- Performs better than PN

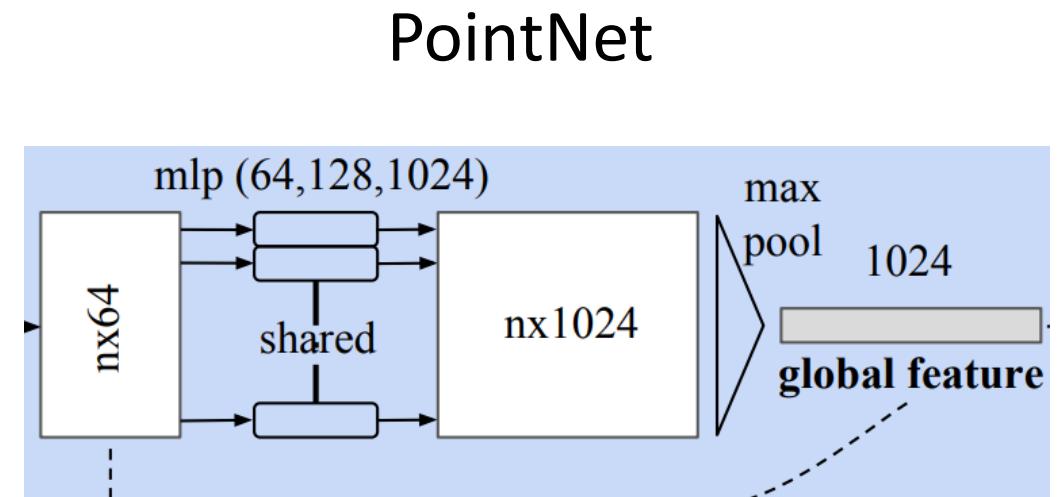
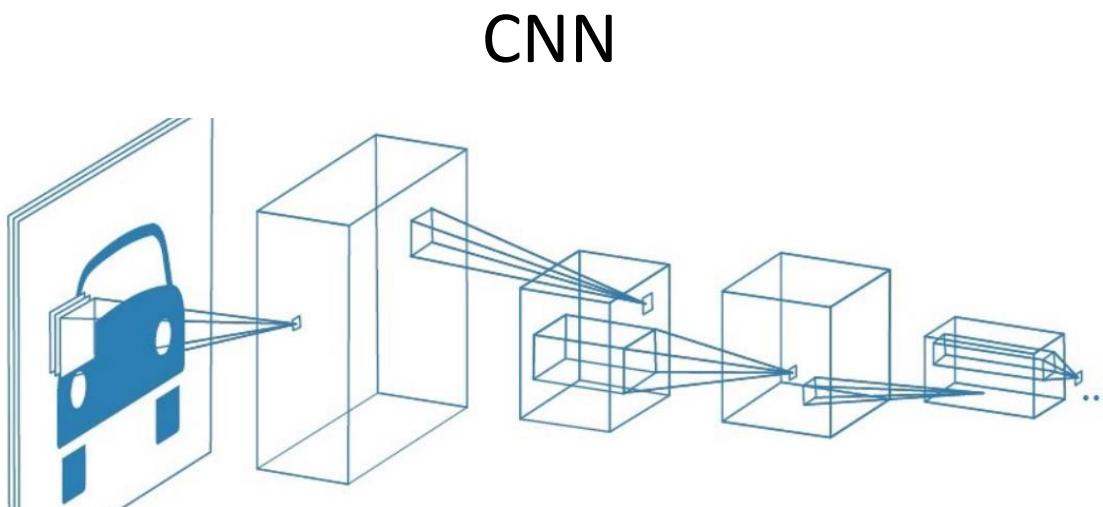


点云特征编码  
PointNet – Improved as VFE



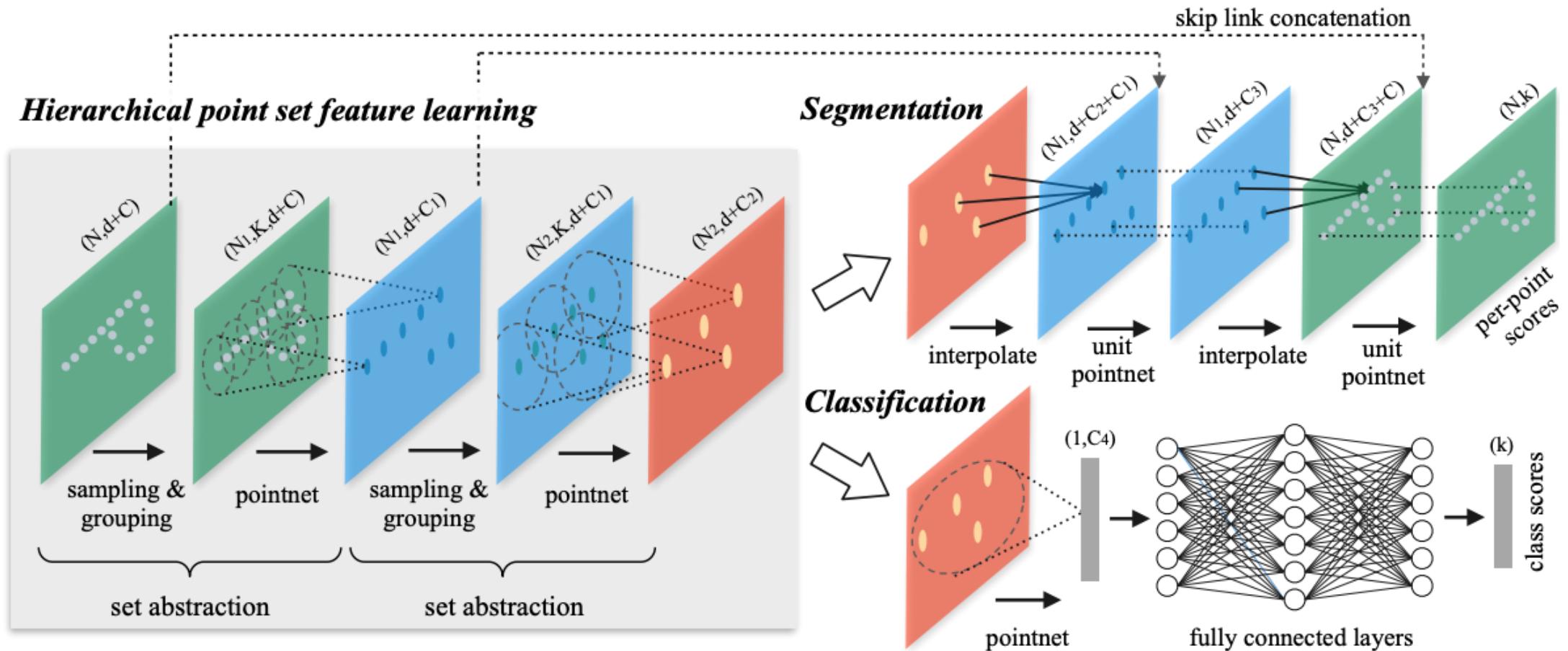
### • Lack of hierarchical feature aggregation

- CNN has multiple, increasing receptive field
- PointNet has one receptive field – all points



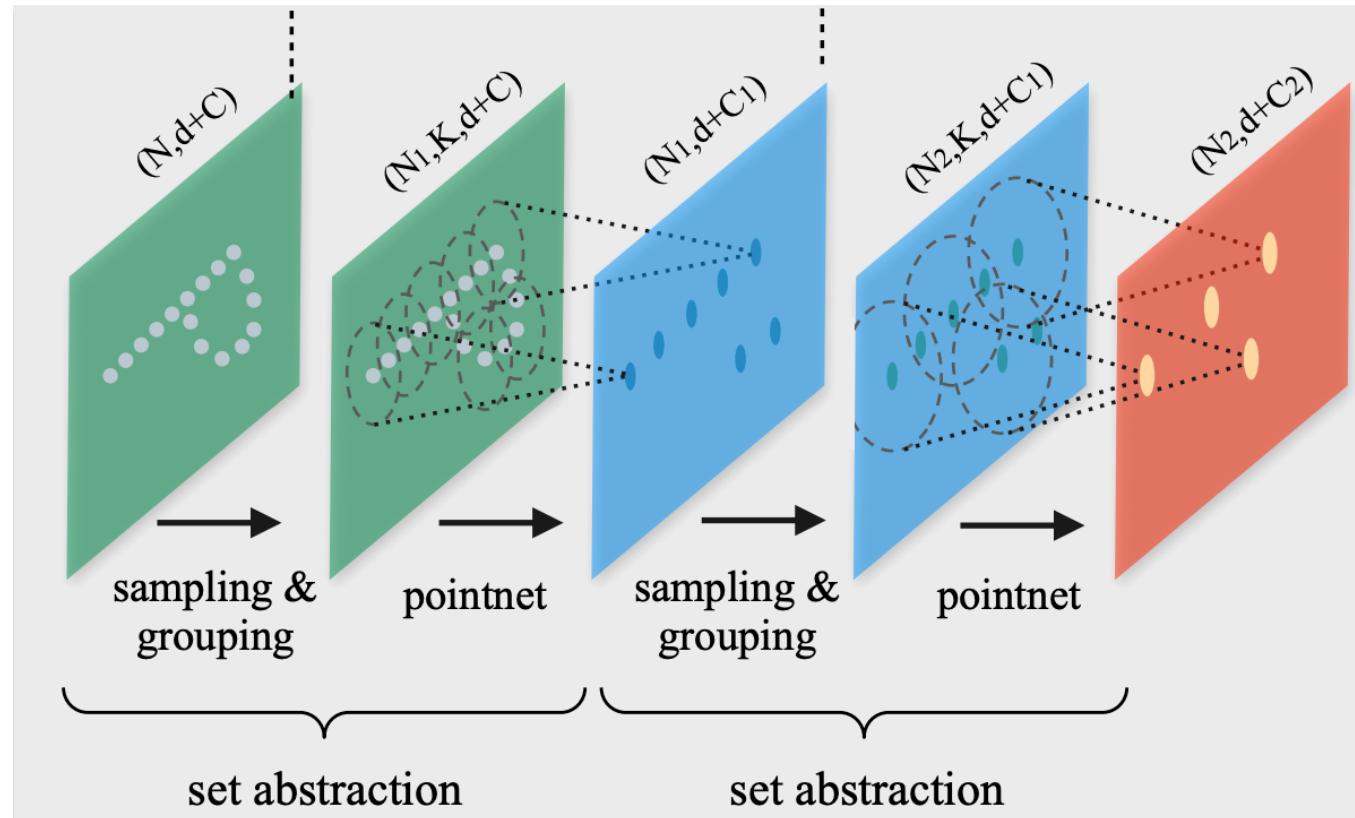


## PointNet++ - Looks complicated?





## PointNet++ - Hierarchical Features



- In each set abstraction:
  - Sampling: FPS
    - Point #:  $N_{i-1} \rightarrow N_i$
  - Grouping:
    - Radius Neighbors + random sampling
    - K Nearest Neighbors
- PointNet
  - Point #:  $N_i$
  - Channel #:  $C_{i-1} \rightarrow C_i$
  - Concatenate with coordinates so  $d + C_{i-1} \rightarrow C_i$
  - Normalize point coordinate in the group*
  - Centered with the Node*



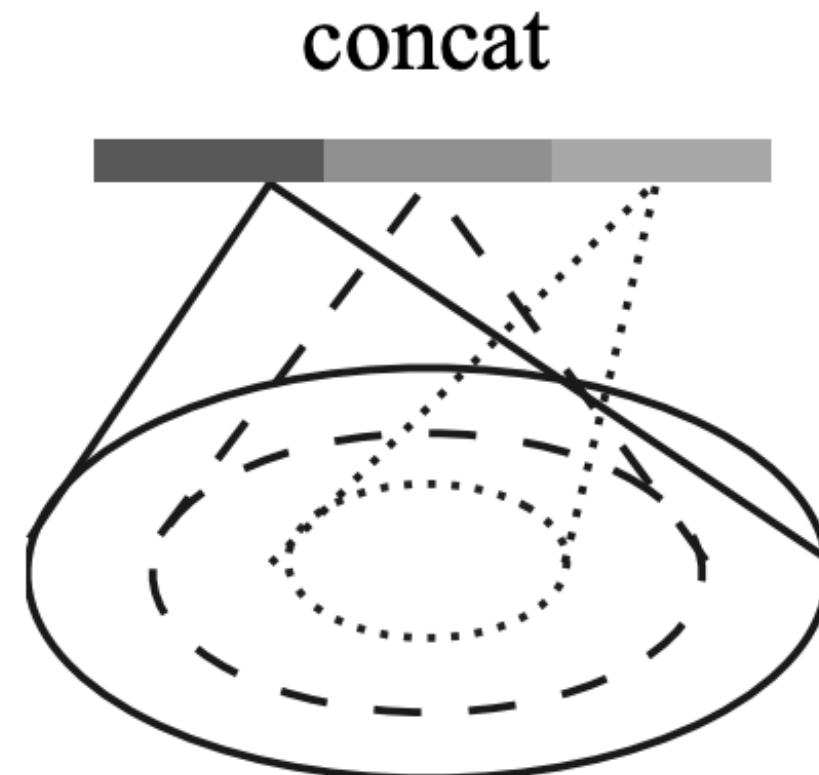
## Multi-scale grouping (MSG)

### One Sampling

### Multiple Grouping & PointNet

- $r = 0.1$  grouping + PN
- $r = 0.2$  grouping + PN
- $r = 0.4$  grouping + PN
- This is compute intensive

### Concatenate the multi-scale feature vectors

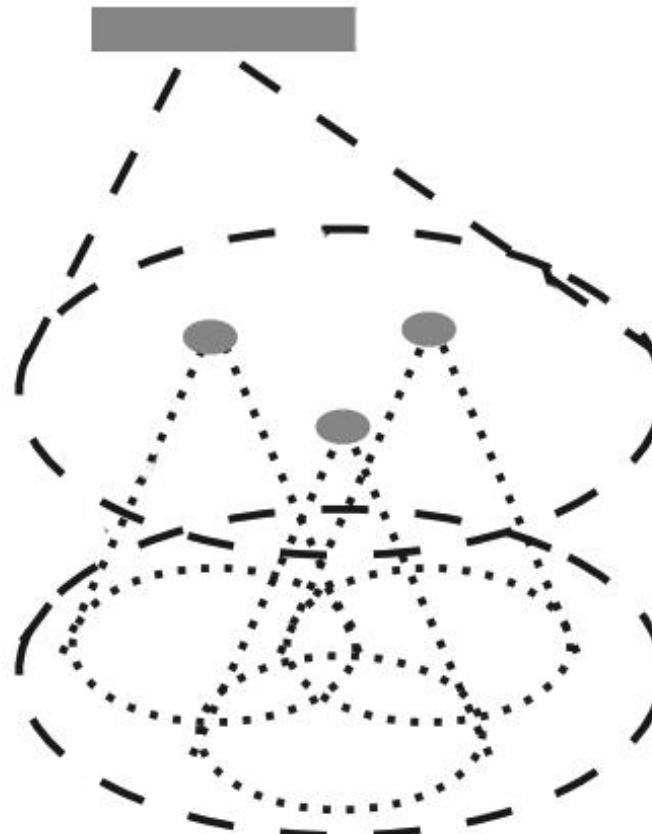




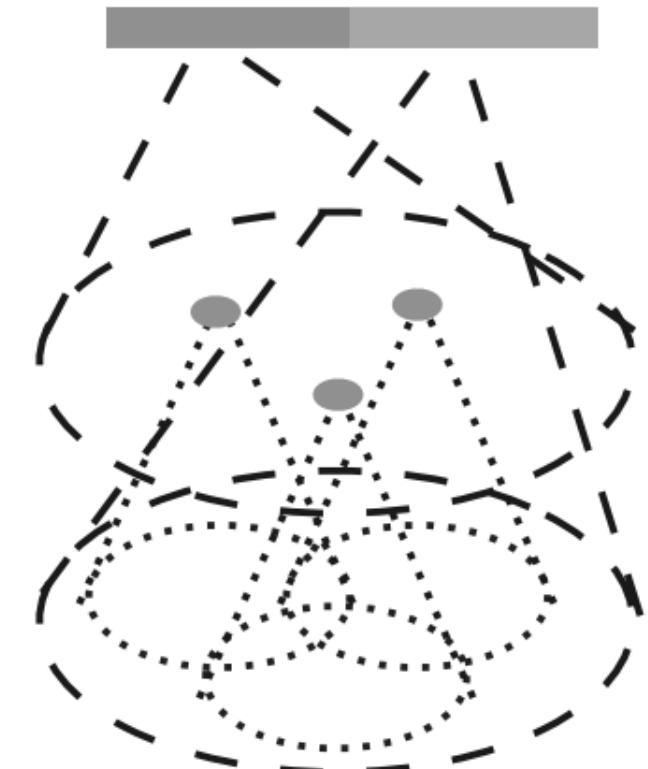
### Get features from

- previous level
- previous previous level
- Still increases compute

Simple PN++

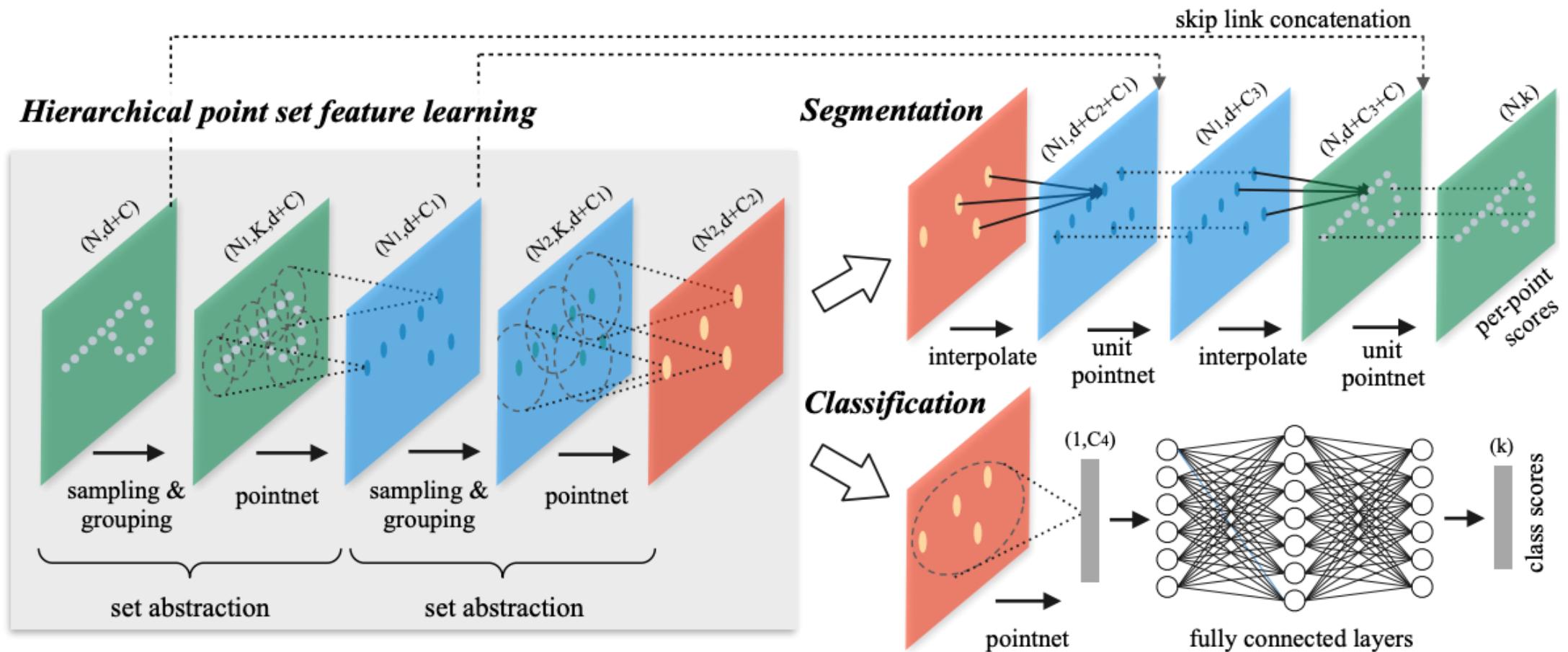


MRG PN++  
**concat**





## PointNet++ - Segmentation





### Interpolation

- Upsample the features from previous layer

•  $x \in \mathbb{R}^3$ : point coordinates at the upsampled level,  $\# = N_1$

•  $f \in \mathbb{R}^{C_2}$ : interpolated features

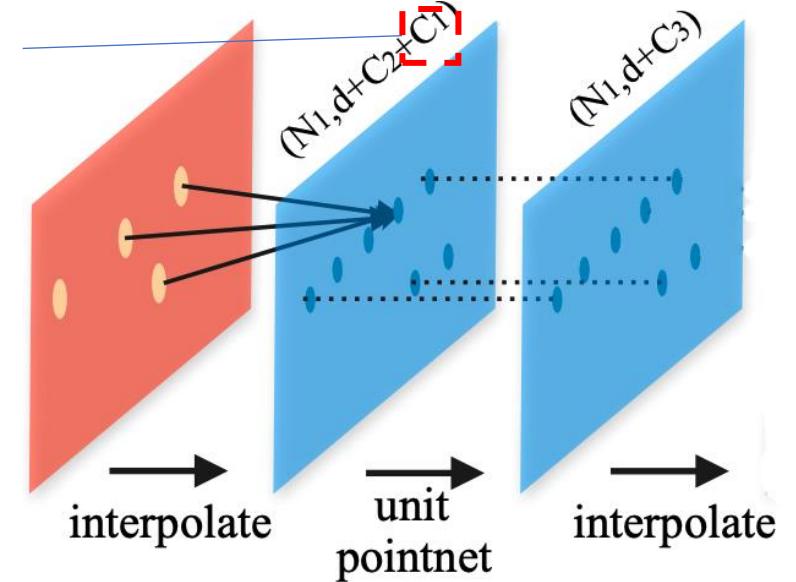
•  $x_i \in \mathbb{R}^3$ : point coordinates at the previous level ( $N_2$  points)

•  $w_i \in \mathbb{R}$ : reciprocal of distance  $d(x, x_i)$

•  $f_i \in R^{C_2}$  : point features at the previous level

$$f^{(j)}(x) = \frac{\sum_{i=1}^k w_i(x) f_i^{(j)}}{\sum_{i=1}^k w_i(x)} \quad \text{where} \quad w_i(x) = \frac{1}{d(x, x_i)^p}, \quad j = 1, \dots, C$$

From encoding stage





## • PointNet vanilla: without T-Net

- Same as a single layer in PN++

## • There isn't T-Net in PN++

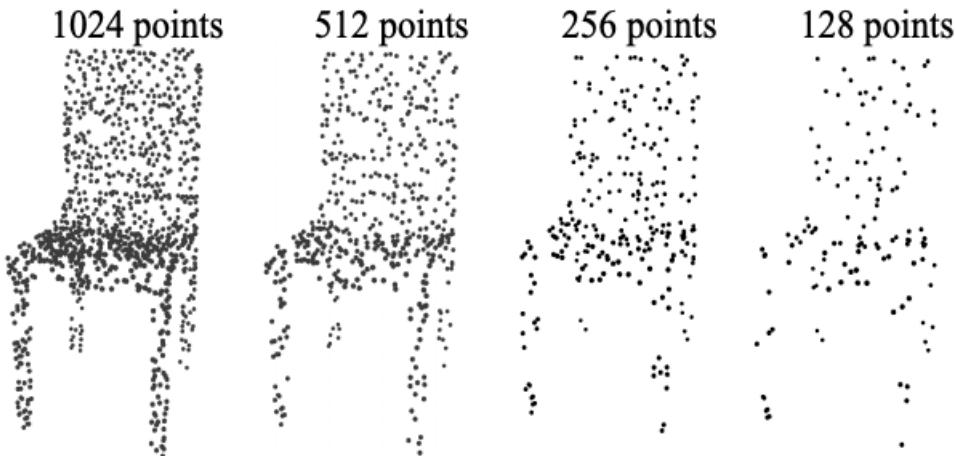
Method	Input	Accuracy (%)
Subvolume [21]	vox	89.2
MVCNN [26]	img	90.1
PointNet (vanilla) [20]	pc	87.2
PointNet [20]	pc	89.2
Ours	pc	90.7
Ours (with normal)	pc	<b>91.9</b>

Table 2: ModelNet40 shape classification.

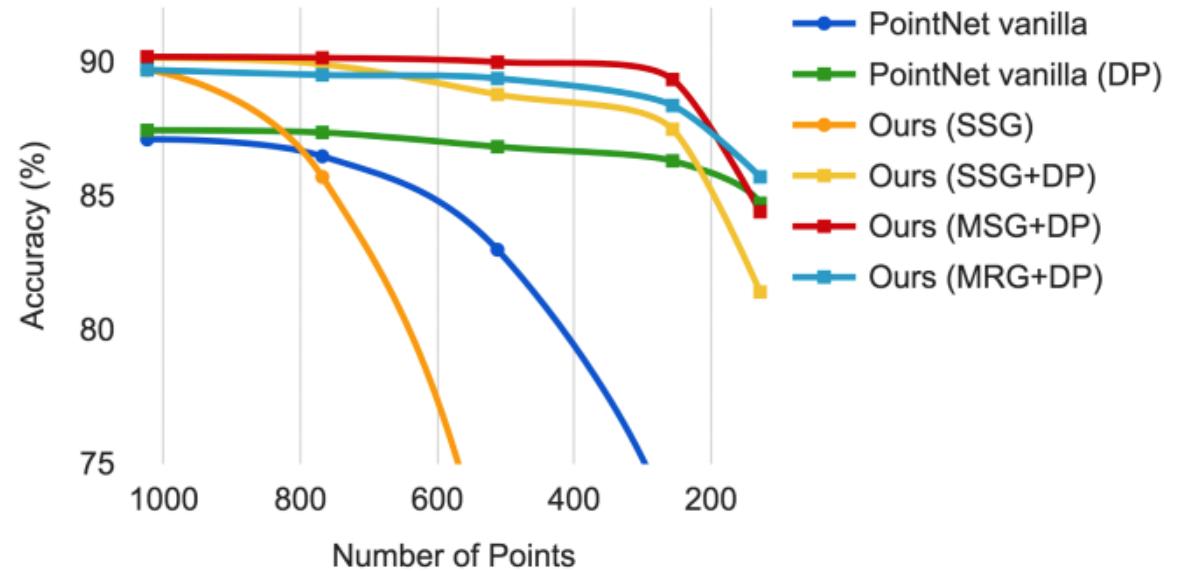


- MSG / MRG collects features from multiple scale / resolution
- Performs better with low intensity

Random Downsample on ModelNet40



Classification on Downsampled Objects





### PointNet

- First method to process point cloud with deep learning
- Lack of hierarchical feature aggregation

### PointNet++

- Hierarchical feature aggregation by repeating *sampling-grouping-PointNet*
- Significantly better performance
- Requires slightly more compute



## Tricks – Data Augmentation

增广

### Normalization

- PointNet:
  - Normalized input to zero-mean
- PointNet++:
  - centered-with-node
  - zero-mean

### Input Point Dropout

- E.g. Max input point number 5000, randomly dropout to [100, 5000] in each batch

### Gaussian Noise

### Rotation ???

- Less overfitting
- Worse performance
- Rotation equivariance / invariance is a research topic



## Homework - Classification



- Classification over ModelNet40

- Build your own network with pytorch
    - PointNet example: <https://github.com/fxia22/pointnet.pytorch>
  - ModelNet40 Dataset given by PointNet++:  
[https://shapenet.cs.stanford.edu/media/modelnet40\\_normal\\_resampled.zip](https://shapenet.cs.stanford.edu/media/modelnet40_normal_resampled.zip)
  - Follow the training/testing split
  - Remember to add random rotation over z-axis *加旋转*
  - Report testing accuracy
- 
- If you are not familiar with pytorch / deep learning
    - Simply run the open-source code to train / test on the ModelNet40 dataset.
    - Report the testing accuracy.
  - Please write a report with screenshots to show the training/testing loss/accuracy curve.



### Object detection pipeline for lidar

- Use KITTI 3D object detection dataset
- Step 1. Remove the ground from the lidar points
  - Any method you want – LSQ, Hough, RANSAC
- Step 2. Clustering over the remaining points
  - Any method you want
- **Step 3. Classification over the clusters**
- Step 4. Report the detection precision-recall for three categories: vehicle, pedestrian, cyclist  
(Next Lecture)



### Step 3. Classification over the clusters

- vehicle / pedestrian / cyclist / other
  - Step 3.1. Build dataset from KITTI 3D object detection dataset
    - Extract objects in the box for vehicle / pedestrian / cyclist
    - Other objects? Clustering! *一分为四类*
  - Step 3.2. Build deep network to classify them.
  - Step 3.3. Report classification accuracy

### Step 4. Report the detection Precision-Recall for three categories: vehicle, pedestrian, cyclist

- Fit a cuboid over object detected as vehicle / pedestrian / cyclist
- Generate object detection results for KITTI
- Evaluation code provided next Lecture.