

Deep Learning lecture 2 Supervised Learning (1)

Yi Wu, IIIS

Spring 2025

Feb-24

An overview of Lecture 1

- History and big names in deep learning
 - From Boolean circuits (starting from MP neuron in 1943) to differentiable networks
 - Backpropagation (1986) : first time to show neural network can learn features
 - The fundamental idea
 - Breakthrough in 2012
 - Speech Recognition
 - Unsupervised learning of concepts (cat)
 - Image classification

An overview of Lecture 1

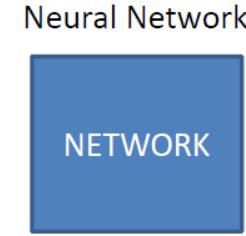
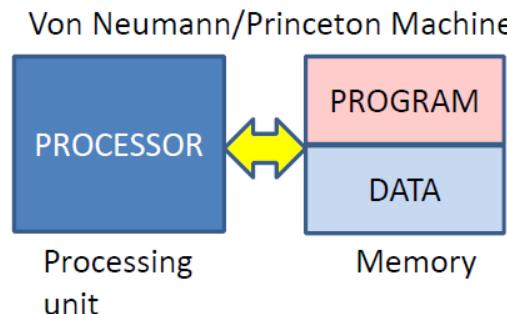
- History and big names in deep learning
 - From Boolean circuits (starting from MP neuron in 1943) to differentiable networks
 - Backpropagation (1986) : first time to show neural network can learn features
 - The fundamental idea
 - Breakthrough in 2012
 - **Speech Recognition → Today's lecture (basics)**
 - Unsupervised learning of concepts (cat) → lecture 4
 - **Image classification → Today's lecture (basics)**

An overview of Lecture 1

- What makes deep learning so special?
 - $y = f(\phi(x); \theta_f) \rightarrow y = f(NN(x; \theta_{NN}); \theta_f)$
 - Replace hand-tuned feature with a NN \rightarrow *representation learning*
 - **A differentiable model to learn features!**
 - We will also talk about discrete models in future lectures. ☺

An overview of Lecture 1

- What makes deep learning so special?
 - $y = f(\phi(x); \theta_f) \rightarrow y = f(NN(x; \theta_{NN}); \theta_f)$
 - A differentiable model to learn features!
 - It redefines a machine learning algorithm
 - Conventional ML algorithm: a few steps of computations (machine program)
 - Data → feature → algorithm/model → output
 - DL algorithm: the network architecture and weights (connectionist machine)
 - Learning → (1) set up the right architecture & (2) enforce the right weights from data
 - A new concept of **algorithm**: the network architecture



Today's Lecture

- Part 1: The simplest deep learning application ---- classification!
 - The very basic ideas of deep learning and backpropagation
 - Get a sense of parameter tuning (调参/炼丹)
- Part 2: Convolutional Neural Networks (CNN)
 - The very basic ideas of CNN
- Let's get a sense of deep learning “algorithm”
 - More tricks and ideas to come in the next lecture

Today's Lecture

- **Part 1: The simplest deep learning application ---- classification!**
 - The very basic ideas of deep learning and backpropagation
 - Get a sense of parameter tuning (调参/炼丹)
- Part 2: Convolutional Neural Networks (CNN)
 - The very basic ideas of CNN
- Let's get a sense of deep learning “algorithm”
 - More tricks and ideas to come in the next lecture

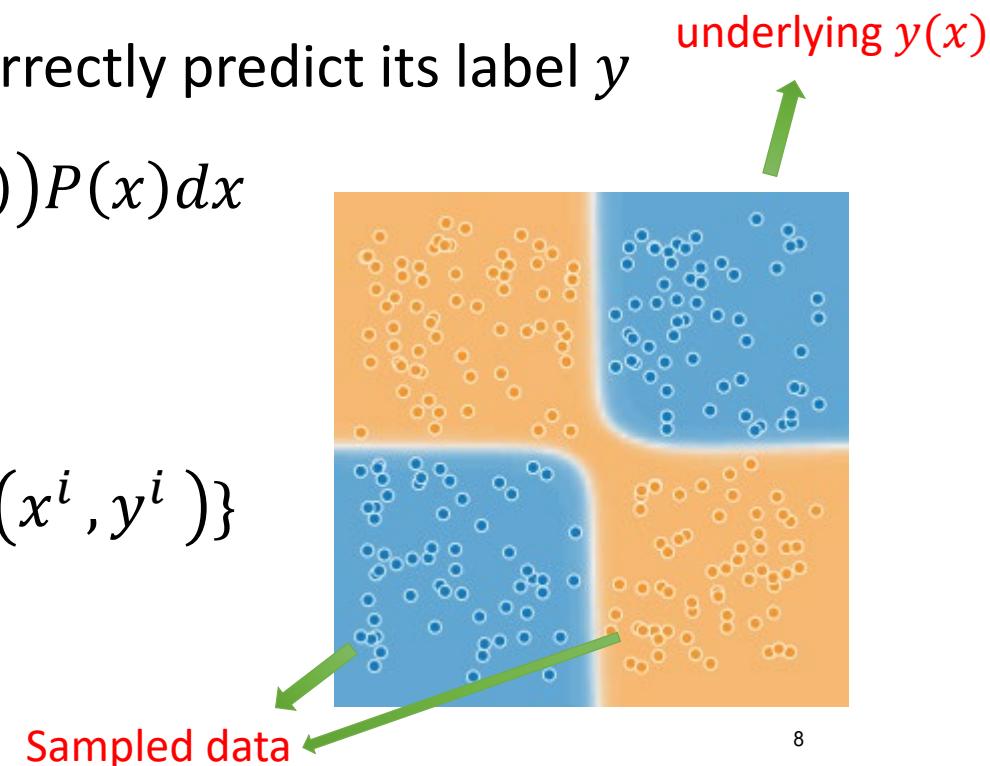
Recap: Classification

- Binary classification problem
 - Data $x \in \mathcal{X} \subseteq \mathbb{R}^d$, label $y(x) \in \{0,1\}$
 - a classifier $f(x; \theta) = y$ w.r.t. some parameter θ
 - Goal: learn θ^* such that for each x , $f(x)$ can correctly predict its label y

$$\theta^* = \arg \min_{\theta} \int_{\mathcal{X}} \text{err}(f(x; \theta), y(x)) P(x) dx$$

- Machine learning for classification

- Define a proper $\text{err}()$ function
- Estimate θ^* from a collection of samples $X = \{(x^i, y^i)\}$



Recap: Logistic Regression

- Logistic regression

- $P(y = 1) = f(x; \theta) = \sigma(w^T x + b)$

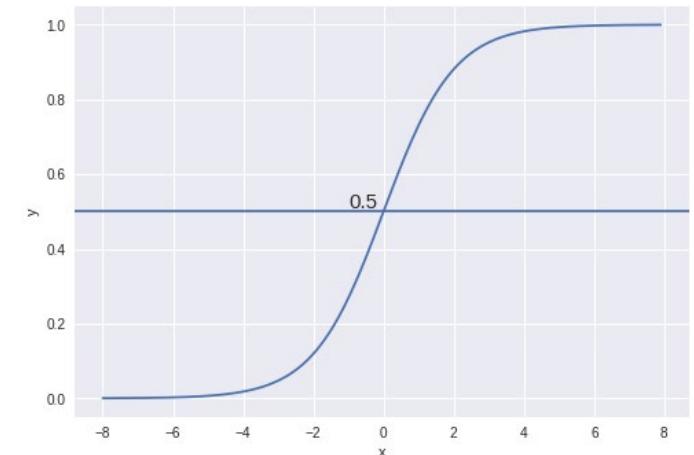
- Logistic function (sigmoid function): $\sigma(x) = \frac{1}{1+e^{-x}}$

- $err(f(x; \theta), y) = \begin{cases} -\log \sigma(w^T x + b) & y = 1 \\ -\log(1 - \sigma(w^T x + b)) & y = 0 \end{cases}$

- Learning from data

- Empirical Risk Minimization

$$\hat{\theta}^* = \arg \min_{\theta} \frac{1}{N} \sum_{(x^i, y^i) \in X} err(f(x^i; \theta), y^i)$$



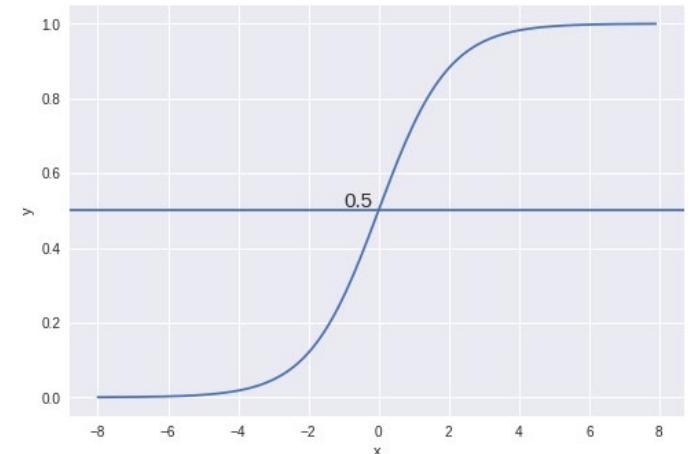
Recap: Logistic Regression

- Logistic regression

- $P(y = 1) = f(x; \theta) = \sigma(w^T x + b)$

- Logistic function (sigmoid function): $\sigma(x) = \frac{1}{1+e^{-x}}$

- $err(f(x; \theta), y) = \begin{cases} -\log \sigma(w^T x + b) & y = 1 \\ -\log(1 - \sigma(w^T x + b)) & y = 0 \end{cases}$



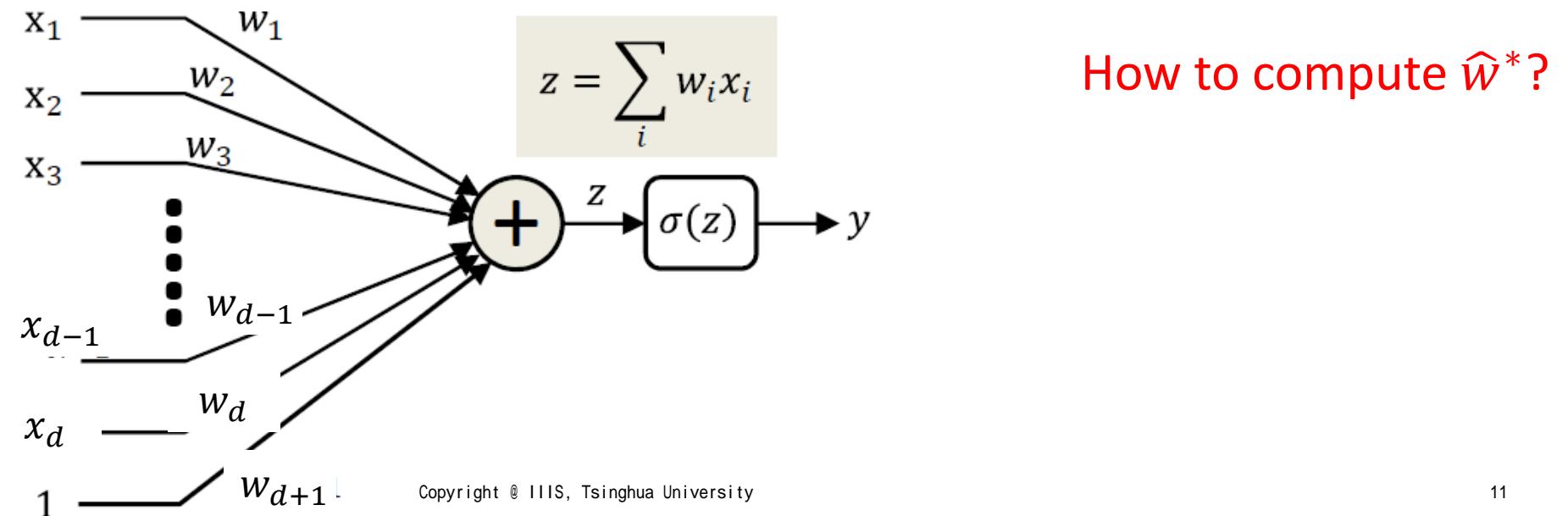
- Learning from data

- Empirical Risk Minimization

$$\hat{\theta}^* = \arg \min_{\theta} \frac{1}{N} \sum_{(x^i, y^i) \in X} err(f(x^i; \theta), y^i)$$

Single-Layer Perceptron

- A differentiable single-layer perceptron with sigmoid activation
 - $\sigma(z)$ is a differentiable function over z and therefore w and x
 - Learning: $\hat{w}^* = \arg \min_w \frac{1}{N} \sum_{(x^i, y^i) \in X} err(f(x^i; w), y^i)$
 - Minimize error on each training data

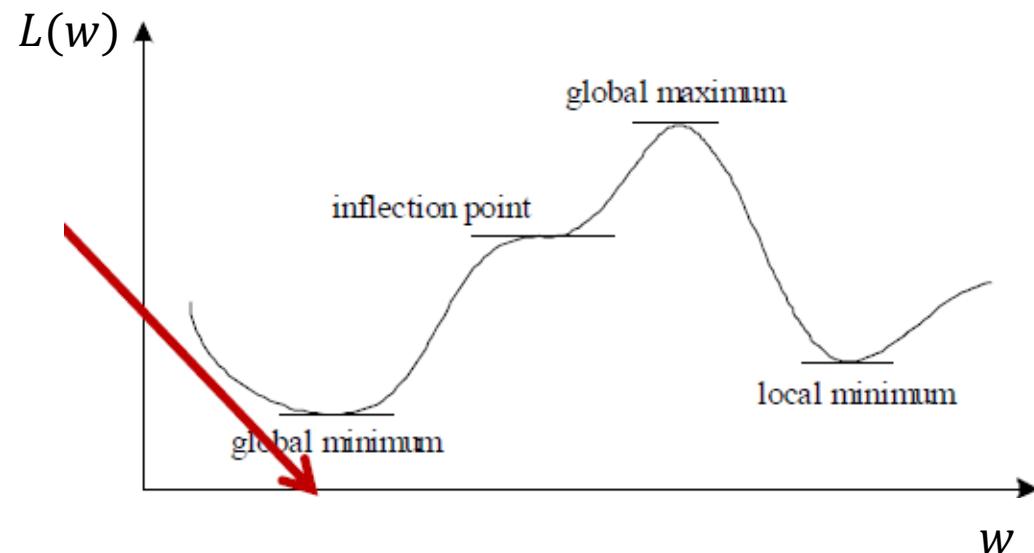


Single-Layer Perceptron

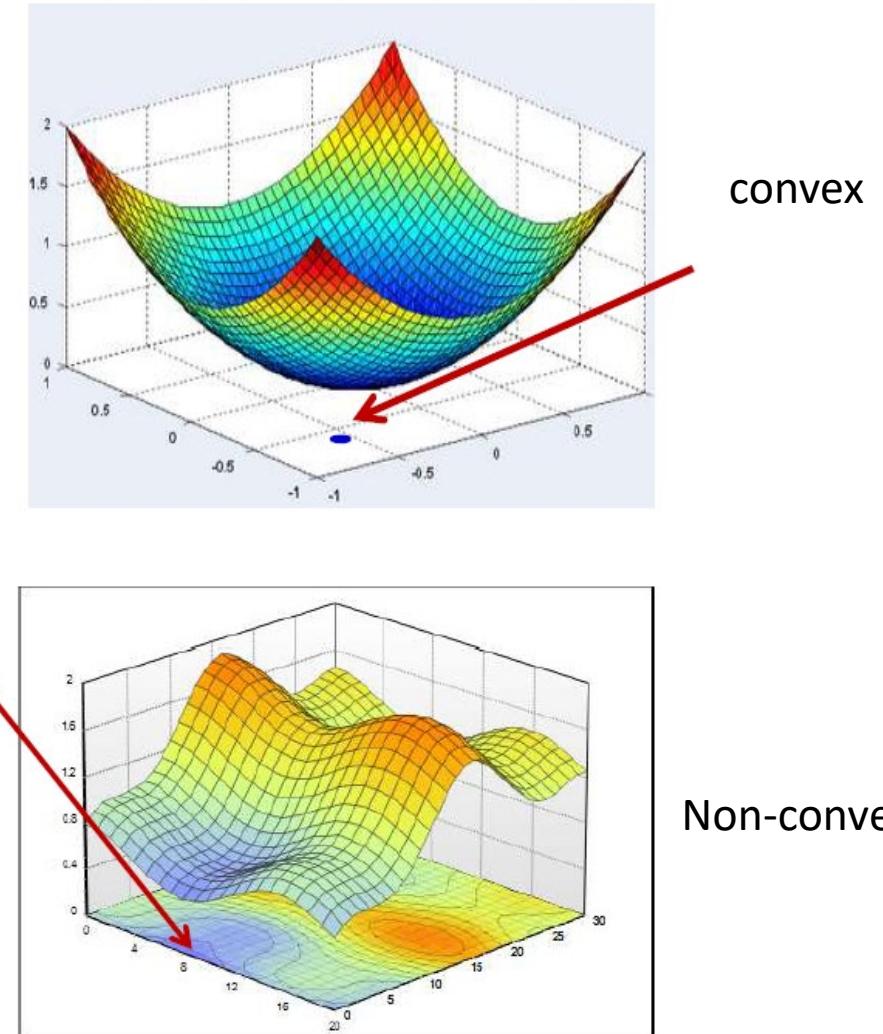
- Problem Statement
 - Given $X = \{(x^i, y^i)\}$
 - Loss function $L(w) = \frac{1}{N} \sum_i err(f(x^i; w), y^i)$
 - Goal: minimize $L(w)$ w.r.t. w
 - $L(w)$ is continuous and differentiable
- An instance of *optimization problem*
 - if $L(w)$ is convex \rightarrow convex optimization, we can find optimal solution
 - If $L(w)$ is non-convex \rightarrow non-convex optimization, no guarantee
 - Deep learning in general is tackling a non-convex optimization problem

Function Optimization

- Problem: minimize $L(w)$ w.r.t. w
- Solution: solve $\nabla L(w) = 0$



- How to ensure it is a (local) minimum?
- The Hessian $\nabla^2 L(w)$ is positive-definite!



Multi-Variable Calculus Recap: Gradient

- Consider $f(X) = f(x_1, x_2, \dots, x_n)$

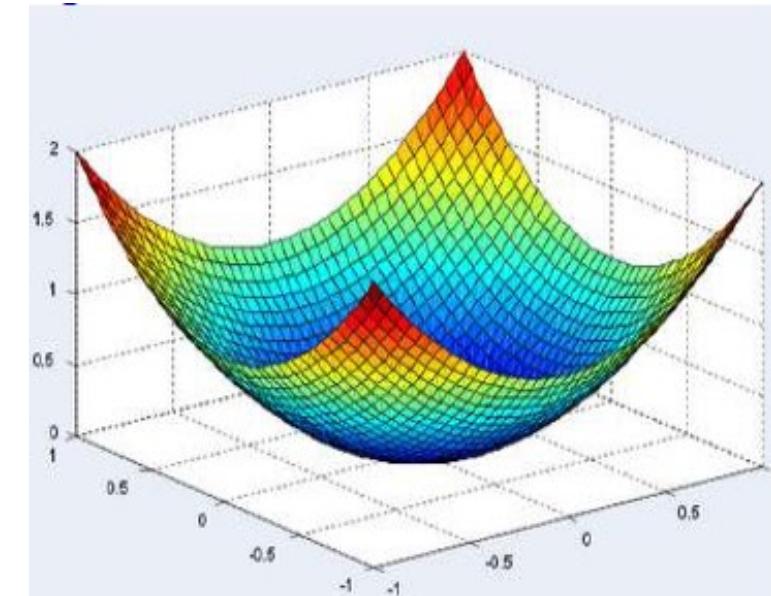
$$\nabla_X f(X) = \left[\frac{\partial f(X)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2}, \dots, \frac{\partial f(X)}{\partial x_n} \right]$$

- Relation

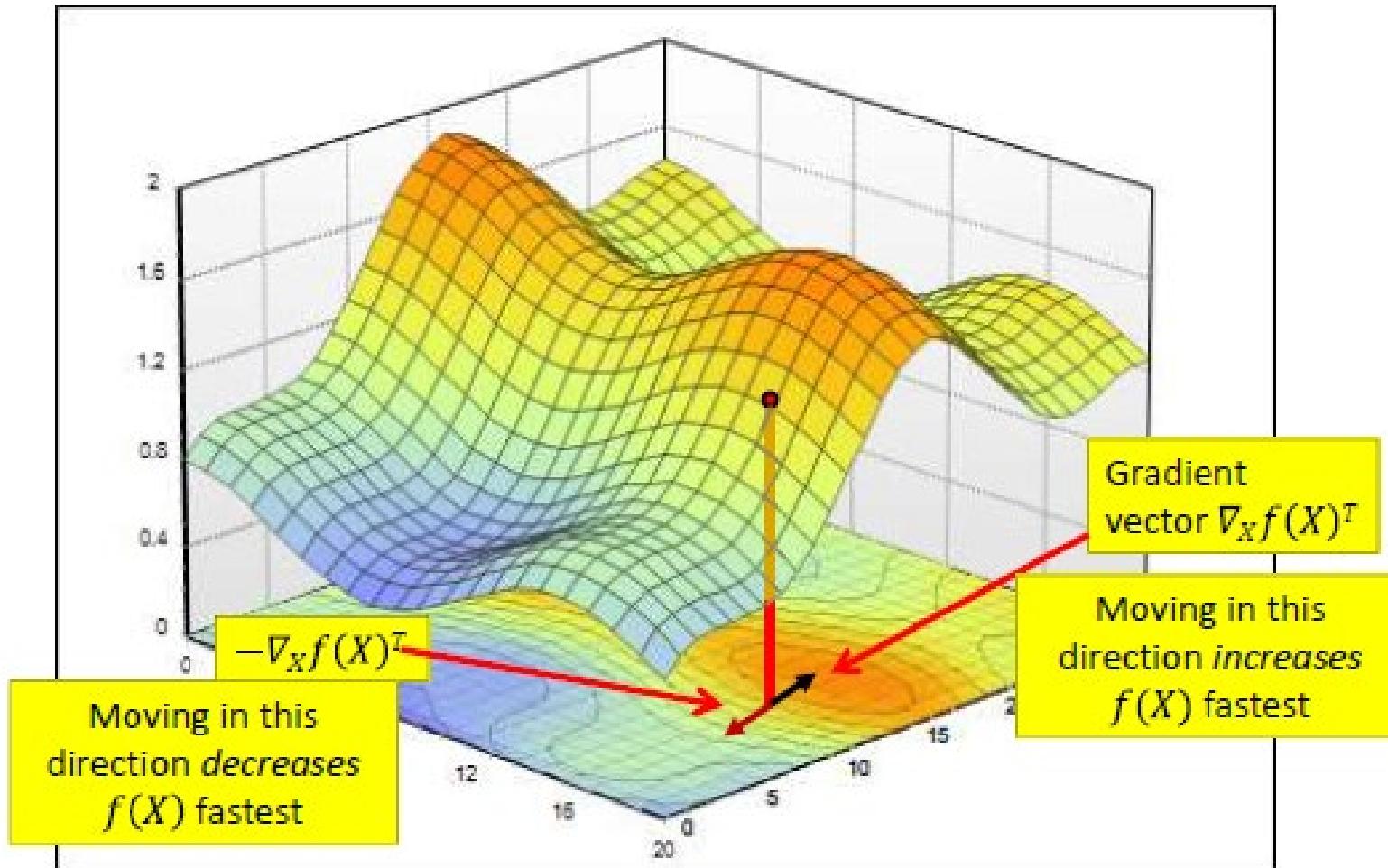
$$df(s) = \nabla_X f(X) dX = \sum_i \frac{\partial f(X)}{\partial x_i} dx_i$$

- Notations

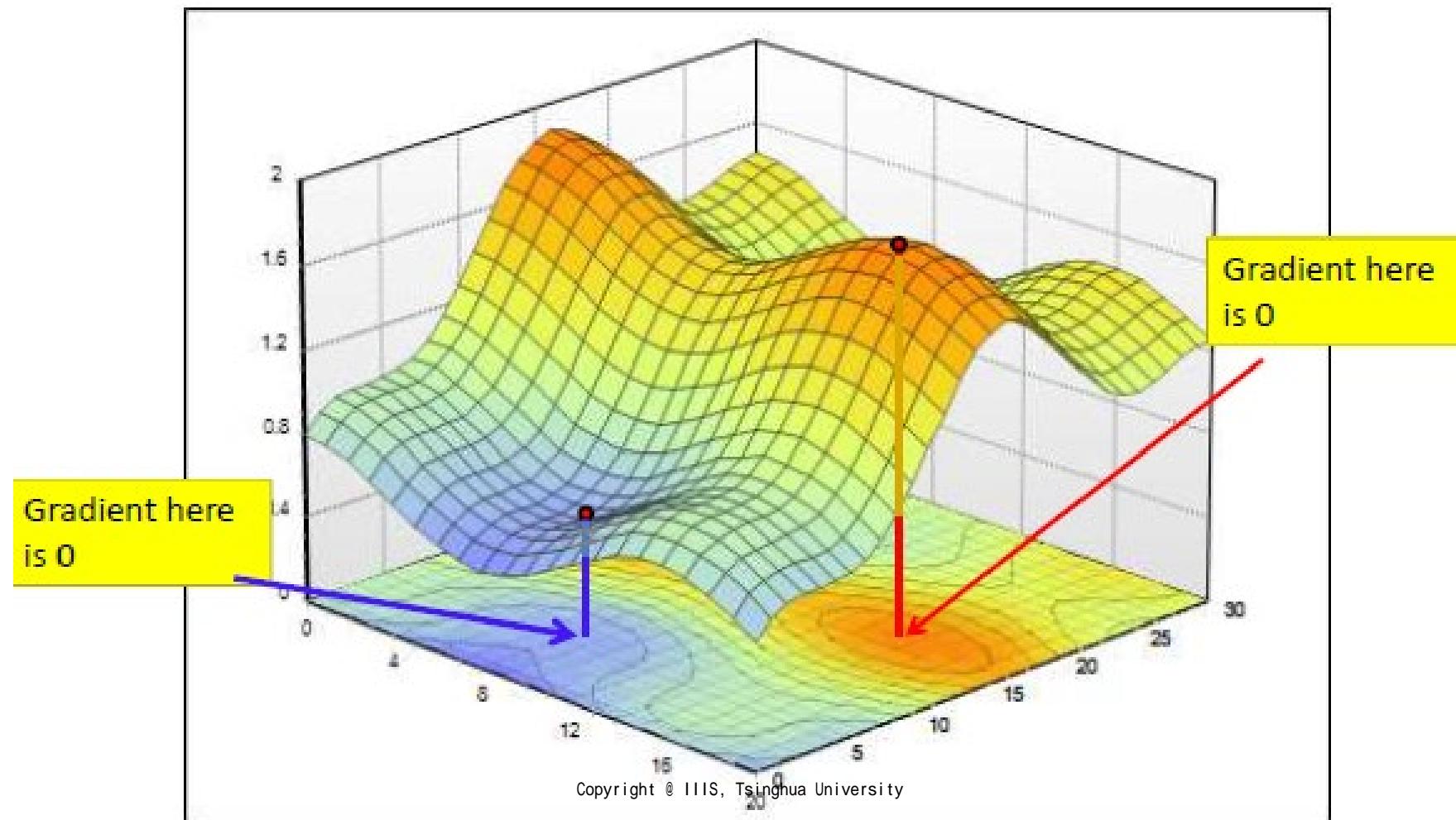
- Gradient (∇), derivative (d), partial derivative (∂)
- The gradient is the direction of fastest increase in $f(X)$



Multi-Variable Calculus Recap: Gradient

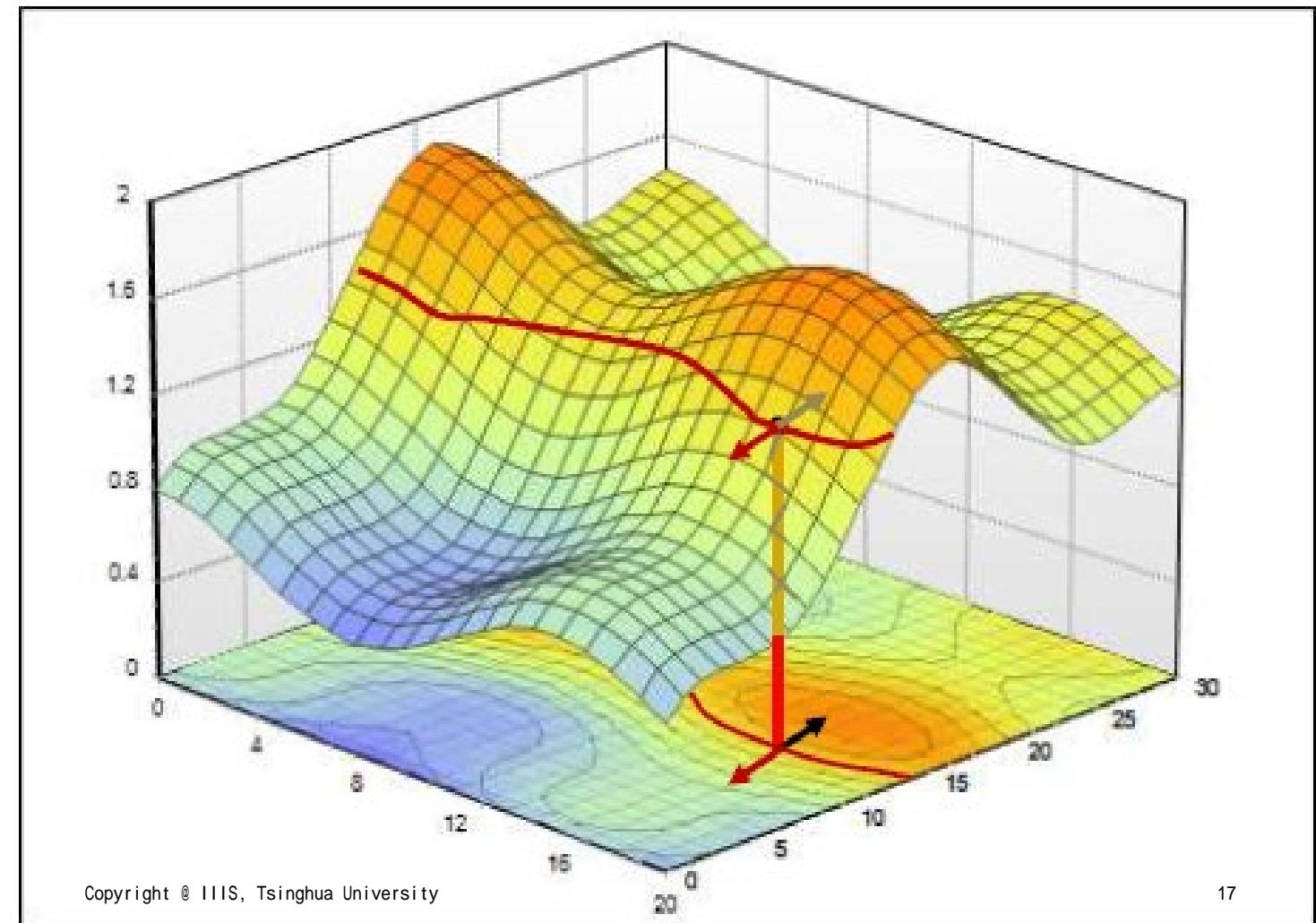


Multi-Variable Calculus Recap: Gradient



Multi-Variable Calculus Recap: Gradient

- The gradient $\nabla_X f(X)$ is perpendicular to the level curve



Multi-Variable Calculus Recap: Hessian

- The Hessian $\nabla^2 f(X)$ of a function $f(x_1, \dots, x_n)$ is given by the second derivative

- Positive-definite for a local minimum

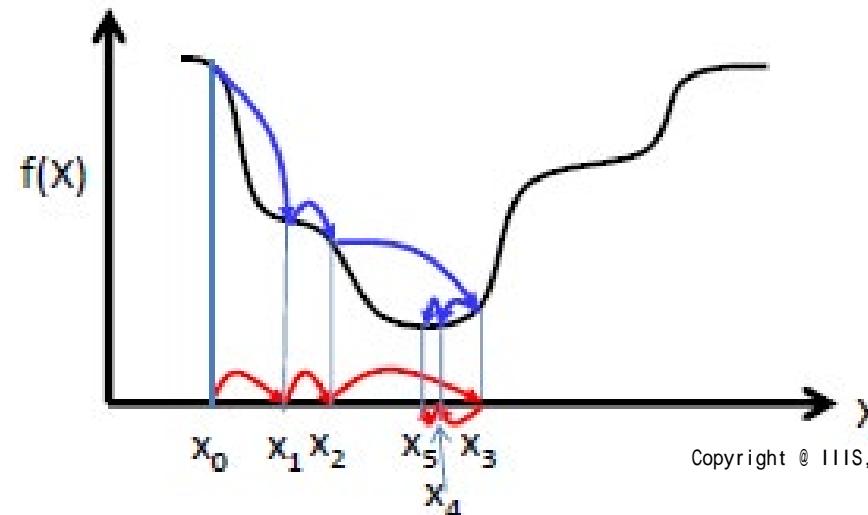
$$\nabla_x^2 f(x_1, \dots, x_n) := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Function Optimization (Cont'd)

- Problem: minimize $f(X)$ w.r.t. X
 - Unconstraint optimization
- Solution
 - Step 1: solve $\nabla f(X) = 0$
 - Step 2: calculate $\nabla^2 f(X)$ for candidates from (1) and verify positive-definiteness
- Issue: what if analytical solution is not feasible?

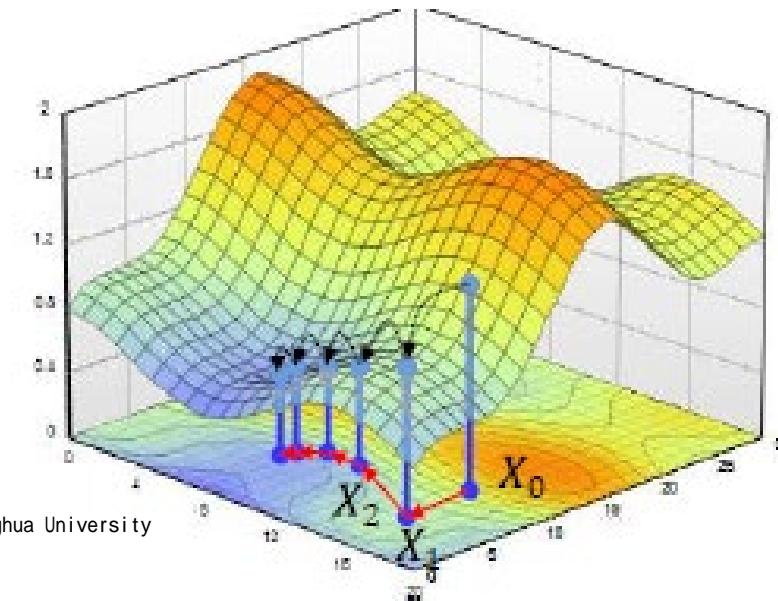
Function Optimization (Cont'd)

- Iterative solution
 - Start with a “guess” X^0 , and iteratively refine X until $\nabla_X f(X) = 0$ is reached
- A greedy solution
 - Refine X such that $f(X)$ is decreased!
 - Idea: follow the gradient direction!



Copyright © IIIS, Tsinghua University

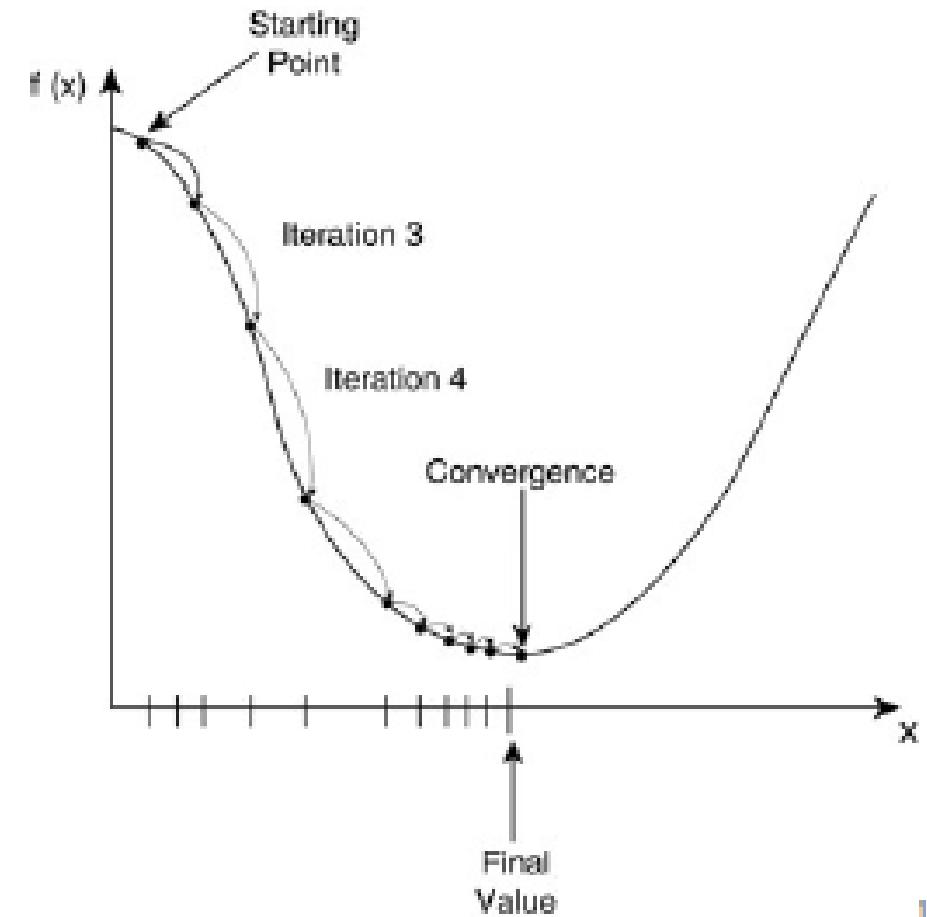
2/24



20

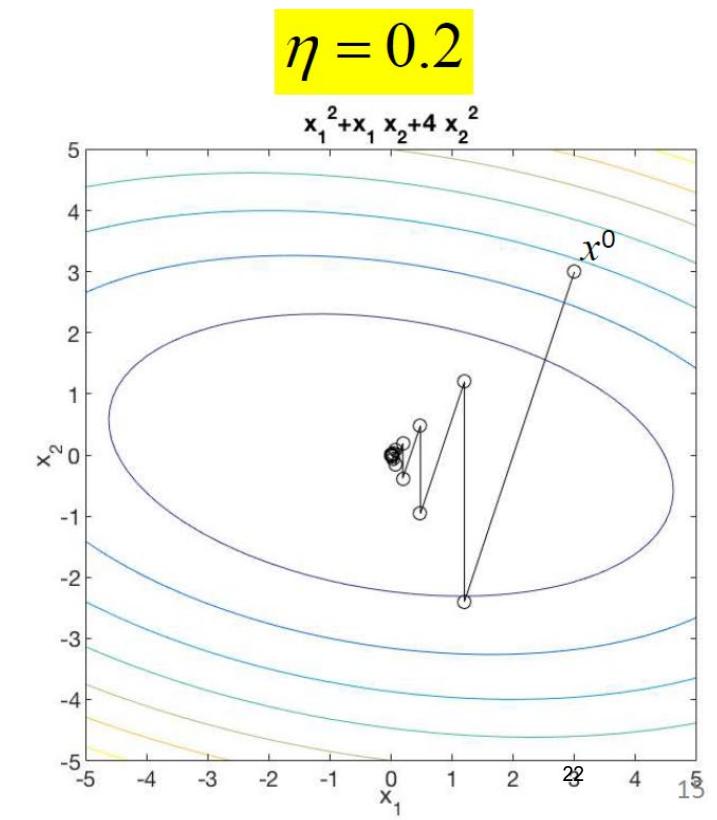
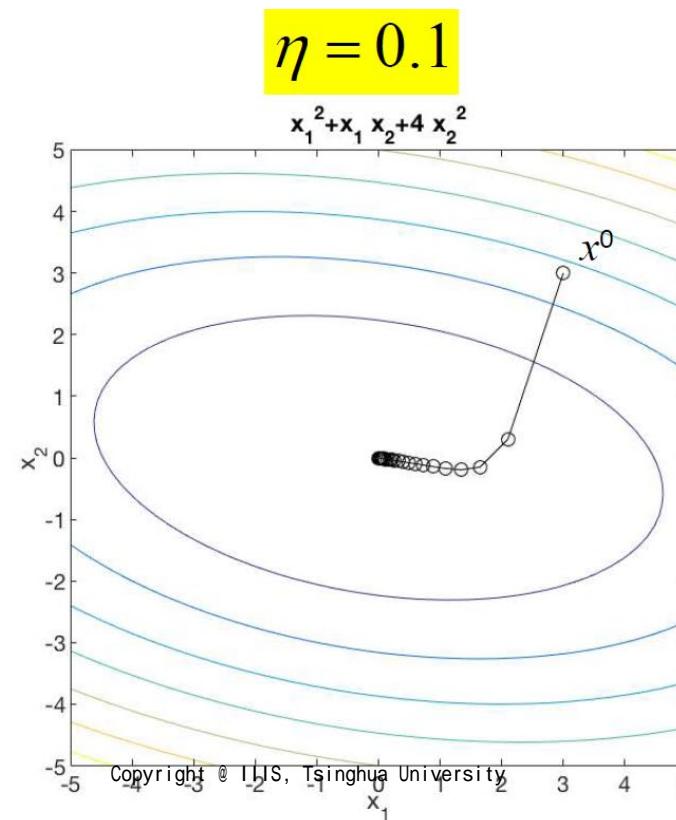
Function Optimization (Cont'd)

- The Gradient Descent algorithm
 - Choose X^0
 - $X^{k+1} = X^k - \eta^k \nabla_X f(X^k)^T$
 - Convergence: $|f(X^{k+1}) - f(X^k)| < \epsilon$
- Remark: GD may find a local optimum or a reflection point
- η^k learning rate
 - A critical parameter for gradient descent
 - More on next lecture



Function Optimization (Cont'd)

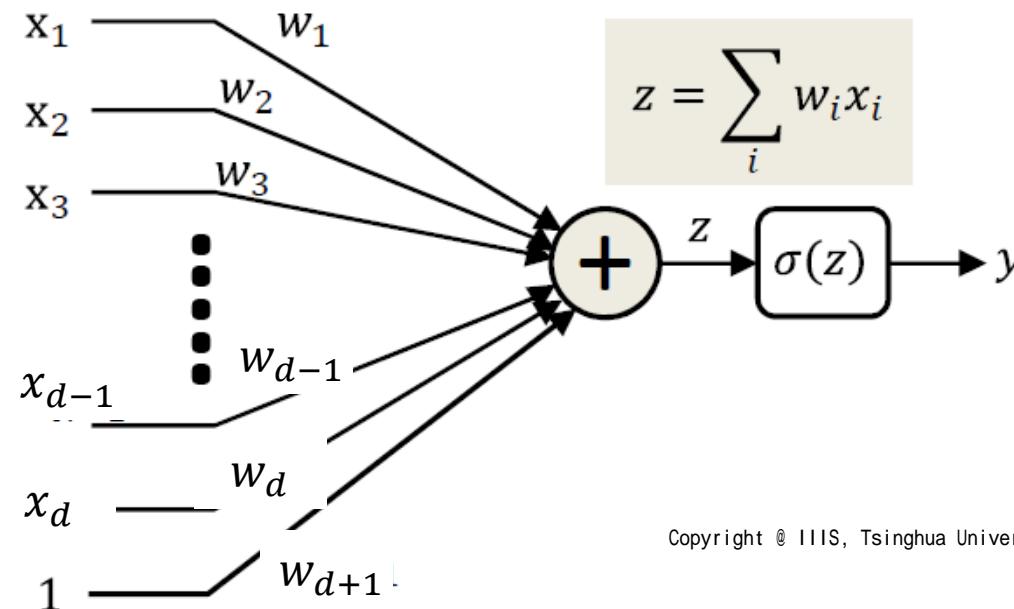
- Effectiveness of learning rate η^k
 - Example: $f(x_1, x_2) = x_1^2 + x_1 x_2 + 4x_2^2$, $X^0 = [3, 3]^T$
 - Remark
 - small η : safe but slow
 - Large η : may diverge



Learning the Single-Layer Perceptron

- Problem Statement

- Given $X = \{(x^i, y^i)\}$
- Loss function $L(w) = \frac{1}{N} \sum_i err(f(x^i; w), y^i)$
- Goal: minimize $L(w)$ w.r.t. w



$$z = \sum_i w_i x_i$$

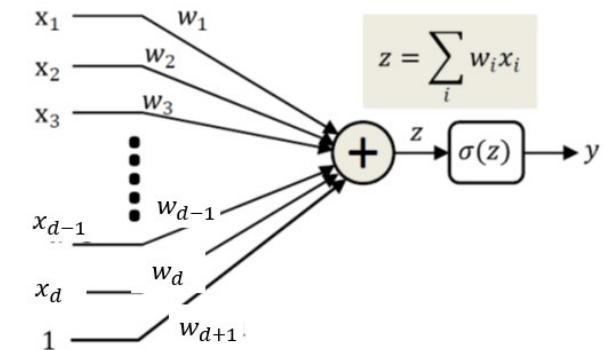
$$\frac{dy}{dz} = \sigma'(z)$$

$$\frac{dy}{dw_i} = \frac{dy}{dz} \frac{dz}{dw_i} = \sigma'(z) x_i$$

$$\frac{dy}{dx_i} = \frac{dy}{dz} \frac{dz}{dx_i} = \sigma'(z) w_i$$

Learning the Single-Layer Perceptron

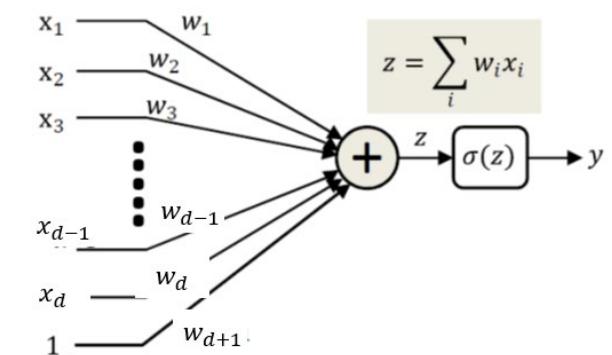
- Gradient Descent
 - Initialize w^0 ; $w^{k+1} = w^k - \eta^k \nabla_w L(w)$ until convergence
- Compute the Gradient
 - $\nabla_w L(w) = \frac{1}{N} \sum_i \nabla_w err(f(x^i; w), y^i)$
 - $y = 1$: $\nabla_w err = -\nabla_w \log \sigma(w^T x^i) = -\frac{1}{err} \nabla_w \sigma(w^T x^i)$
 - $y = 0$: $\nabla_w err = -\nabla_w \log(1 - \sigma(w^T x + b)) = \frac{1}{err} \nabla_w \sigma(w^T x^i)$
- Gradient of Sigmoid neuron
 - $\sigma(z) = \frac{1}{1+e^{-z}}$ where $z = w^T x$
 - $\nabla_{w_i} \sigma(z) = \sigma'(z)x_i$ and $\nabla_{x_i} \sigma(z) = \sigma'(z)w_i$
 - $\sigma'(z) = \sigma(z)(1 - \sigma(z))$



Multi-Class Classification

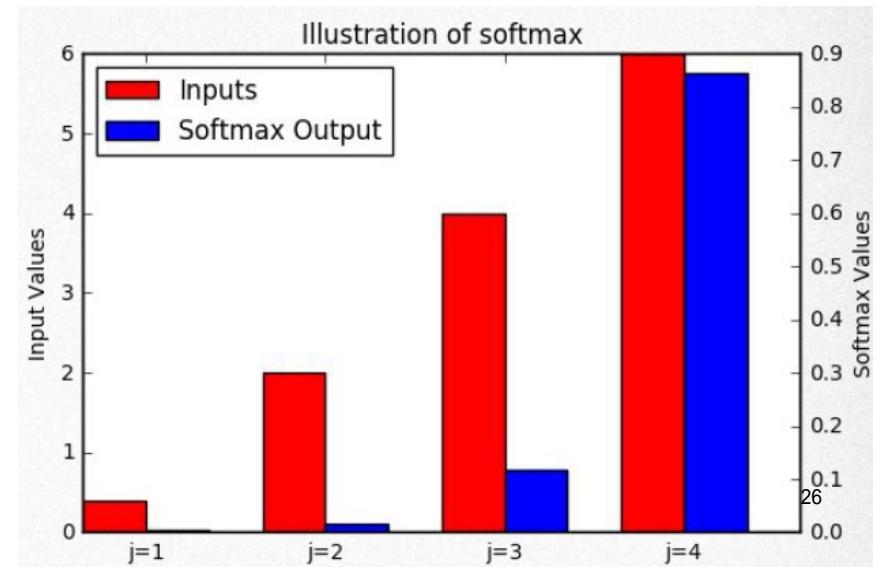
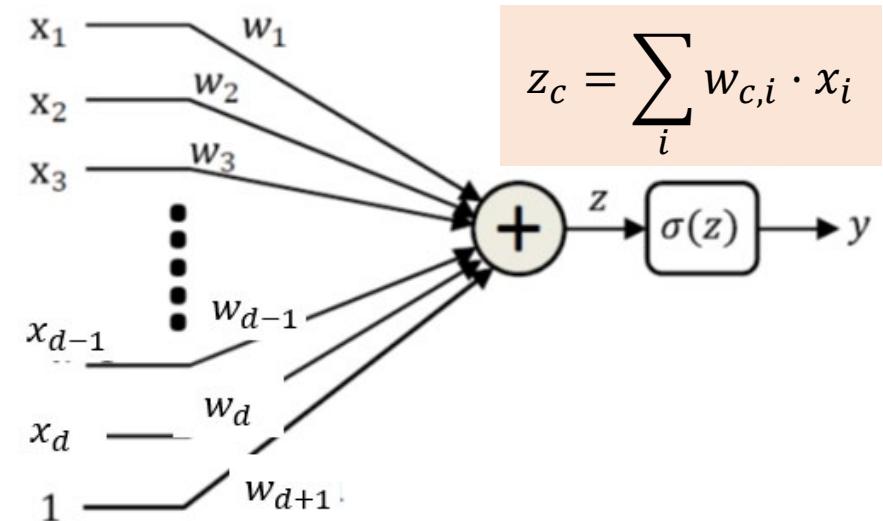
- Example: digit classification
 - 10 classes
 - Sigmoid output → only valued in (0,1) (binary classes)
- Multi-Class Classification Formulation
 - C classes of labels (10 in digit recognition)
 - $f(x; \theta)$: a probability distribution over C classes
 - $P(y = c|x) = f_c(x; \theta)$
 - $f_c \geq 0$ and $\sum_c f_c = 1$
- We need a modified $\sigma(z)$ such that $\sigma(z)$ becomes a ***multi-class*** probability distribution

| Training data | |
|---------------|--------|
| (5, 0) | (2, 1) |
| (2, 1) | (4, 0) |
| (0, 0) | (2, 1) |



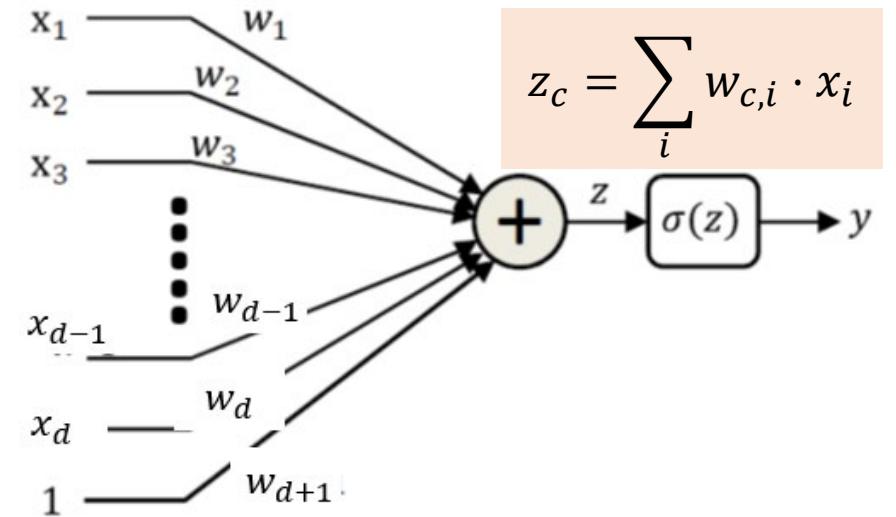
Multi-Class Classification

- Softmax Function
 - C classes, $w \in \mathbb{R}^{(d+1) \times C}$
 - $z = Wx$, z is C dimensional
 - $f(x; w) = \text{softmax}(z)$
 - $f_c(x; w) = \frac{e^{z_c}}{\sum_{j=1}^C e^{z_j}} \propto \exp(z_c)$
- Interpretation
 - A universal operator to convert a vector to ***a probability distribution***
 - z_i is often called *logit* (*refer to an unscaled value*)
 - A “soft” version of max operator



Multi-Class Classification

- A multi-class perceptron
 - C classes, $z = Wx$,
 - $f(x; W) = \text{softmax}(z)$, $f_c(x; W) = \frac{e^{z_c}}{\sum_{j=1}^C e^{z_j}}$
- Learning
 - Given $X = \{(x^i, y^i)\}$, Loss function $L(W) = \frac{1}{N} \sum_i \text{err}(f(x^i; W); y^i)$
- The error function
 - The probability of a class c given input x^i is $P(y = c | x^i) = f_c(x^i; W)$
 - Maximize the log probability of desired class y^i
 - $\text{err}(f(x^i; W), y^i) = -\log f_{y^i}(x^i; W)$



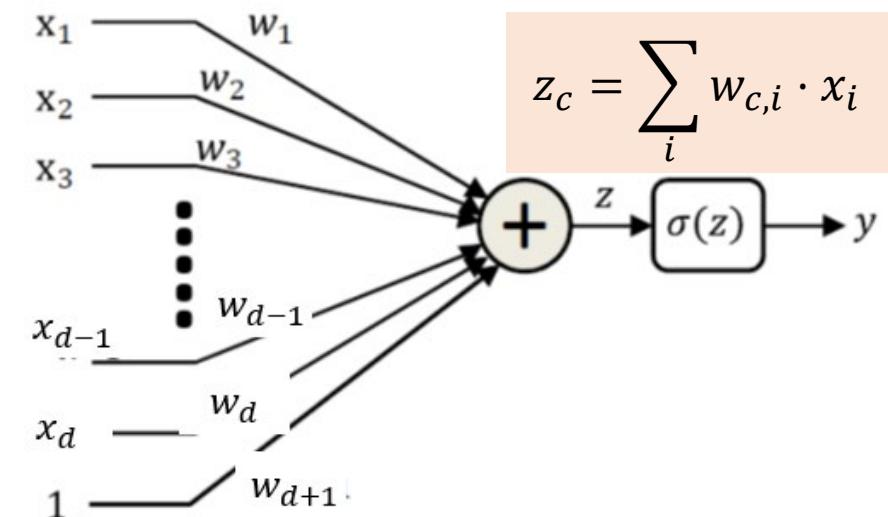
Multi-Class Classification

- The NLL loss (negative log-likelihood loss)

- $err(f(x^i; w), y^i) = -\log f_{y^i}(x^i; w)$

- $z = Wx^i, f_c(x^i; W) = \frac{e^{z_c}}{\sum_{j=1}^C e^{z_j}}$

- $L(W) = -\frac{1}{N} \sum_i \log f_{y^i}(x^i; W) = -\frac{1}{N} \sum_i \left(z_{y^i} - \log \sum_j e^{z_j} \right)$



- NLL loss is also called *cross-entropy loss*

- It is equivalent to the cross entropy (交叉熵) between $f(x; W)$ and the delta probability $[0, 0, \dots, 0, 1, 0, \dots]$ for class k

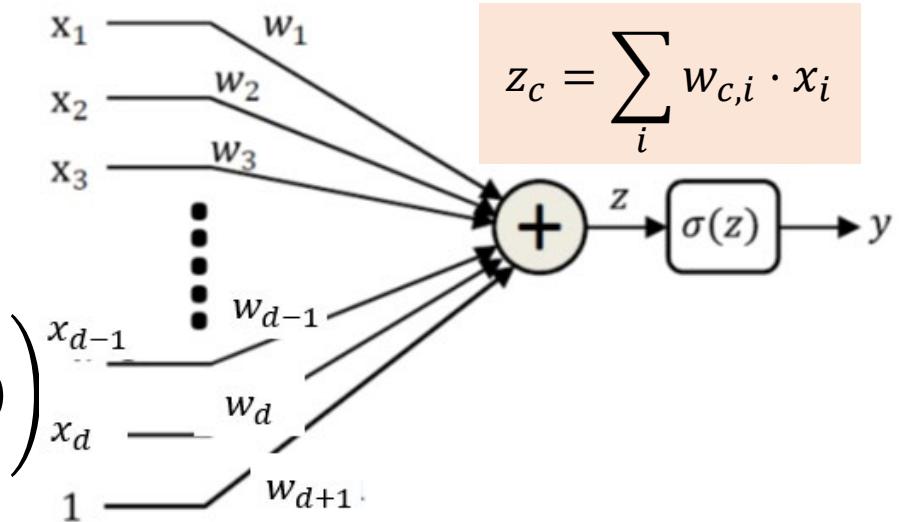
- $CE(p, q) = -\sum_c p(y=c) \log q(y=c)$

- $p(y=k) = 1$ and $q = f(x; W)$

- We also call the vector probability $[0, 0, \dots, 0, 1, 0, \dots, 0]$ a **one-hot vector**

Multi-Class Classification

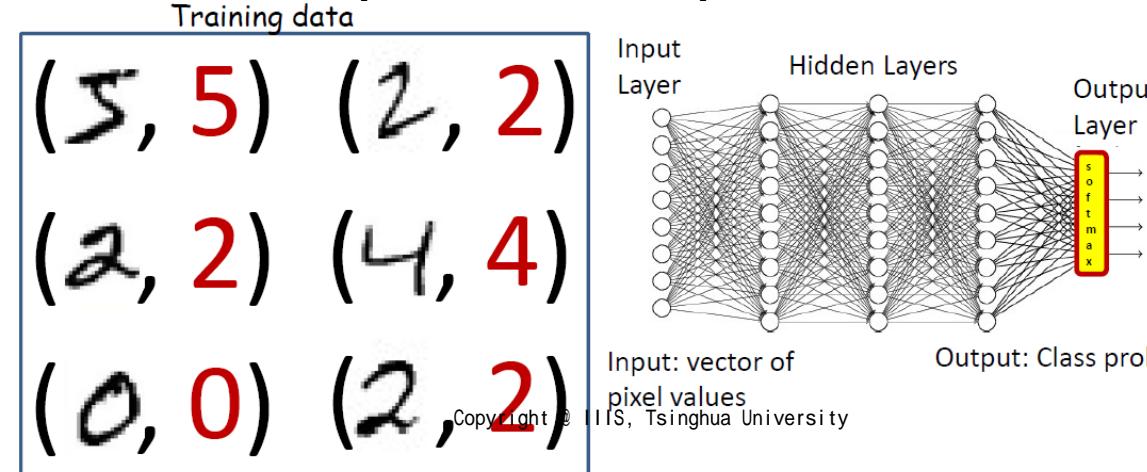
- Gradient Computation
 - Note: $z = Wx^i$ and W is a matrix!
 - $\nabla_{w_{c,j}} L(W) = -\frac{1}{N} \sum_i \nabla_{w_{c,j}} \log f_{y^i}(x^i; W)$
 - $\nabla_{w_{c,j}} f_c(x^i; W) = \left(\nabla_{w_{c,j}} z_c - \frac{1}{\sum_k e^{z_k}} \nabla_{w_{c,j}} (\sum_k e^{z_k}) \right)$
 - $\nabla_{w_{c,j}} z_c = \nabla_{w_{c,j}} (\sum_{l=1}^{d+1} w_{c,l} x_l^i) = x_j^i$



- You have to do this in your coding project 😊

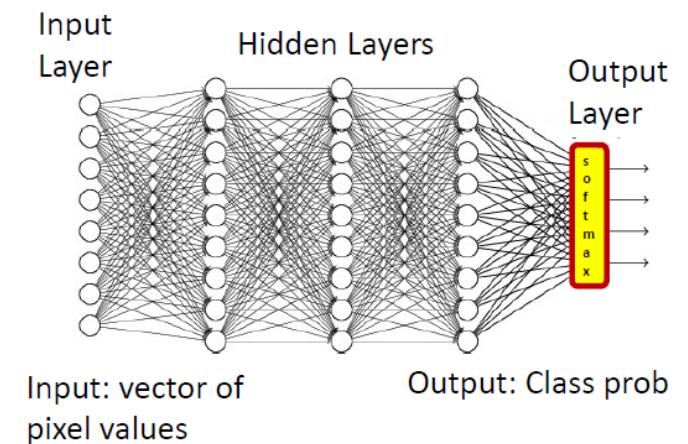
Multi-Layer Perceptron

- A N -layered MLP with Sigmoid activation function
 - Input layer: $y^{(0)} = x$
 - Hidden layer: $y^{(k)} = f_k(z^{(k)}) = f_k(W^{(k)}y^{(k-1)} + b^{(k)})$
 - Activation Function $f_k(z)$ (e.g., sigmoid)
 - Output Layer: $y = \text{softmax}(y^{(N)})$
- Learning all the weights $\{W^{(k)}, b^{(k)}\}$ (also called weight and bias)



Learning MLPs

- Learning weights by gradient descent
 - Initialize all $\{W^{(k)}\}$ (assume $b^{(k)}$ is included in $W^{(k)}$)
 - For every k, i, j
 - Update $w_{i,j}^{(k)} \leftarrow w_{i,j}^{(k)} - \eta \frac{dL(W)}{dw_{i,j}^{(k)}}$ until convergence
- We need to compute gradient for ***every weight!***
- How to efficiently compute all these gradients?



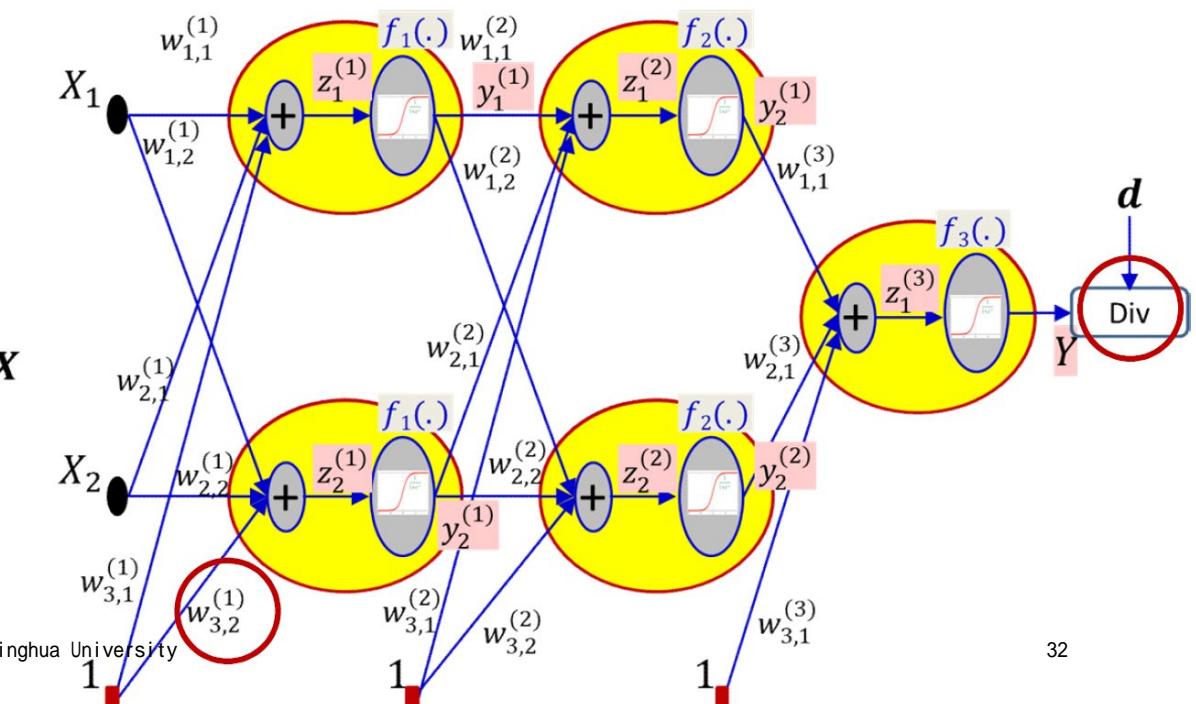
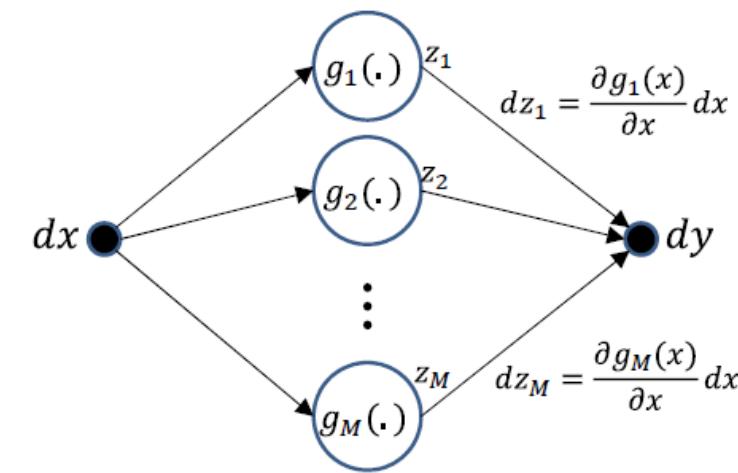
The Chain Rule

- Recap: Chain Rule for nested functions

- $y = f(g_1(x), g_2(x), \dots, g_M(x))$
- $\frac{dy}{dx} = \frac{\partial f}{\partial g_1(x)} \cdot \frac{dg_1(x)}{dx} + \dots + \frac{\partial f}{\partial g_M(x)} \cdot \frac{dg_M(x)}{dx}$

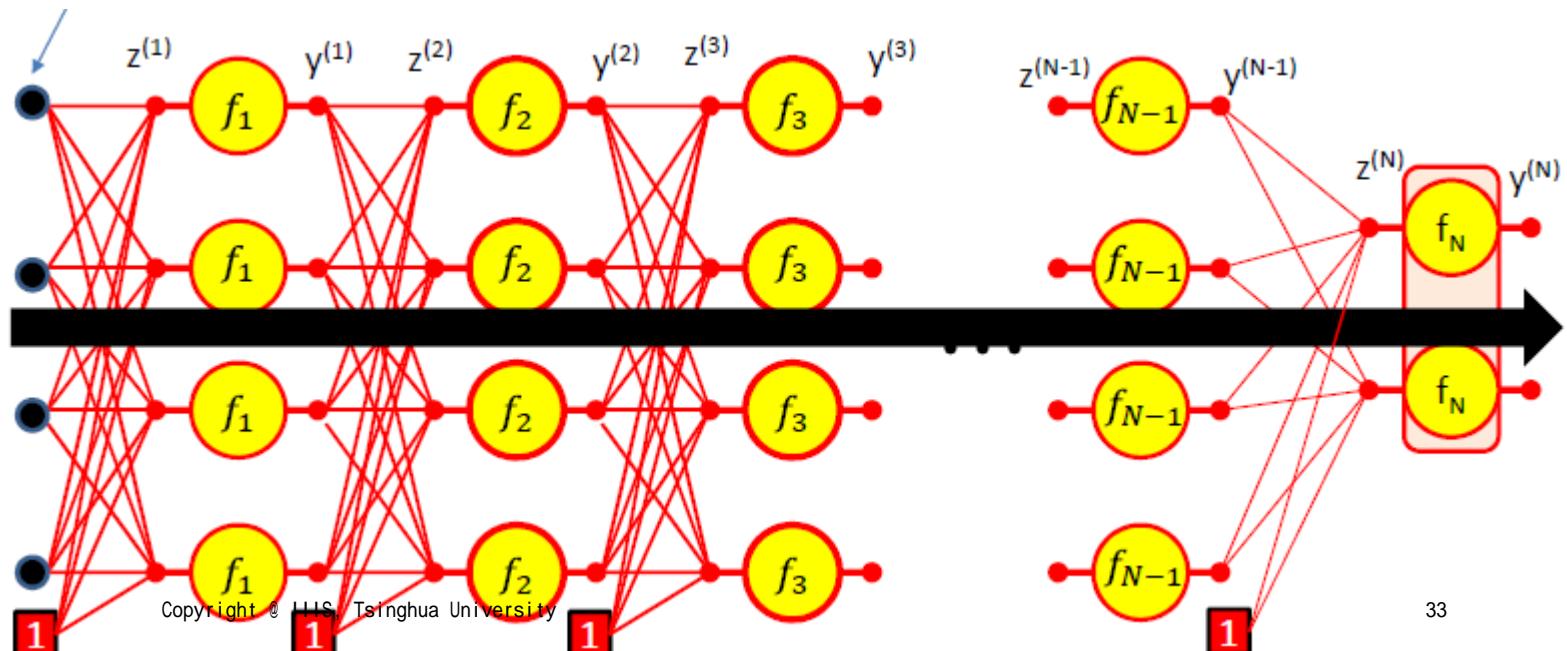
- A more complex example

- How to compute $\frac{d\text{Div}(Y,d)}{dw_{3,2}^{(1)}}$?
- We need all intermediate values!



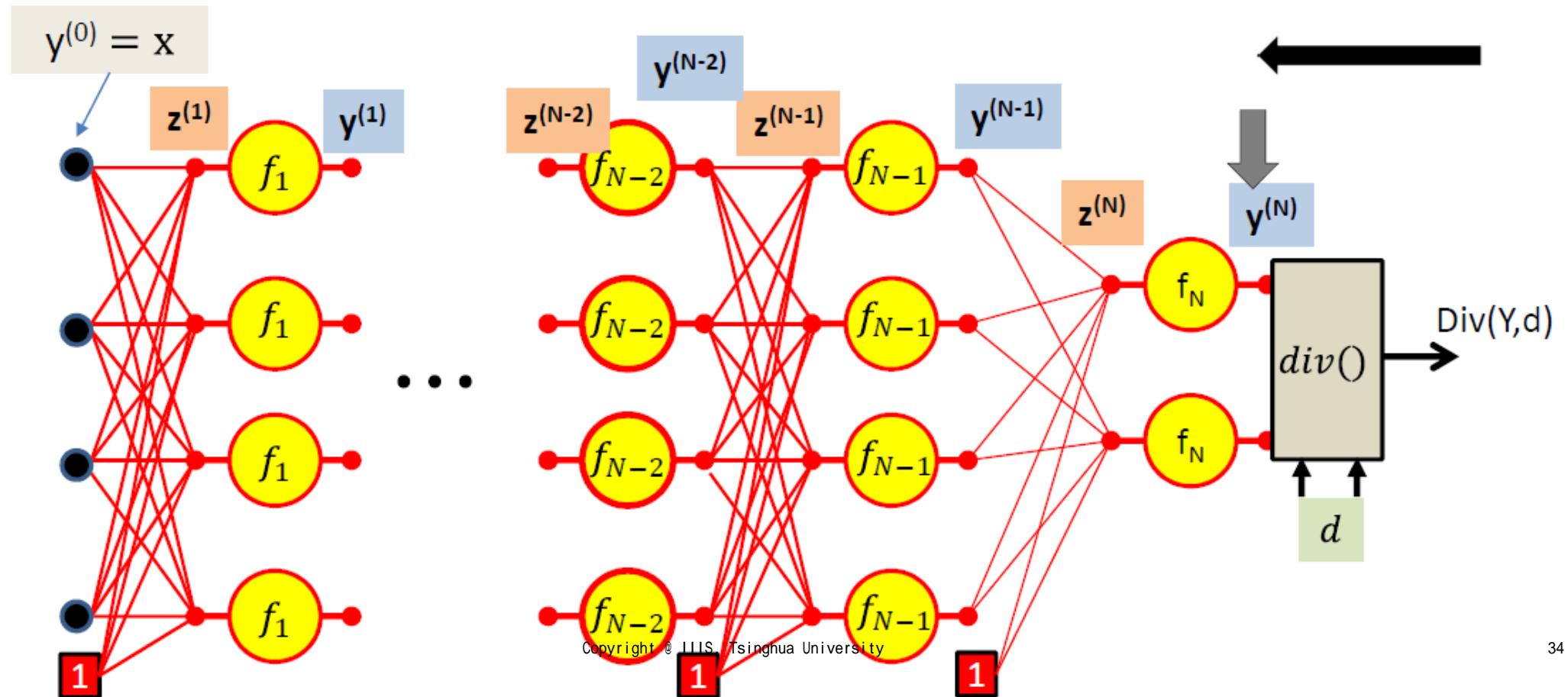
Computing Gradients

- The Forward Pass
 - Compute and save all the intermediate values
 - Set $y_{1:d_k}^{(0)} \leftarrow x$; and $y_{d_k+1}^{(k)} \leftarrow 1$
 - For layer $k \leftarrow 1 \dots N$
 - $z^{(k)} = W^{(k)}y^{(k-1)}$
 - $y^{(k)} = f_k(z^{(k)})$
 - Output $Y = y^{(K)}$
- Then Gradient?

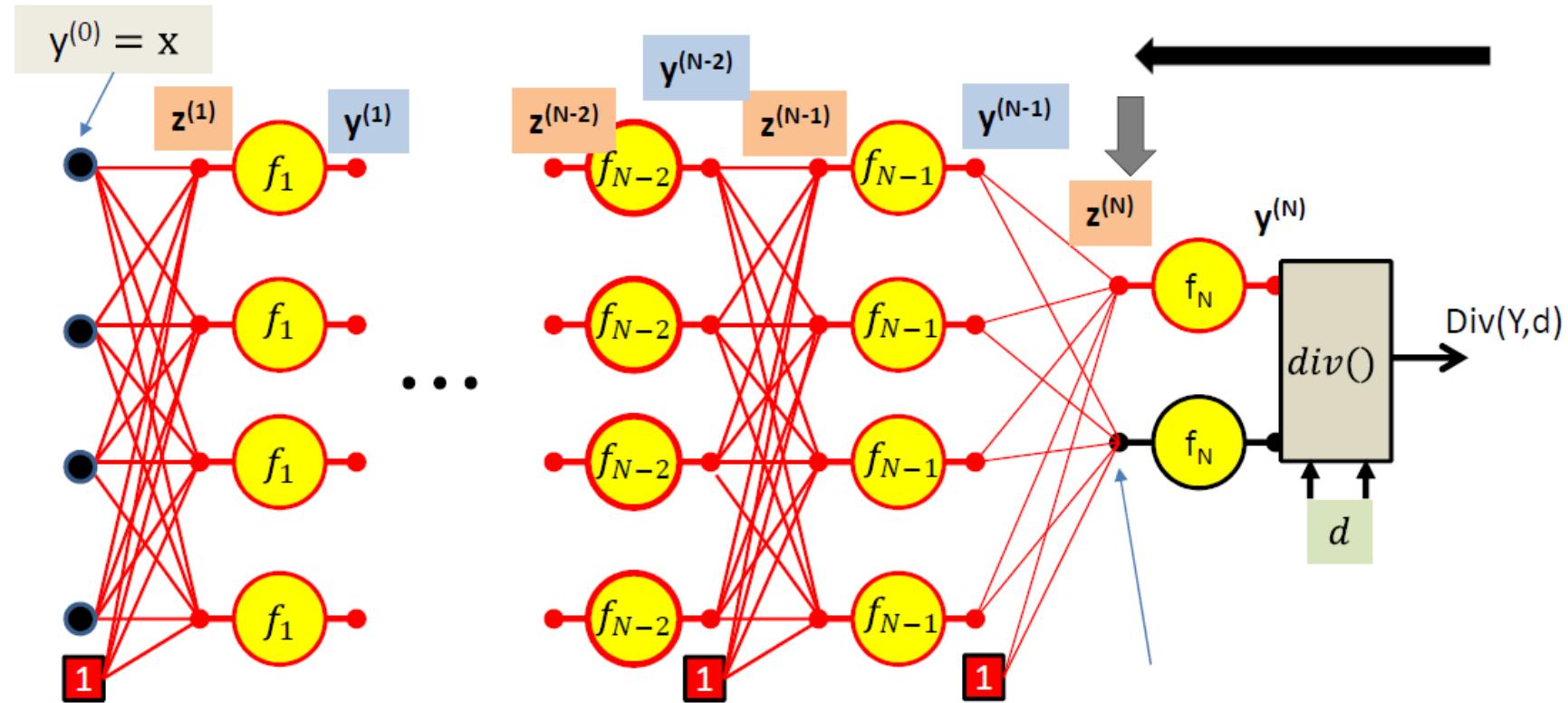


Computing Gradients

- First compute $\nabla_{y^{(N)}} \text{Div}(\cdot)$ directly



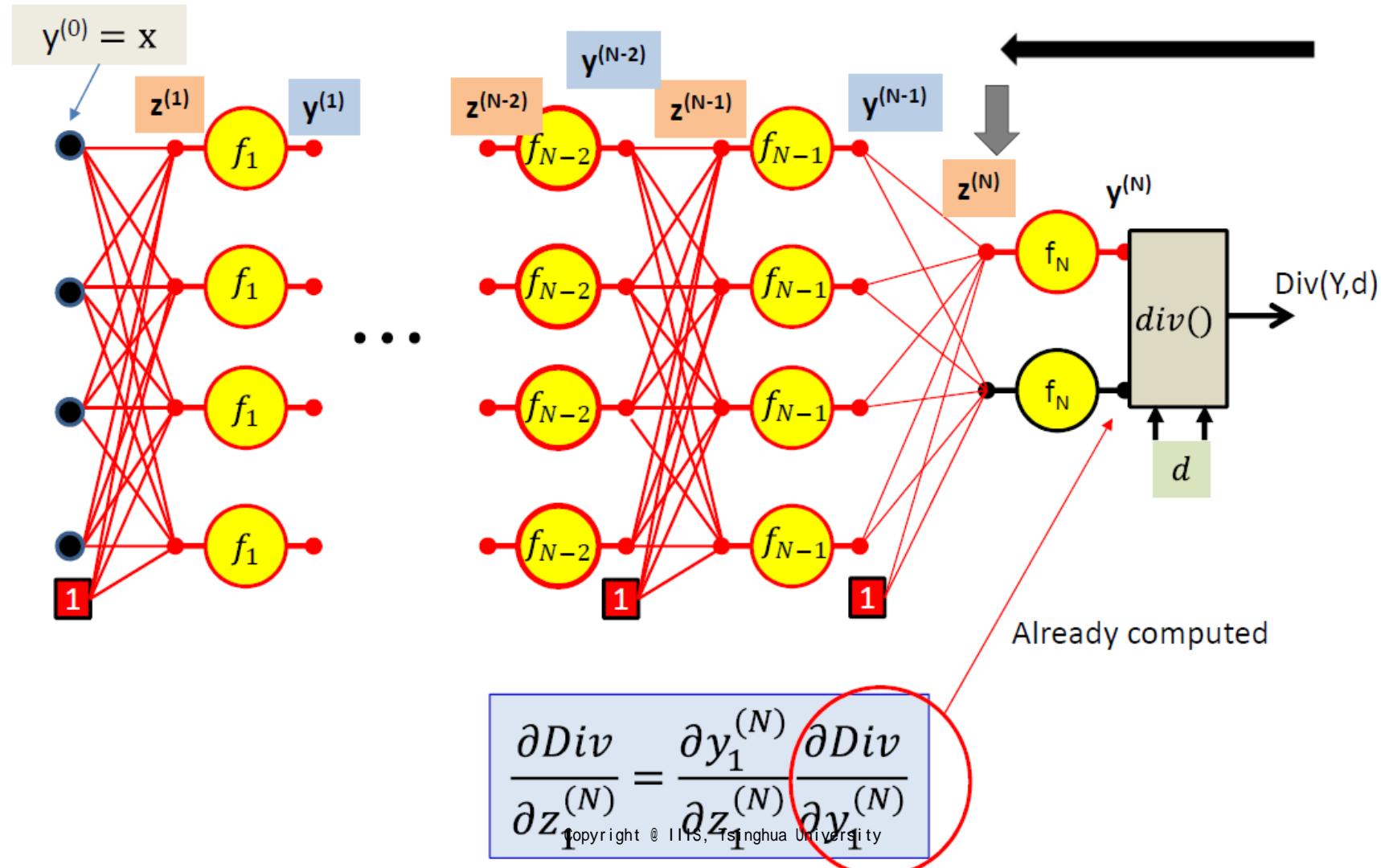
Computing Gradients



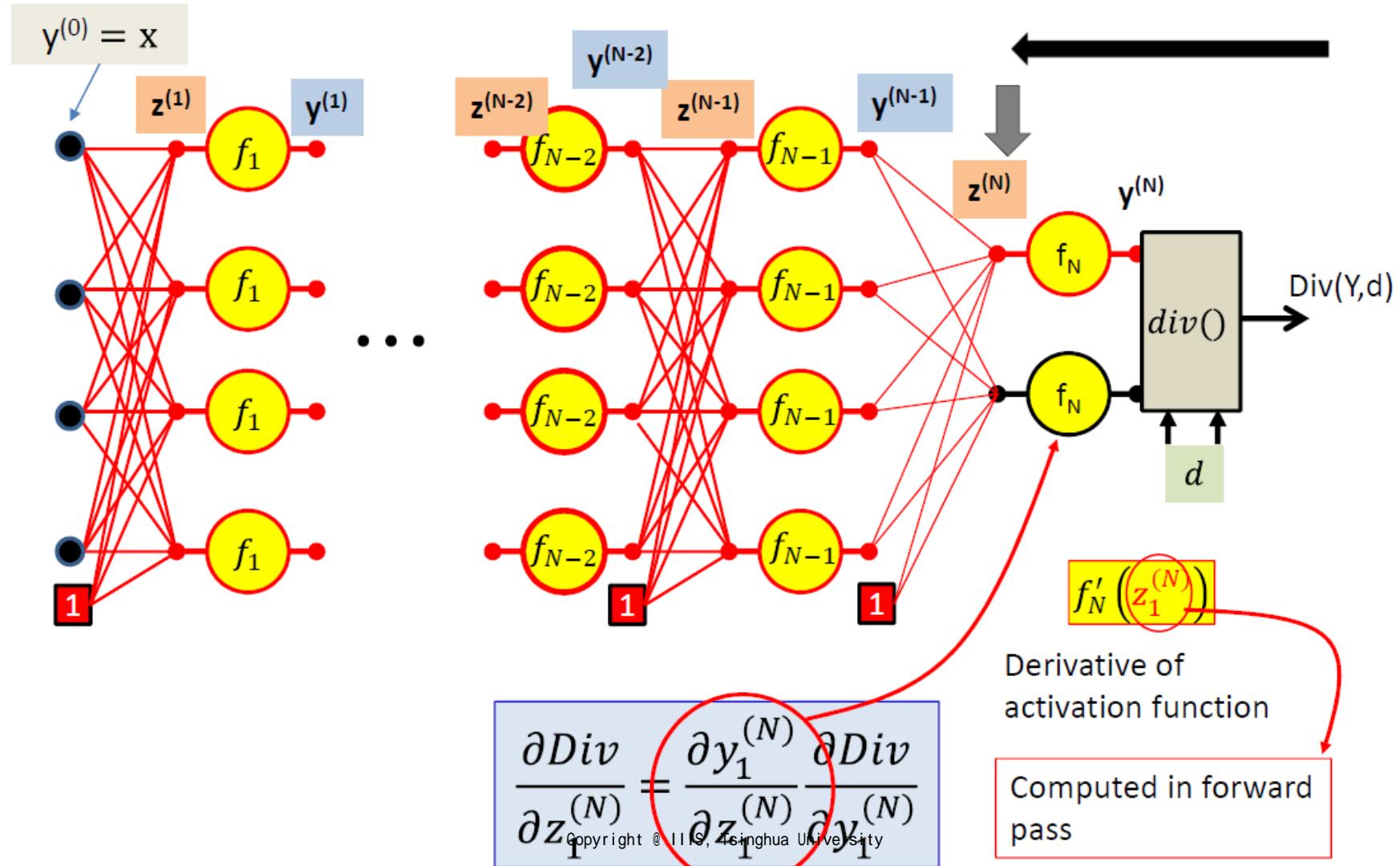
$$\frac{\partial Div}{\partial z_1^{(N)}} = \frac{\partial y_1^{(N)}}{\partial z_1^{(N)}} \frac{\partial Div}{\partial y_1^{(N)}}$$

Copyright © IIIS, Tsinghua University

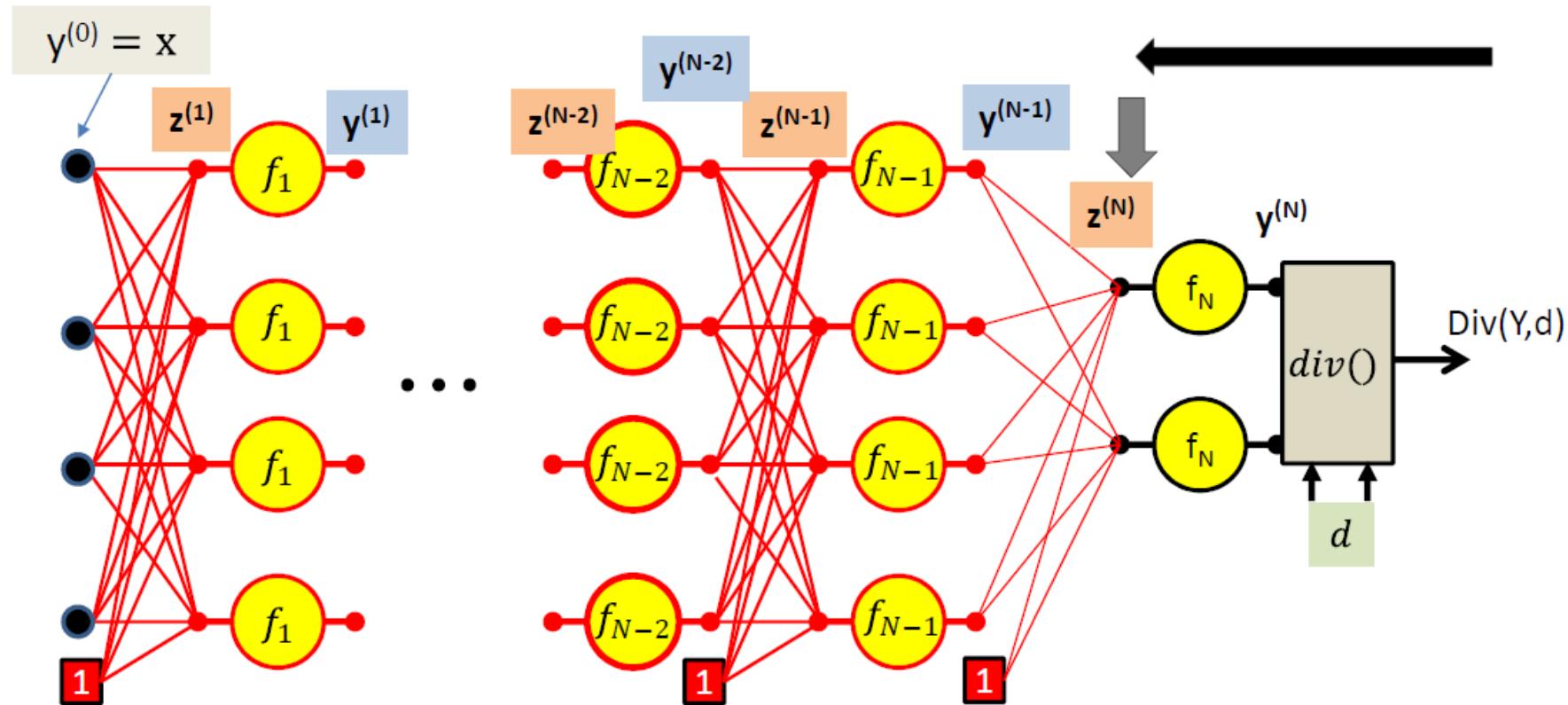
Computing Gradients



Computing Gradients

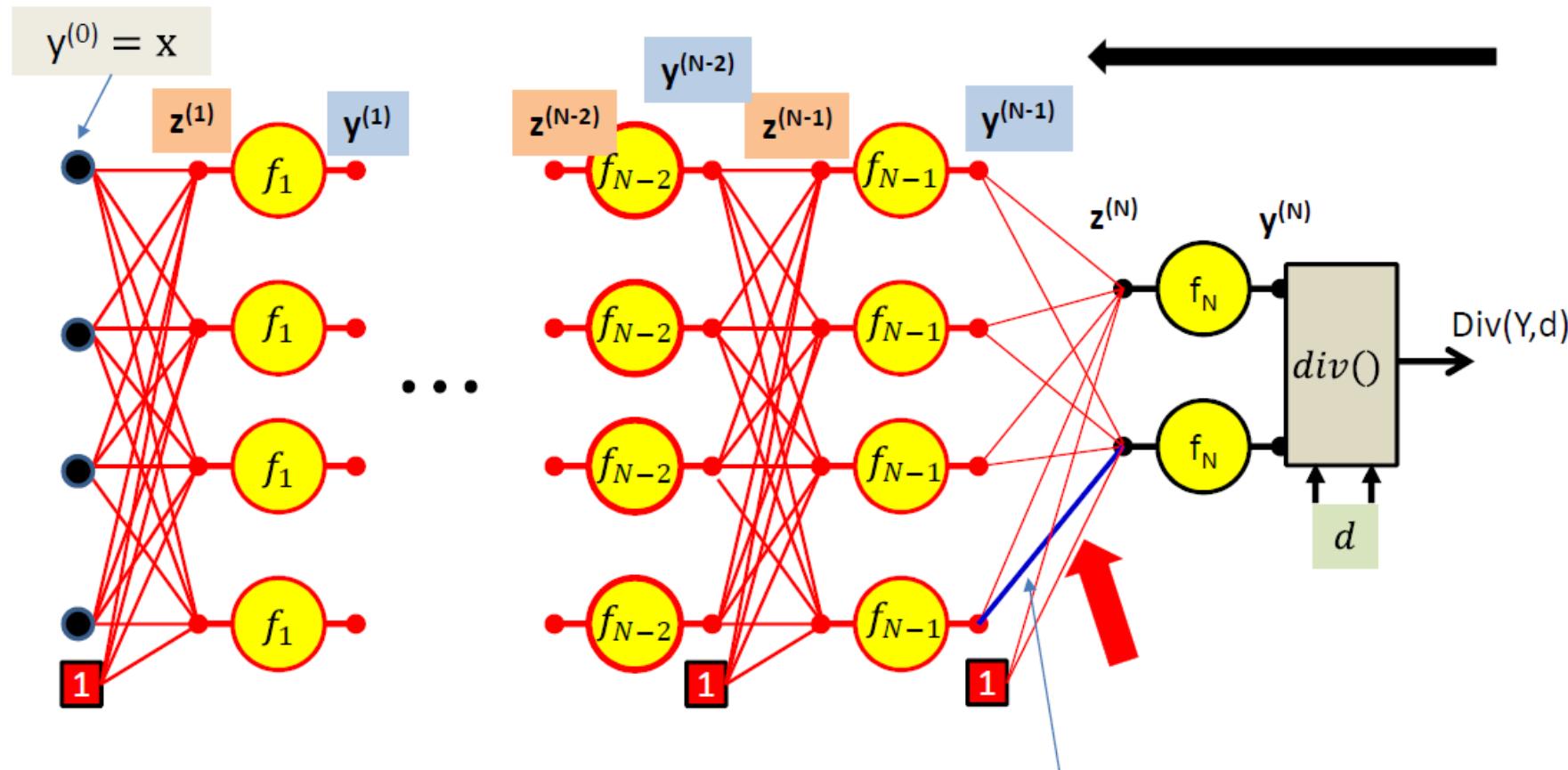


Computing Gradients



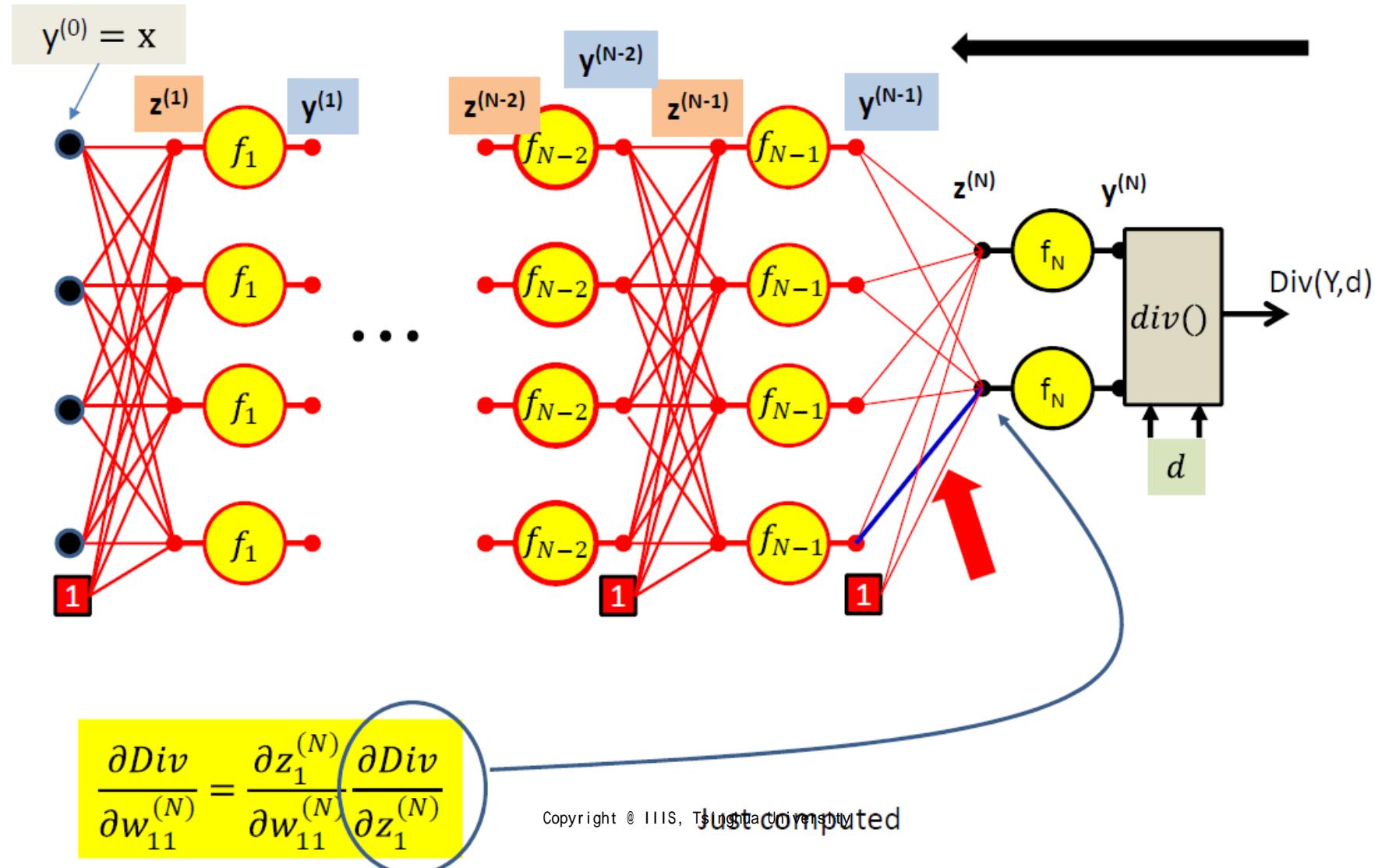
$$\frac{\partial Div}{\partial z_i^{(N)}} = f'_N(z_i^{(N)}) \frac{\partial Div}{\partial y_i^{(N)}}$$

Computing Gradients

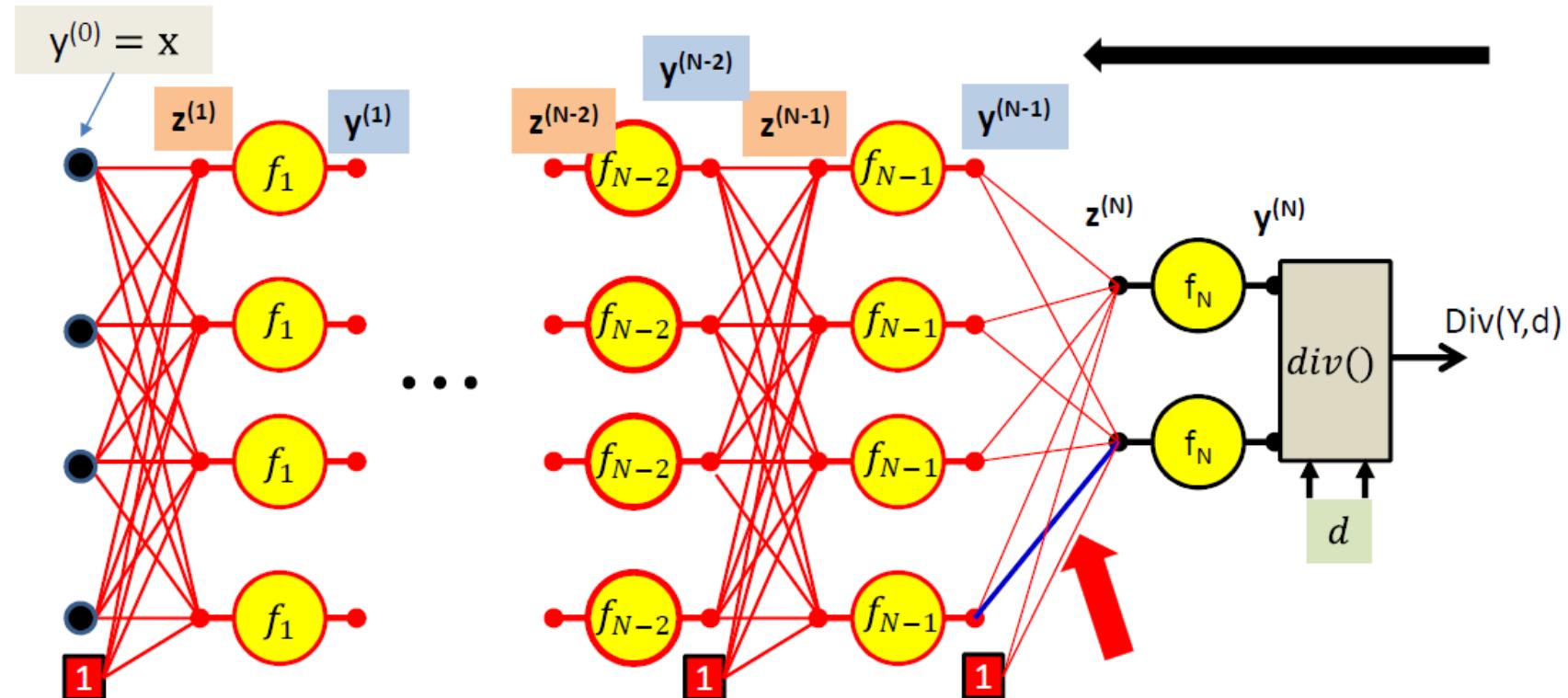


$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

Computing Gradients



Computing Gradients

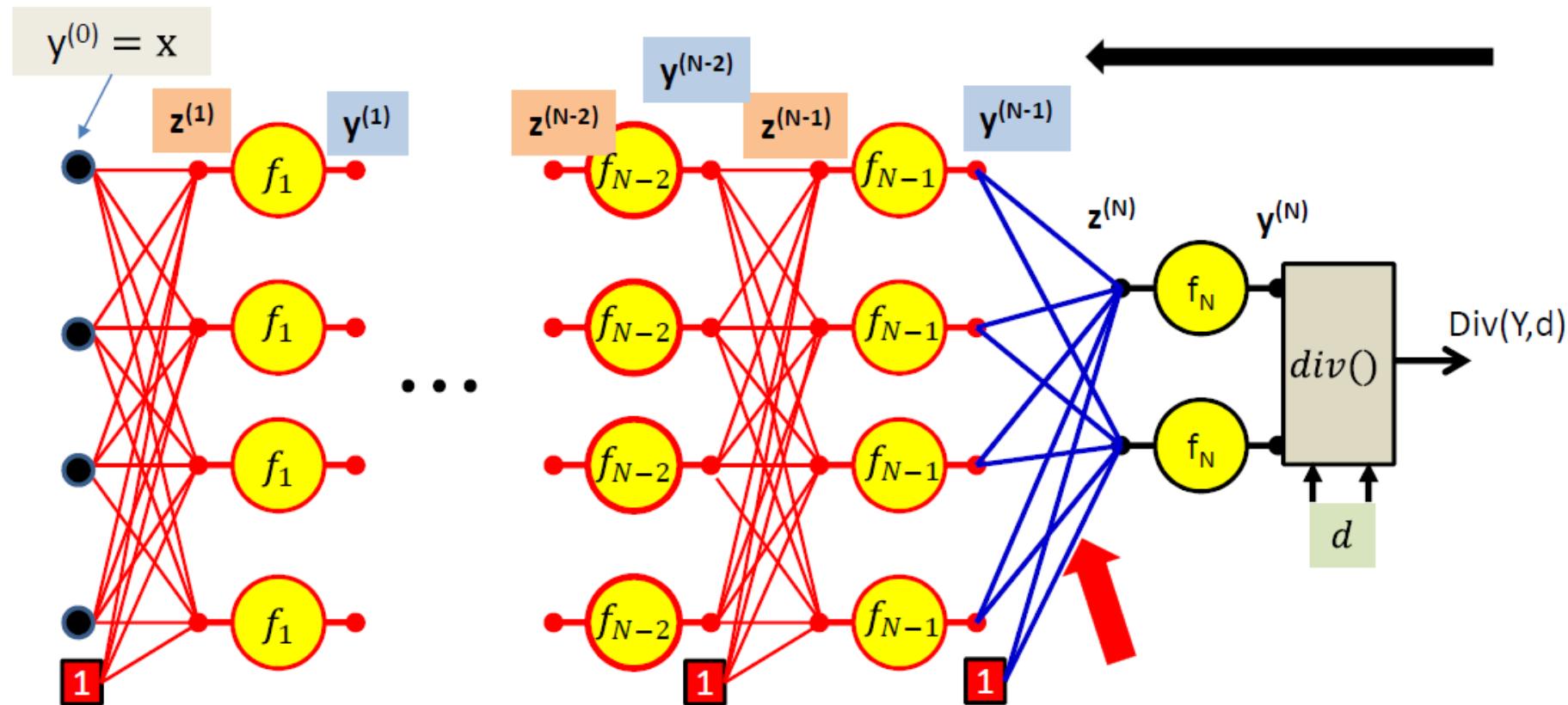


$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

Because
 $z_1^{(N)} = w_{11}^{(N)} y_1^{(N-1)} + \text{other terms}$

Copyright © IIIS, Tsinghua University
 Computed in forward pass

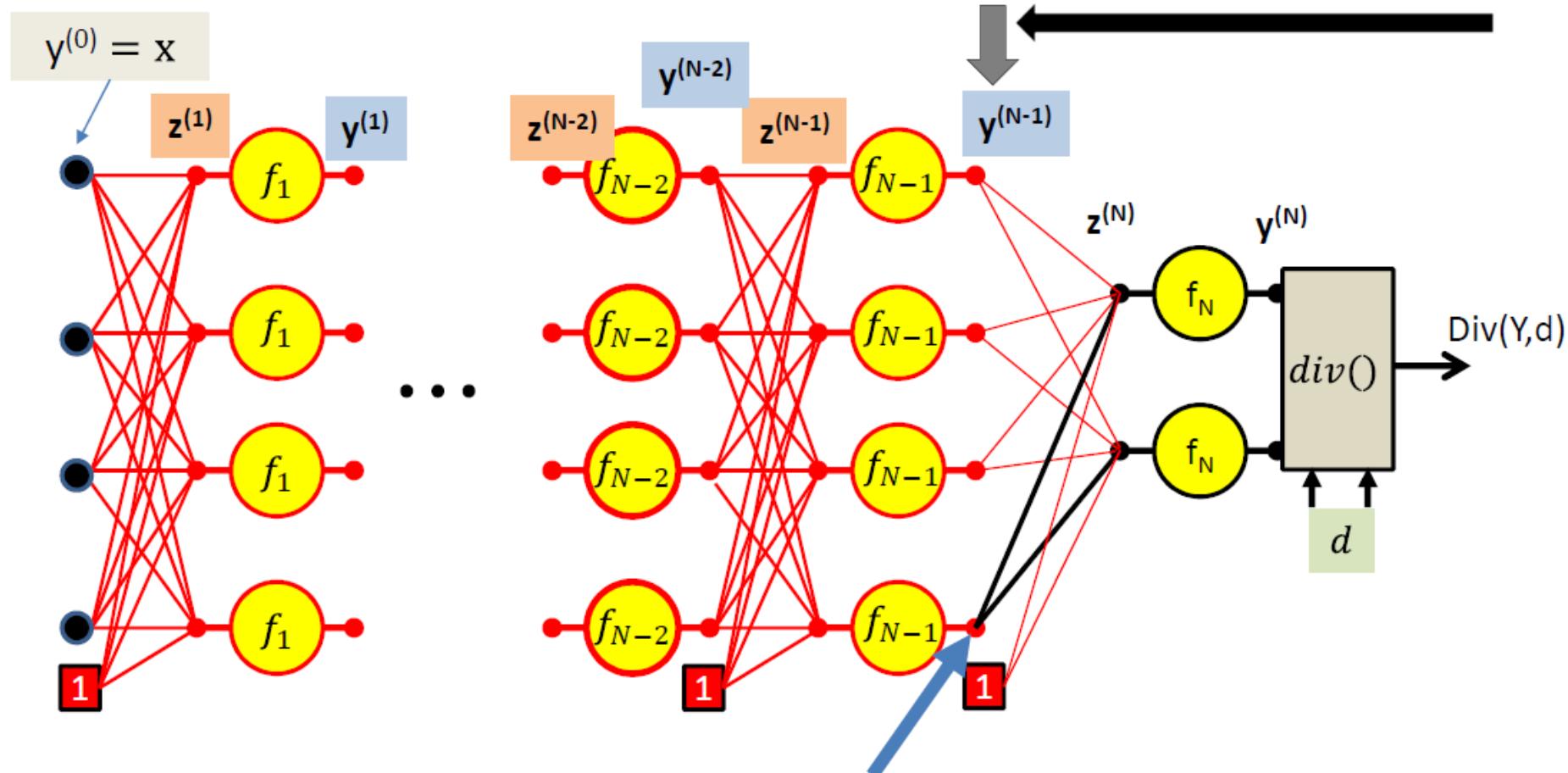
Computing Gradients



$$\frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \text{Div}}{\partial z_j^{(N)}}$$

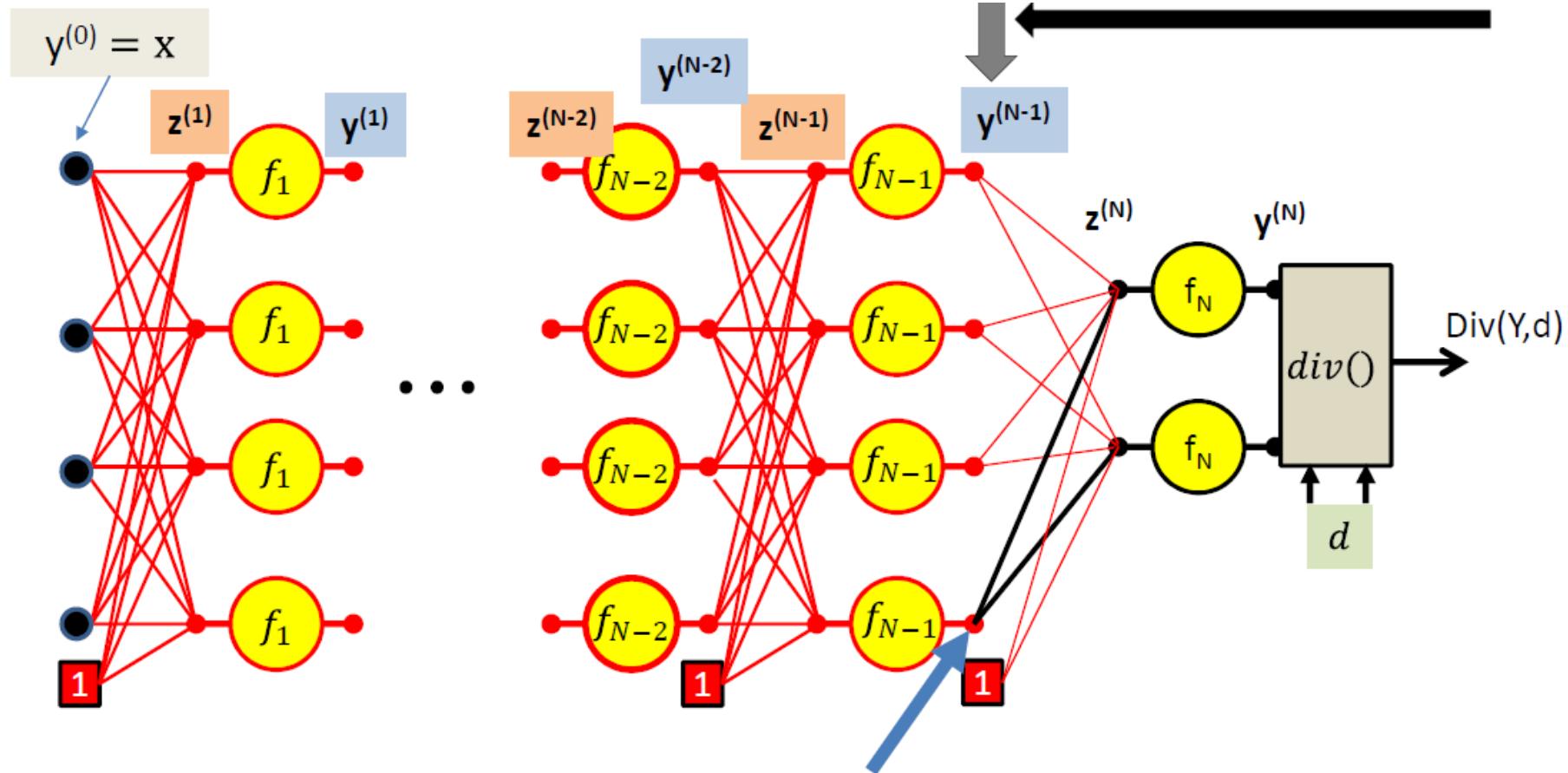
For the bias term $y_0^{(N-1)} = 1$

Computing Gradients



$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

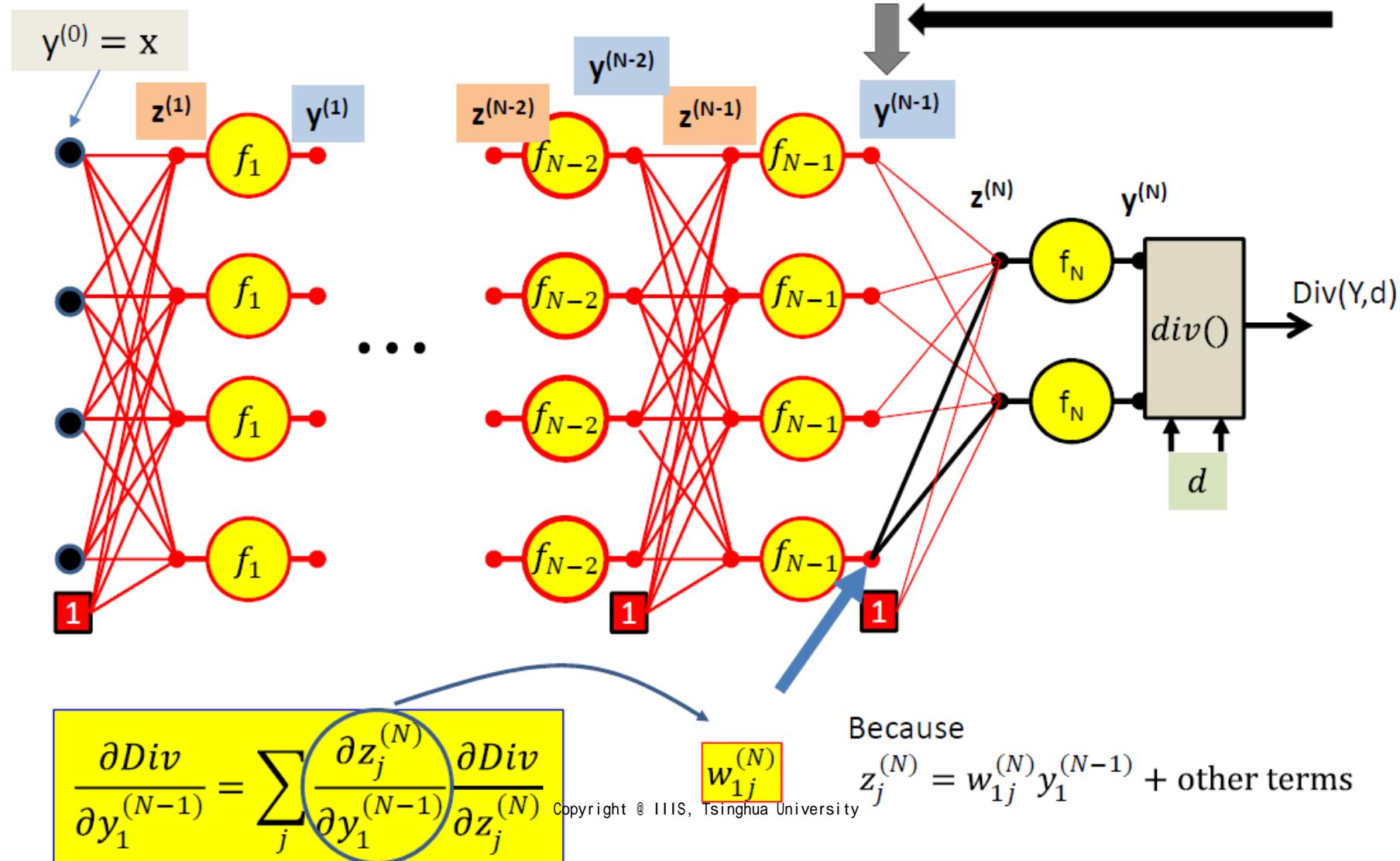
Computing Gradients



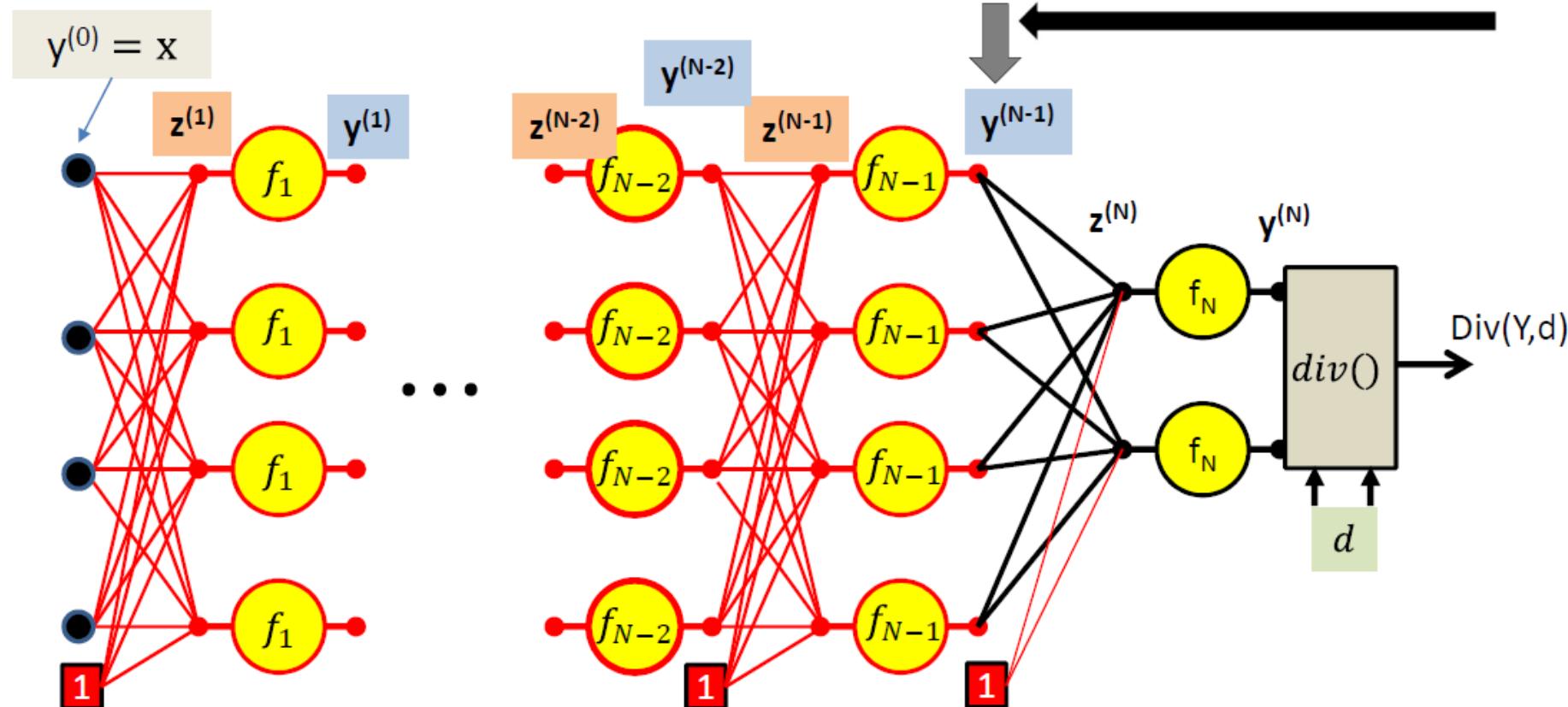
$$\frac{\partial \text{Div}}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial \text{Div}}{\partial z_j^{(N)}}$$

Copyright © IIIS, Tsinghua University

Computing Gradients

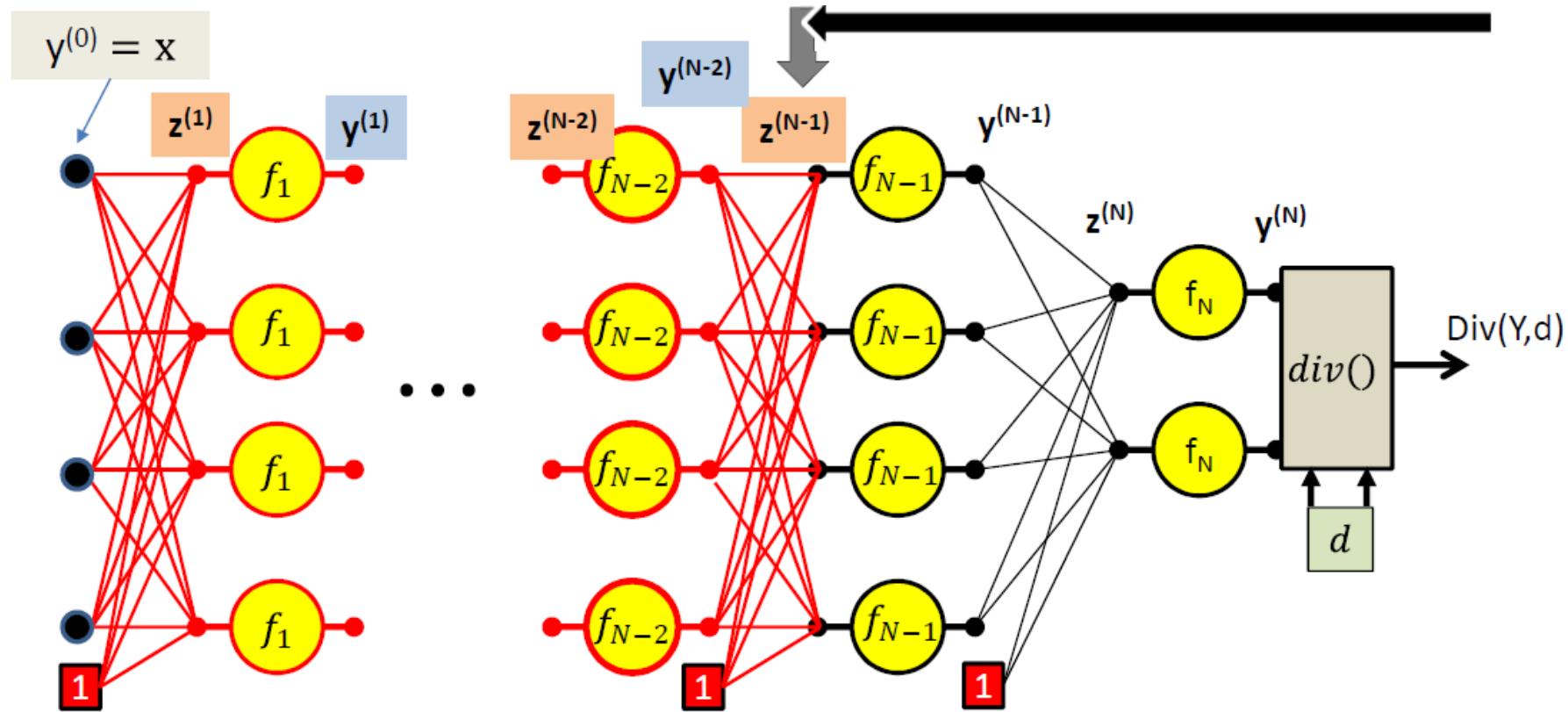


Computing Gradients



$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing Gradients

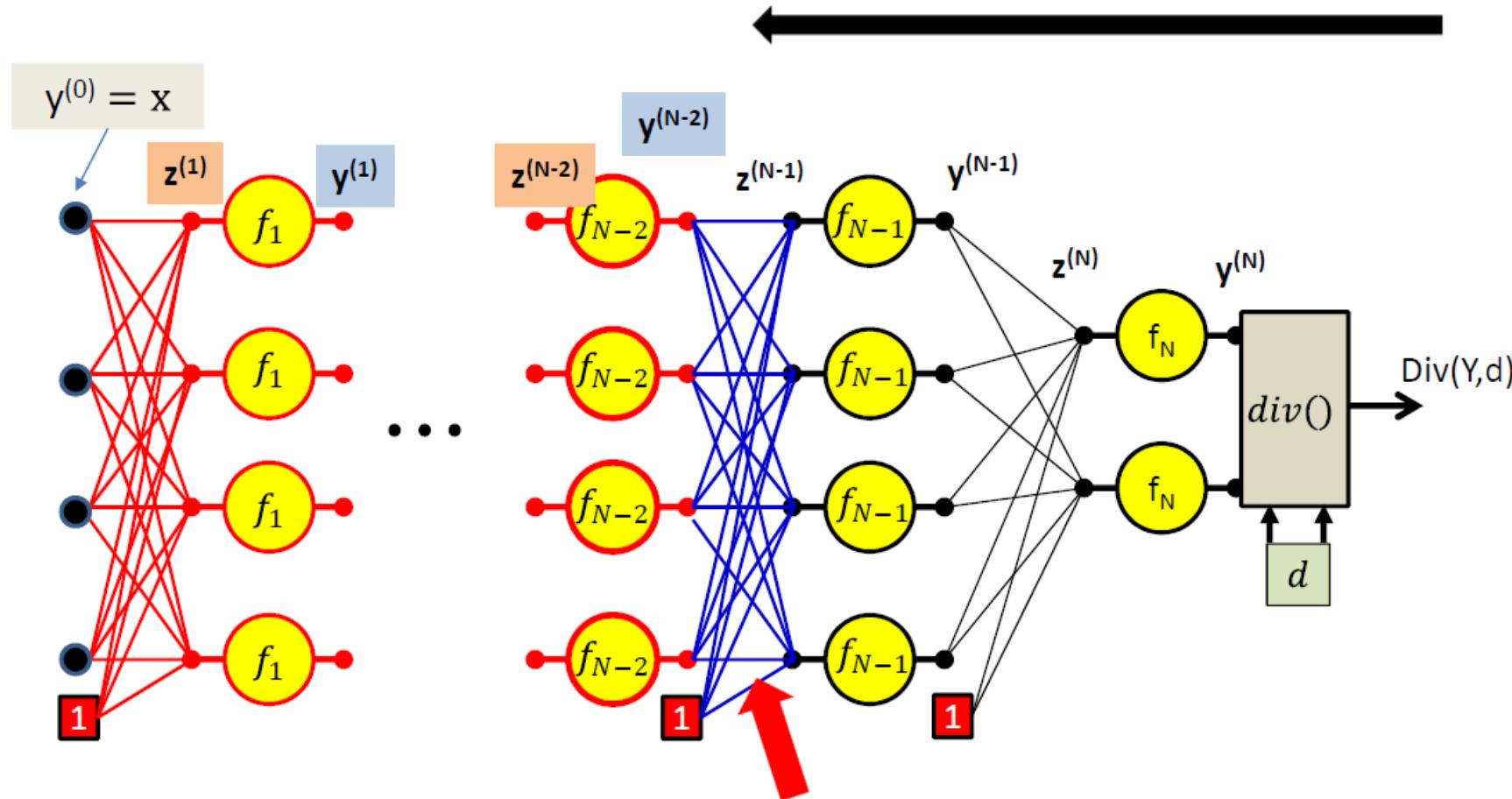


We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial z_i^{(N-1)}} = f'_{N-1}(z_i^{(N-1)}) \frac{\partial Div}{\partial y_i}$$

Copyright © IIIS, Tsinghua University

Computing Gradients



We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial Div}{\partial z_j^{(N-1)}} \quad \text{For the bias term } y_0^{(N-2)} = 1$$

Backpropagation

- Run forward pass to save all the values and then compute gradients in the reverse order
- Output layer (N):

- Directly compute gradients $\frac{\partial L}{\partial y_i^{(N)}}$ and $\frac{\partial L}{\partial z_i^{(N)}} = \frac{\partial L}{\partial y_i^{(N)}} \cdot \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$

- For layer $k = N - 1$ down to 0

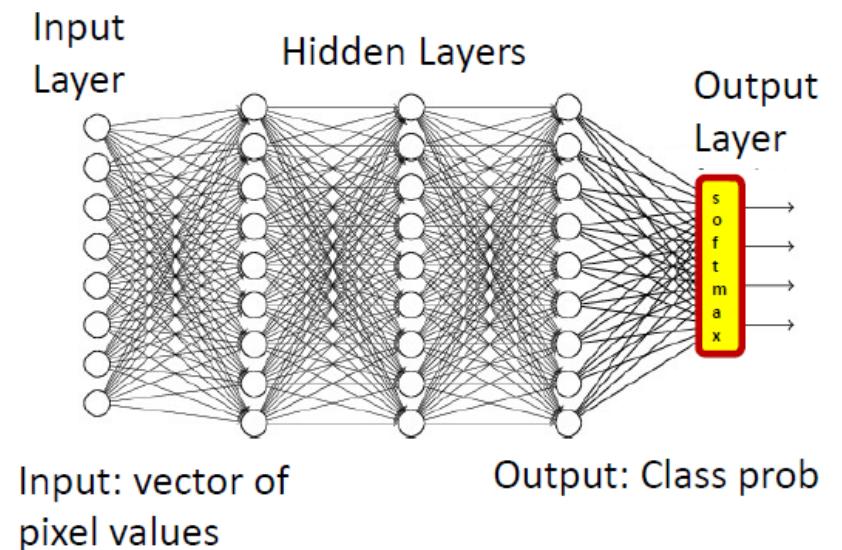
- $\frac{\partial L}{\partial w_{i,j}^{(k+1)}} = y_i^{(k)} \cdot \frac{\partial L}{\partial z_i^{(k+1)}}$

- $\frac{\partial L}{\partial y_i^k} = \sum_j w_{i,j}^{(k+1)} \frac{\partial L}{\partial z_j^{(k+1)}}$

- $\frac{\partial L}{\partial z_i^{(k)}} = \frac{\partial L}{\partial y_i^{(k)}} f'_k(z_i^{(k)})$

MLP for Classification

- The workflow
 - Design a N -layer MLP neural network
 - Initial weights $\{W^{(k)}\}$
 - Gradient descent until convergence
 - With backpropagation

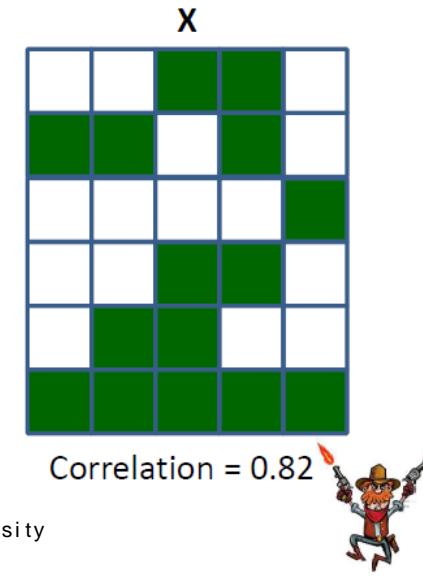
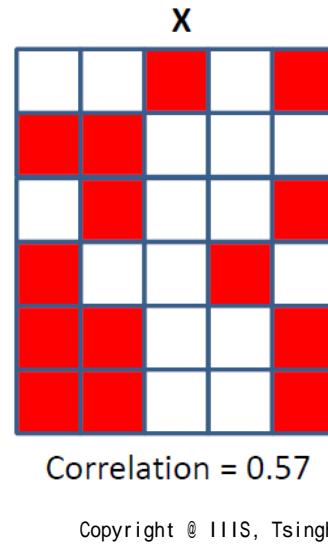


Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Intuition: in the image classification problem
 - Let's consider a particular neuron in the input layer: $y = f(w^T x)$;
 - y is activated when x is more ***correlated*** with the weight

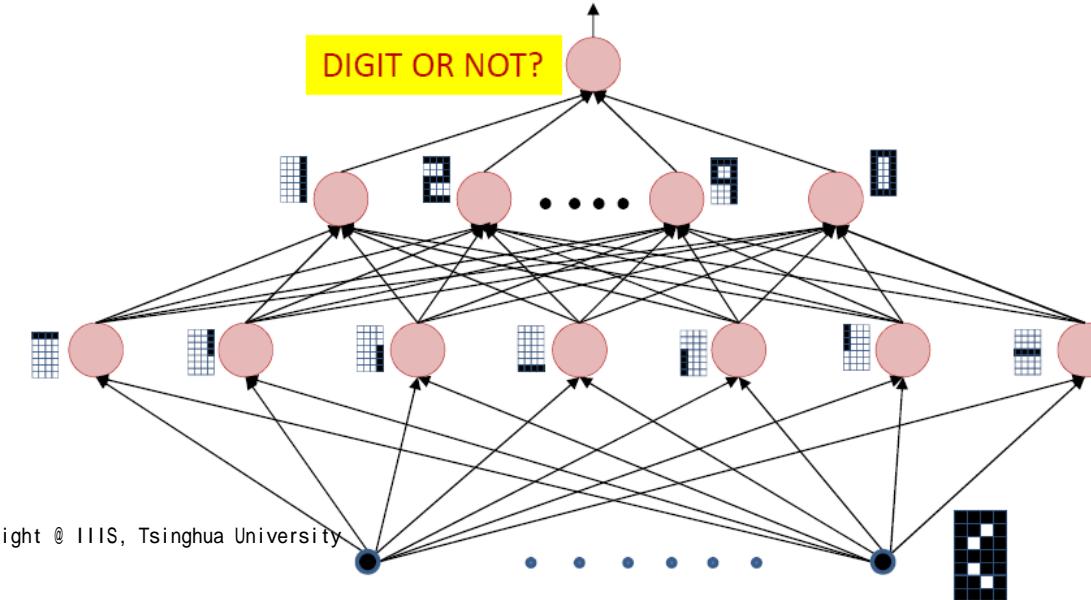
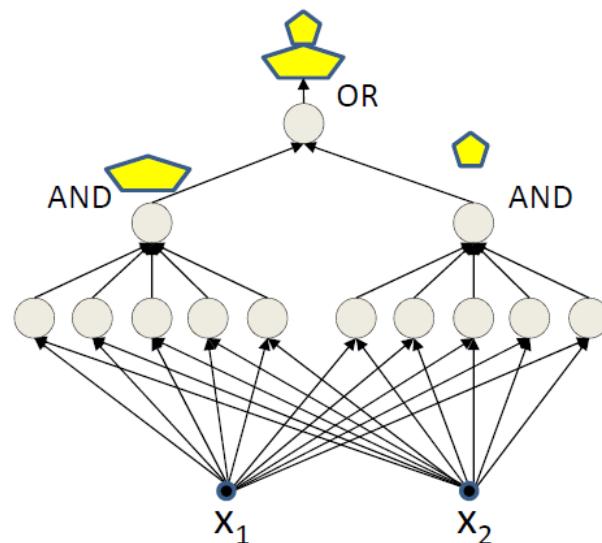
w

$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$



Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Intuition: in the image classification problem
 - MLP learns a cascade of features
 - It is important for the first layer to capture important low-level features

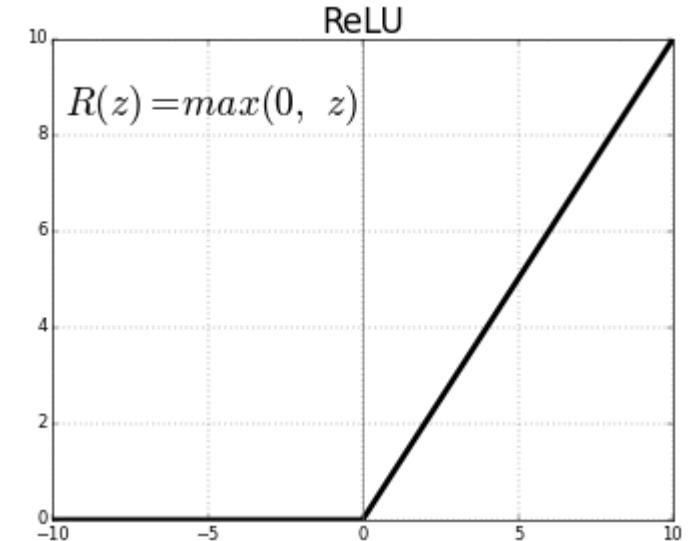


Deep Learning in Practice

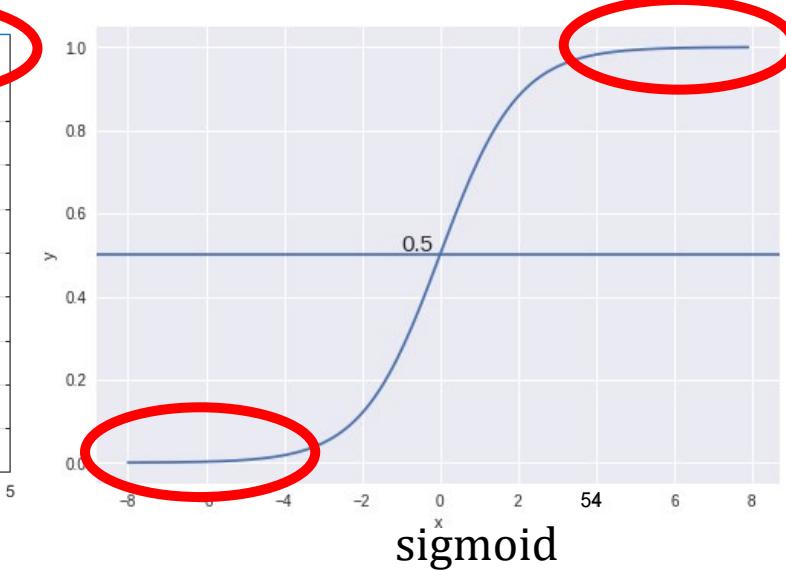
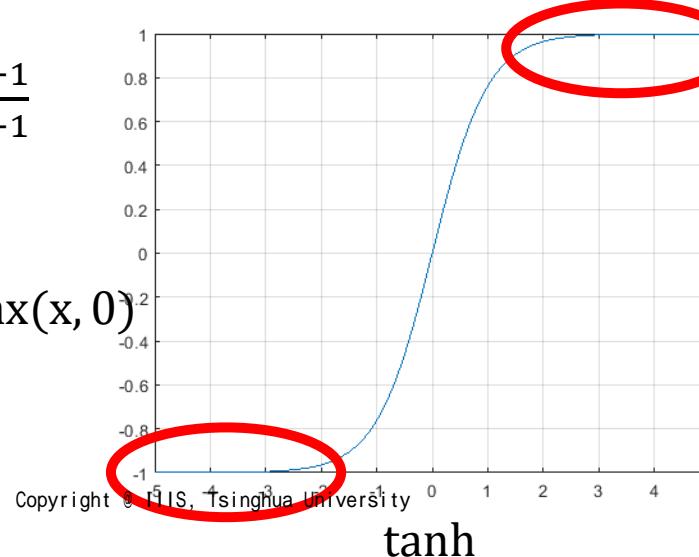
- How to design the MLP?
 - Width and depth
- Intuition: MLP learns a cascade of features
 - Practical suggestion: more neurons in the low level
- Depth?
 - Deeper network are harder to learn
 - Intuition: gradients are **products** over layers
 - hard to control the learning rate
 - More on next lecture to address this challenge for really deep networks

Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Activation Functions?
 - Sigmoid has a beautiful probability interpretation but ...
 - Issues with Sigmoid function
 - Always non-negative
 - Alternative: $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$
 - Gradient vanishing
 - Initialization matters!
 - Alternative: $\text{ReLU}(x) = \max(x, 0)$
 - ... and more variants

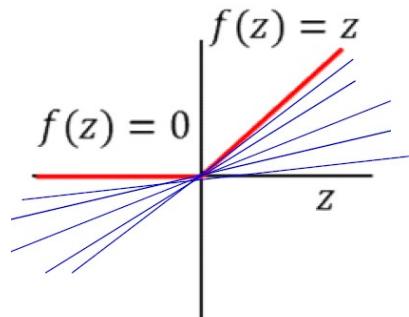


Rectified Linear United



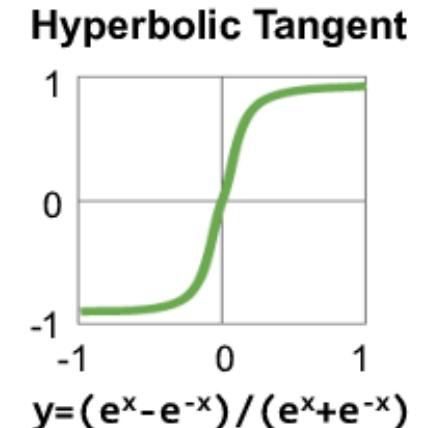
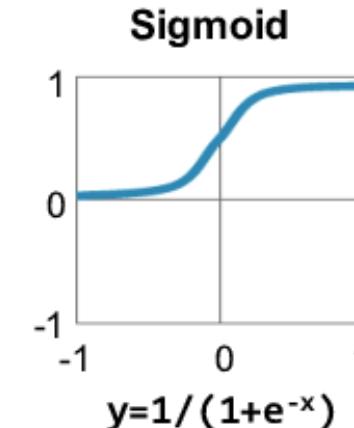
Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Activation Functions
 - A collection of ReLU variants
 - Subgradients
 - A direction that decrease the function
 - Gradients are subgradients but not vice versa

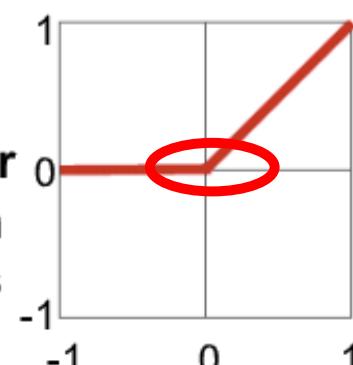


2/24

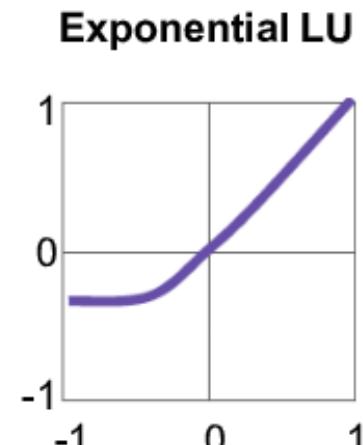
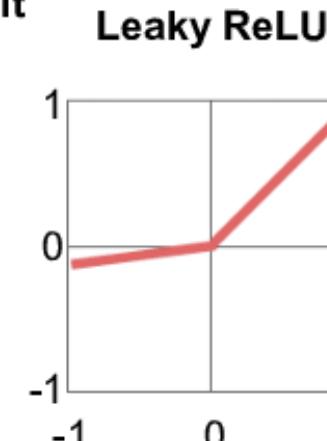
Traditional Non-Linear Activation Functions



Rectified Linear Unit (ReLU)



Modern Non-Linear Activation Functions



α = small const. (e.g. 0.1)

Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Activation Functions
 - ReLu and Subgradients
- Quiz
 - $f(X) = \max(x_1, x_2, \dots, x_n);$
 - Compute $\frac{\partial f}{\partial x_i}$?

Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Activation Functions
 - ReLu and Subgradients
- Learning rate
 - For now: larger learning rate can lead to divergence while small learning rate may lead to no progress
 - More on next lecture

Deep Learning in Practice

- How to design the MLP?
 - Width and depth
- Activation Functions
 - ReLu and Subgradients
- Learning rate
- Regularization
 - Tricks to stabilize the learning process
 - L2 norm on all the weights.
 - $L(w) = Loss(w) + \alpha|w|^2$
 - This is also called weight decay
 - $\nabla_w L = \nabla Loss(w) + \alpha w$

Deep Learning in Practice

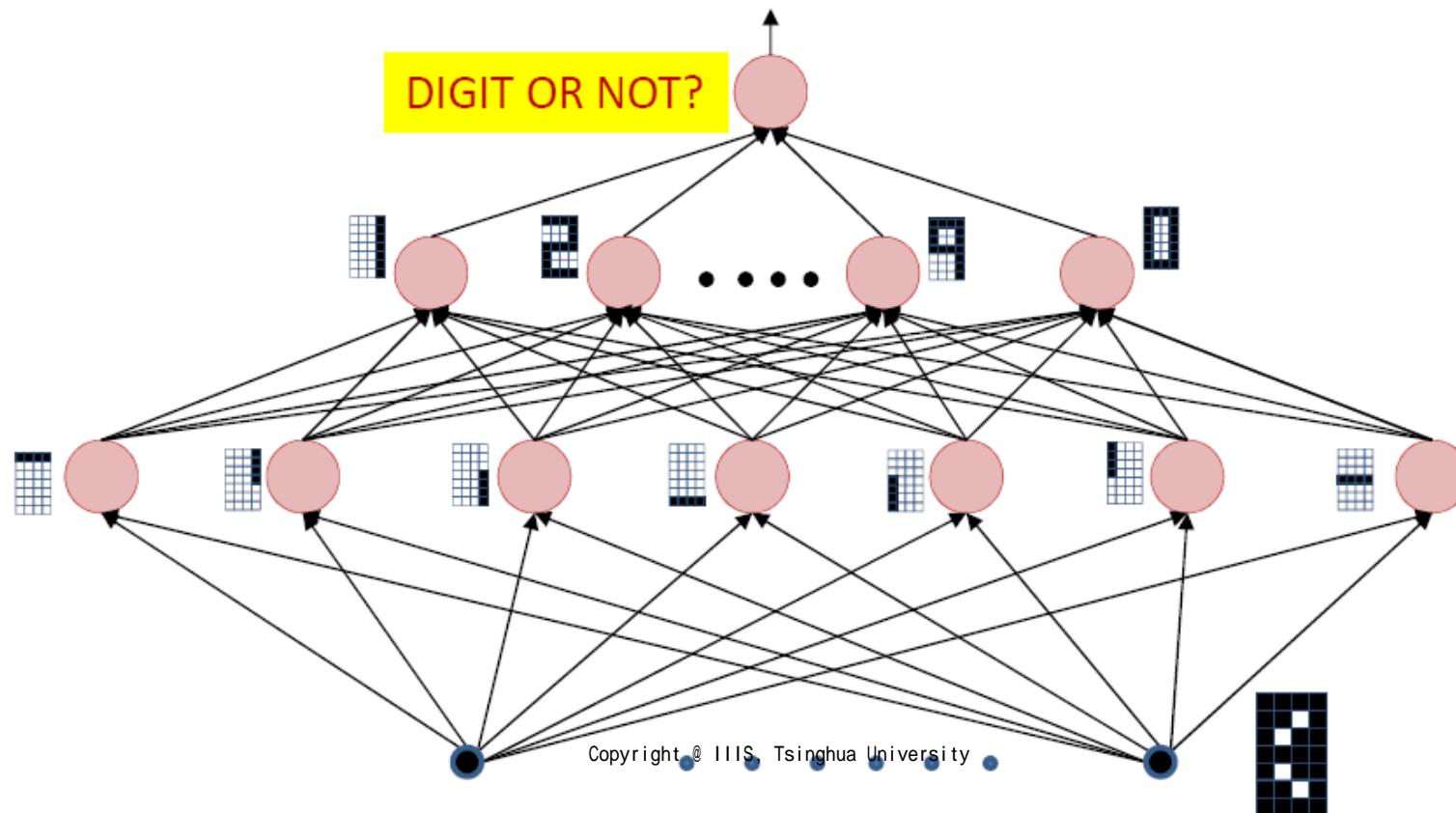
- How to design the MLP?
 - Width and depth
- Activation Functions
 - ReLu and Subgradients
- Learning rate
- Regularization
- Let's get hand dirty!
 - <http://playground.tensorflow.org/>

Today's Lecture

- Part 1: The simplest deep learning application ---- classification!
 - The very basic ideas of deep learning and backpropagation
 - Get a sense of parameter tuning (调参/炼丹)
- **Part 2: Convolutional Neural Networks (CNN)**
 - The very basic ideas of CNN
- Let's get a sense of deep learning “algorithm”
 - More tricks and ideas to come in the next lecture

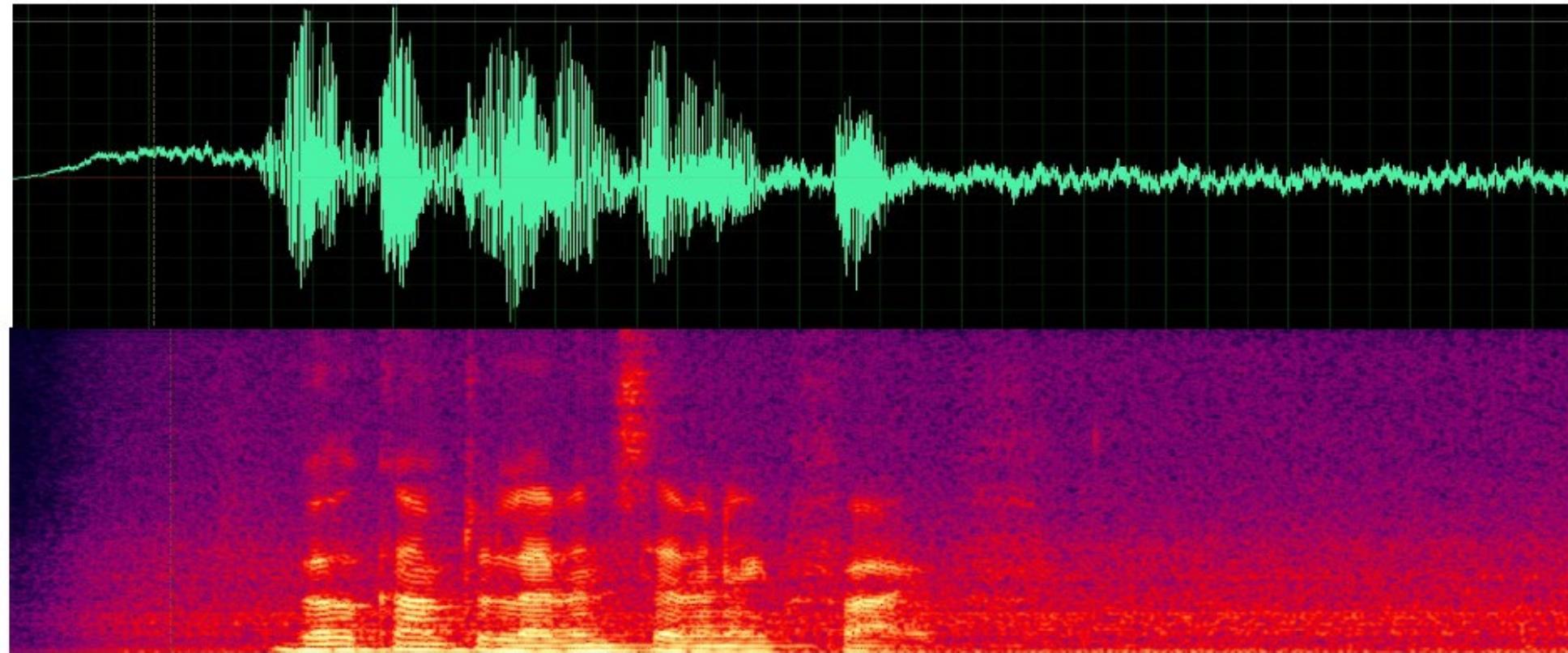
Recap: What does MLP learn?

- A cascade of features (patterns)
 - Weights are correlated filters



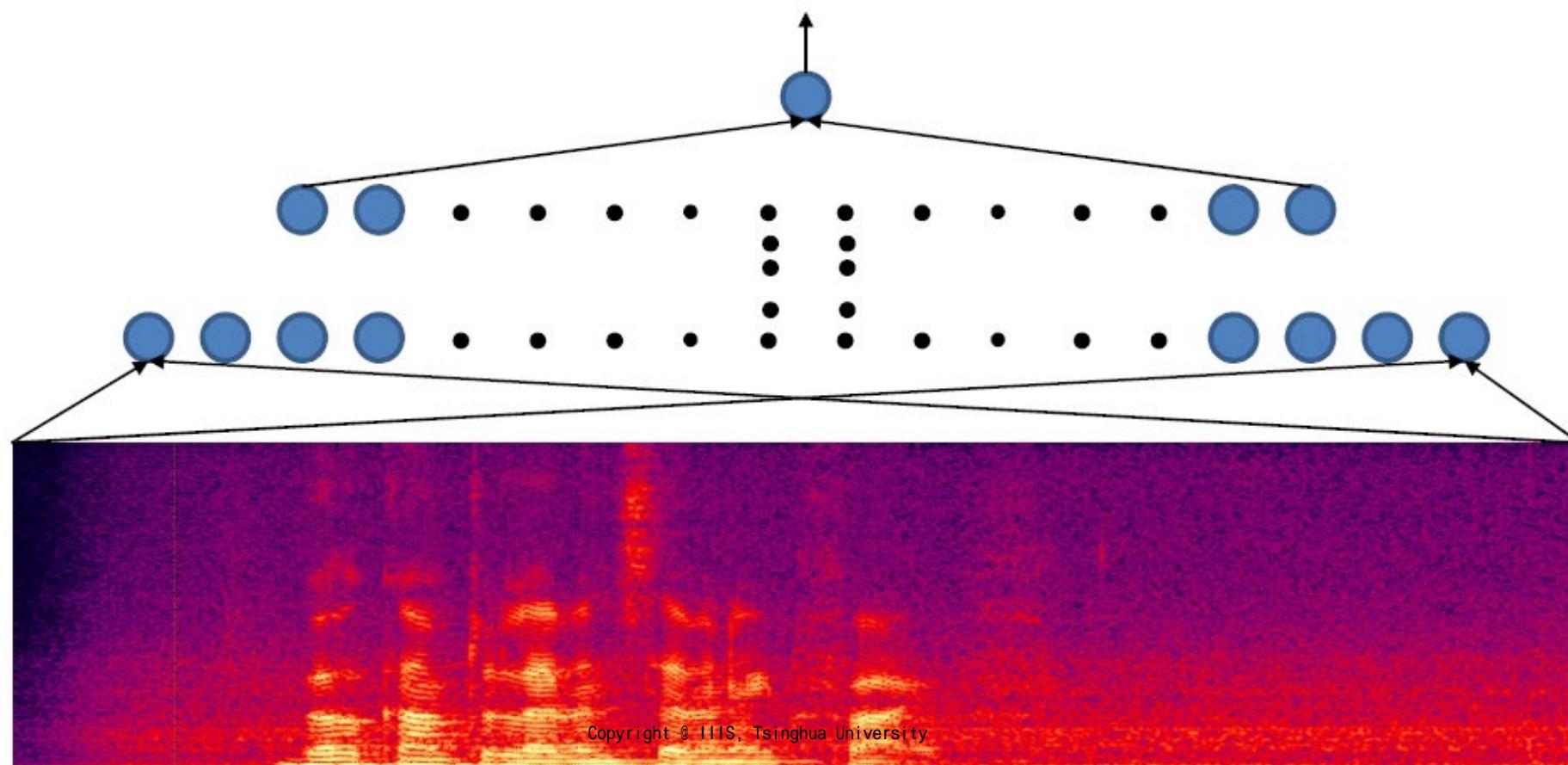
Finding the welcome

- Does the signal contain “welcome”?



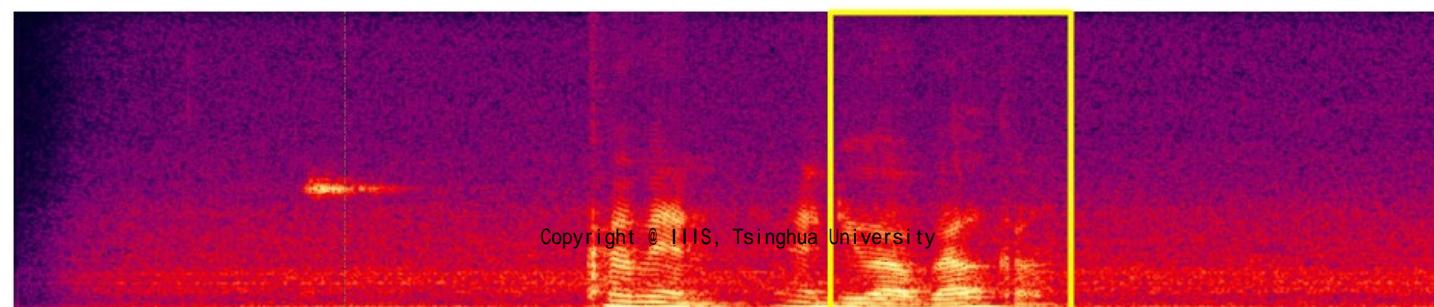
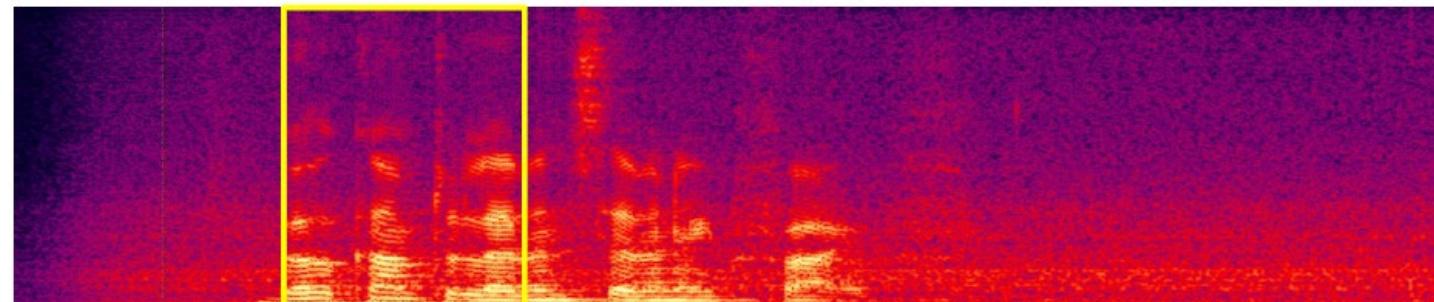
Finding the welcome

- A trivial solution: train a MLP!



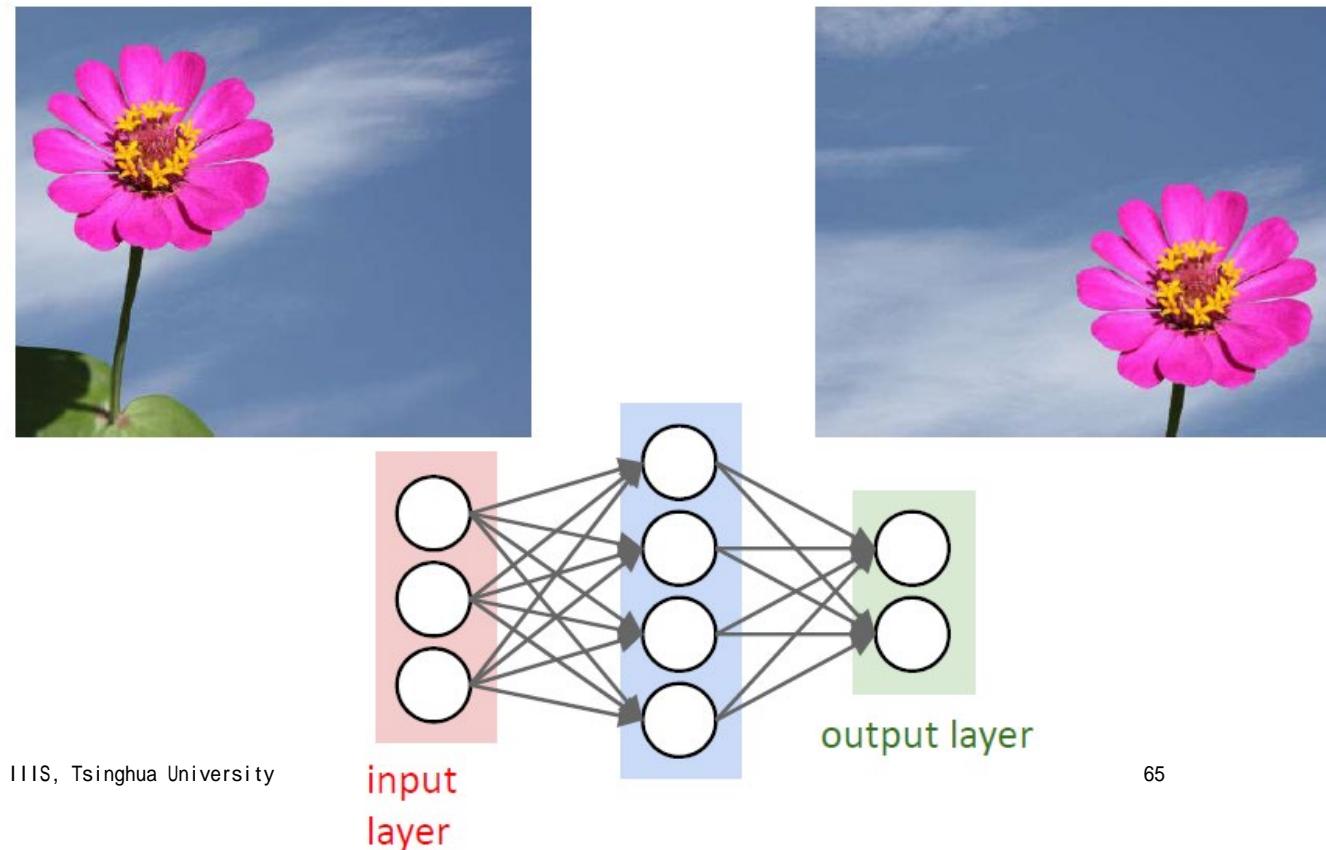
Finding the welcome

- The issue
 - The filter detects the first welcome does not apply to the second
 - We need a huge amount of training data!
- We need a *simple* network that can fire regardless of the pattern *location*



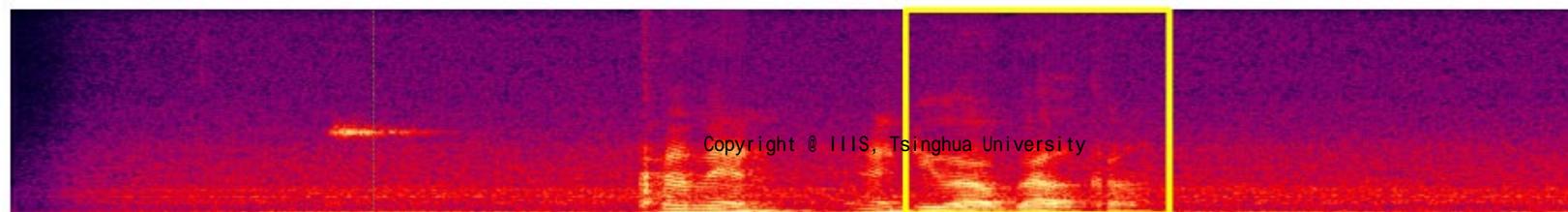
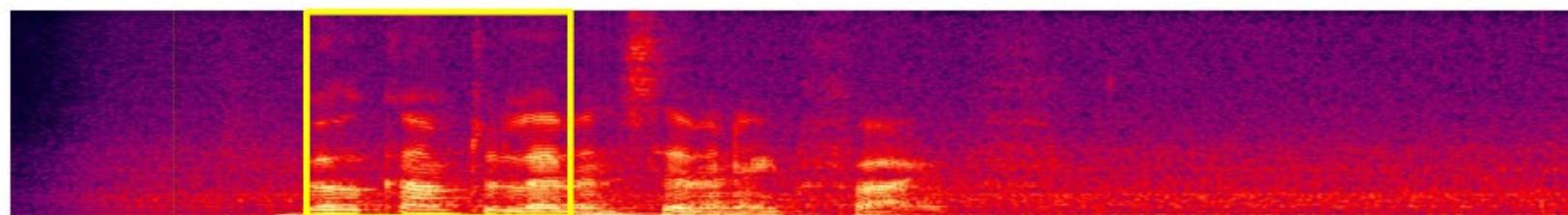
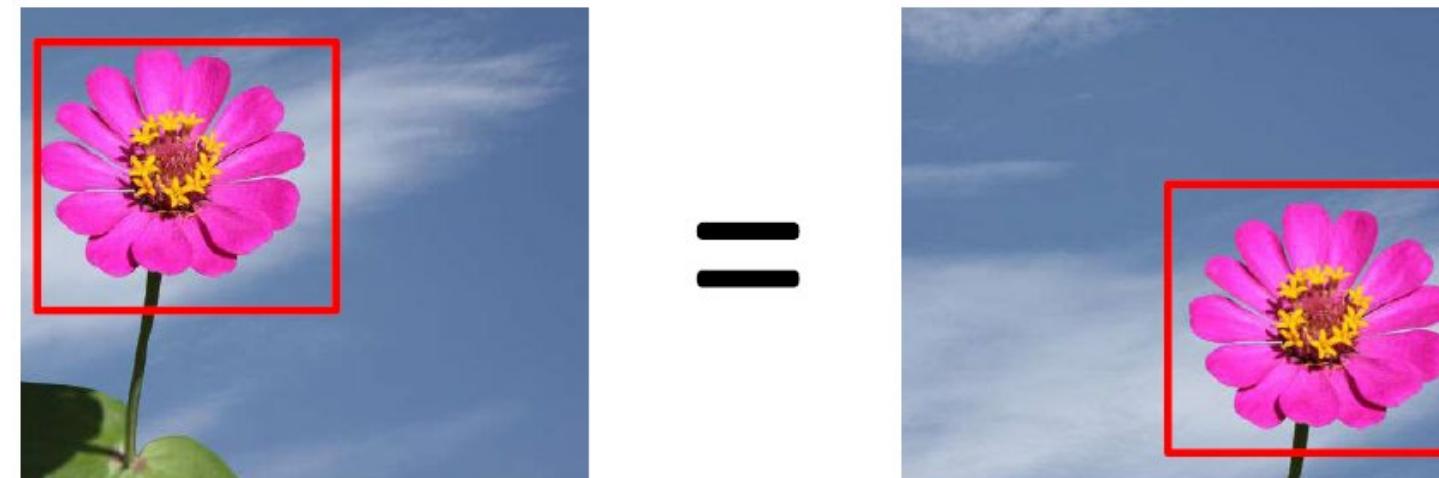
Finding the flowers

- Another example
 - Is there a flower in any of the images?
 - Similar issue:
 - An MLP detecting the left does not apply to the right one



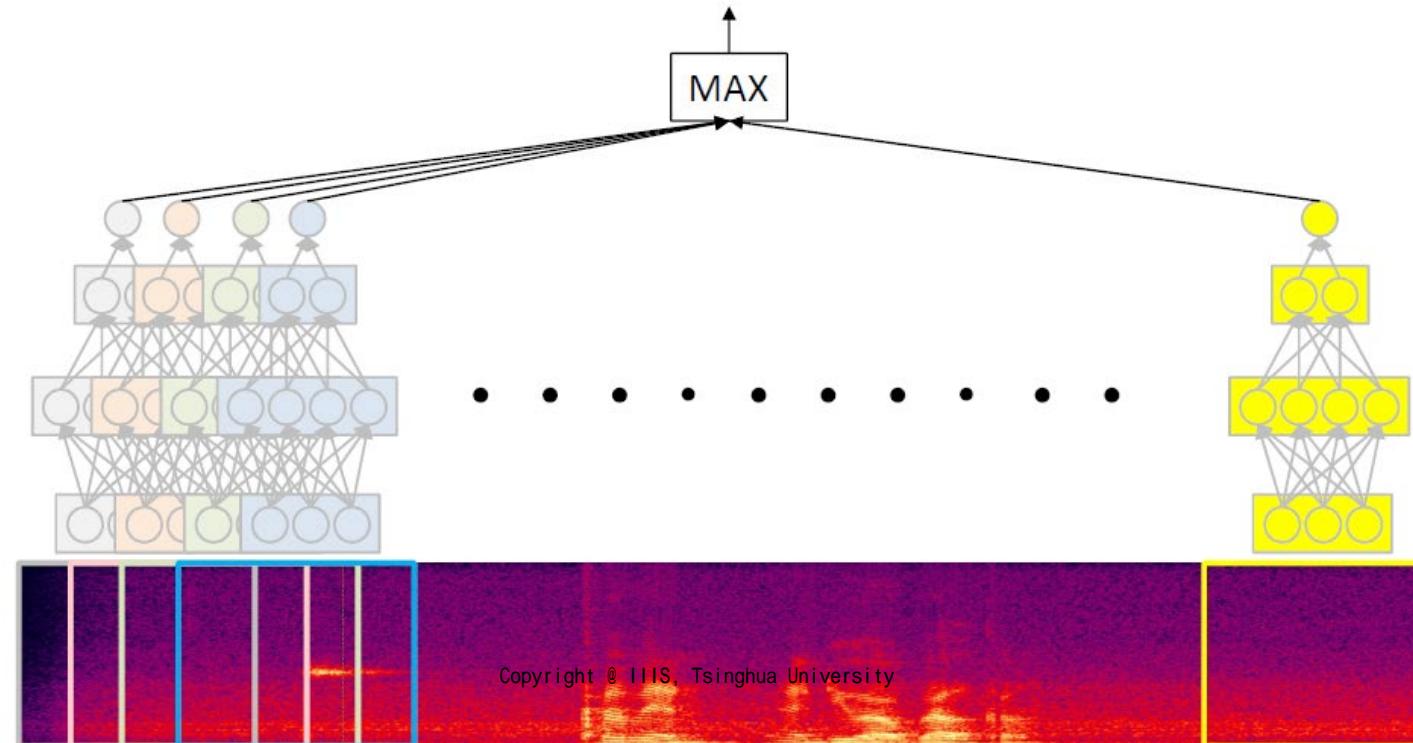
Shift Invariance

- The network needs to be shift invariant



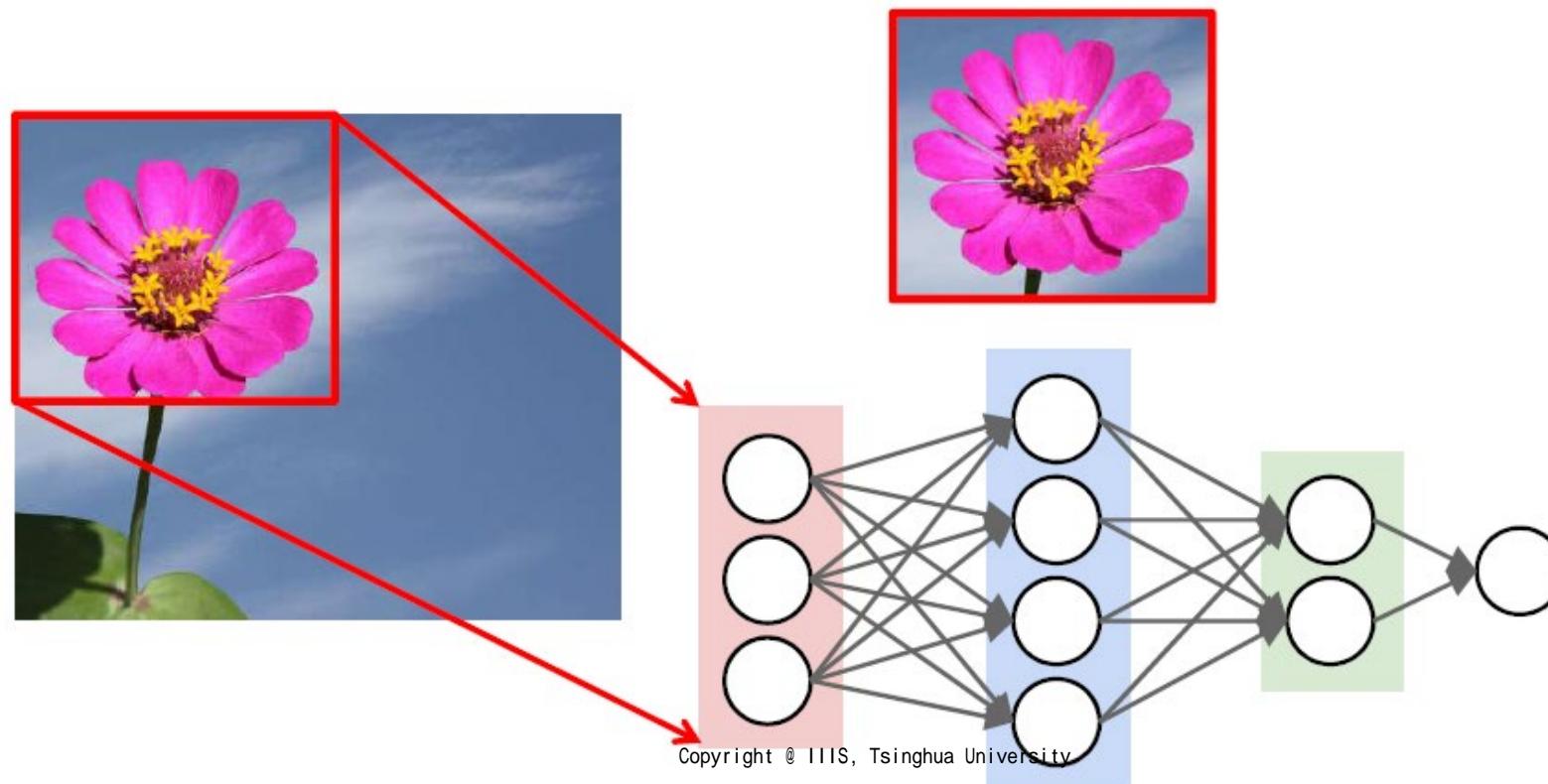
A Simple solution: Scan

- Use a **shared** MLP to detect every possible local patterns
 - Use a maximum (Boolean OR) over the activations over all the locations



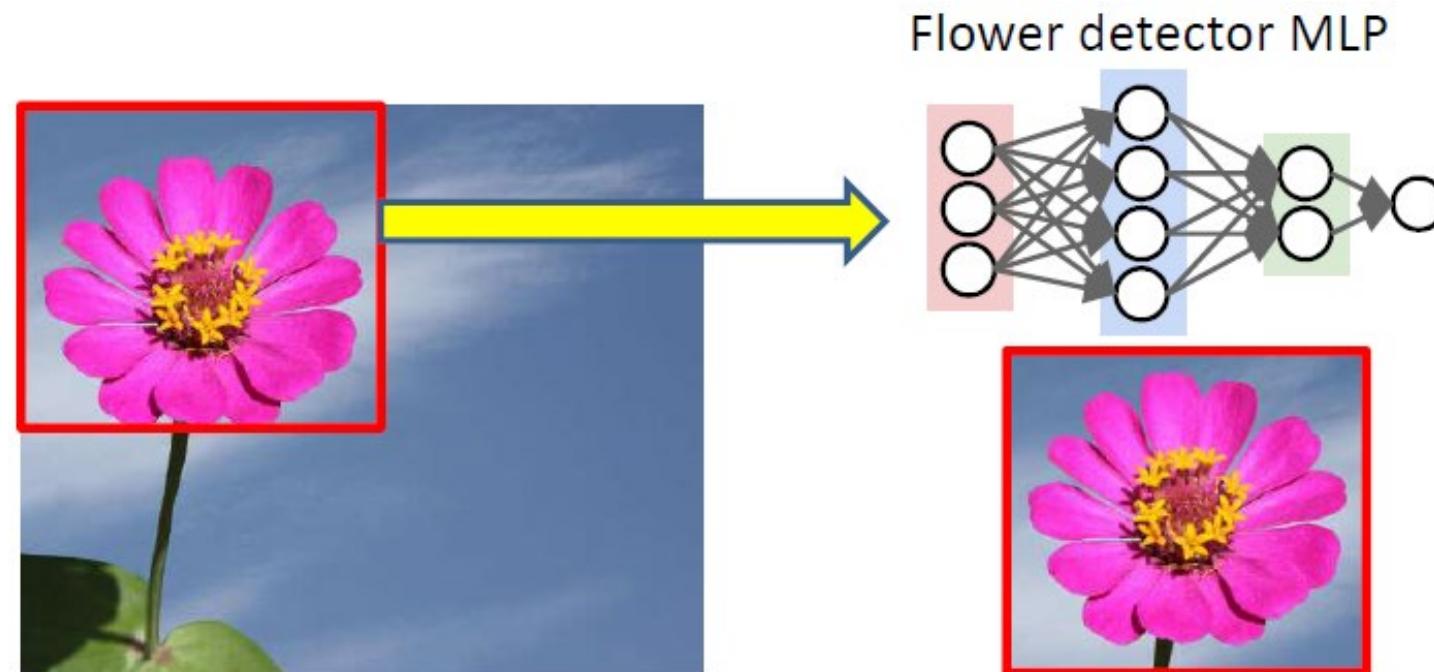
A Simple solution: Scan in 2D

- Detecting a flower using a **shared** MLP



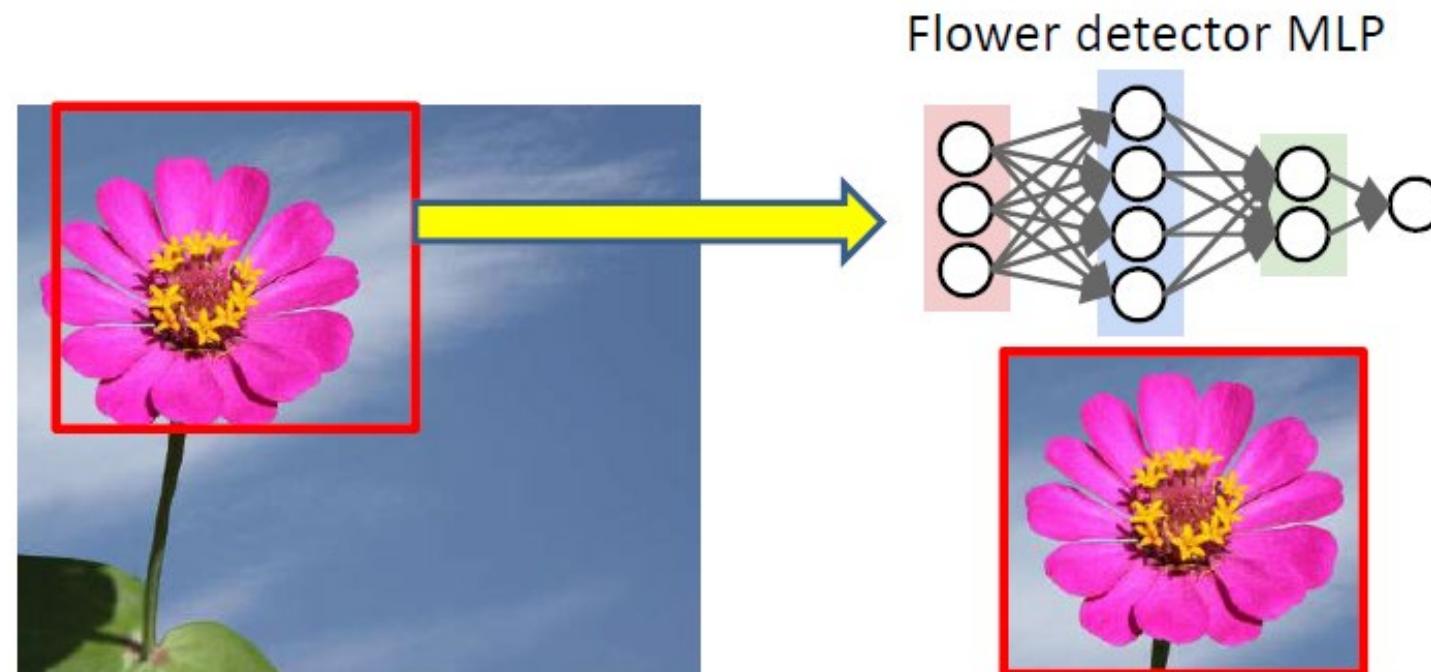
A Simple solution: Scan in 2D

- Detecting a flower using a **shared** MLP
 - Look at every possible positions



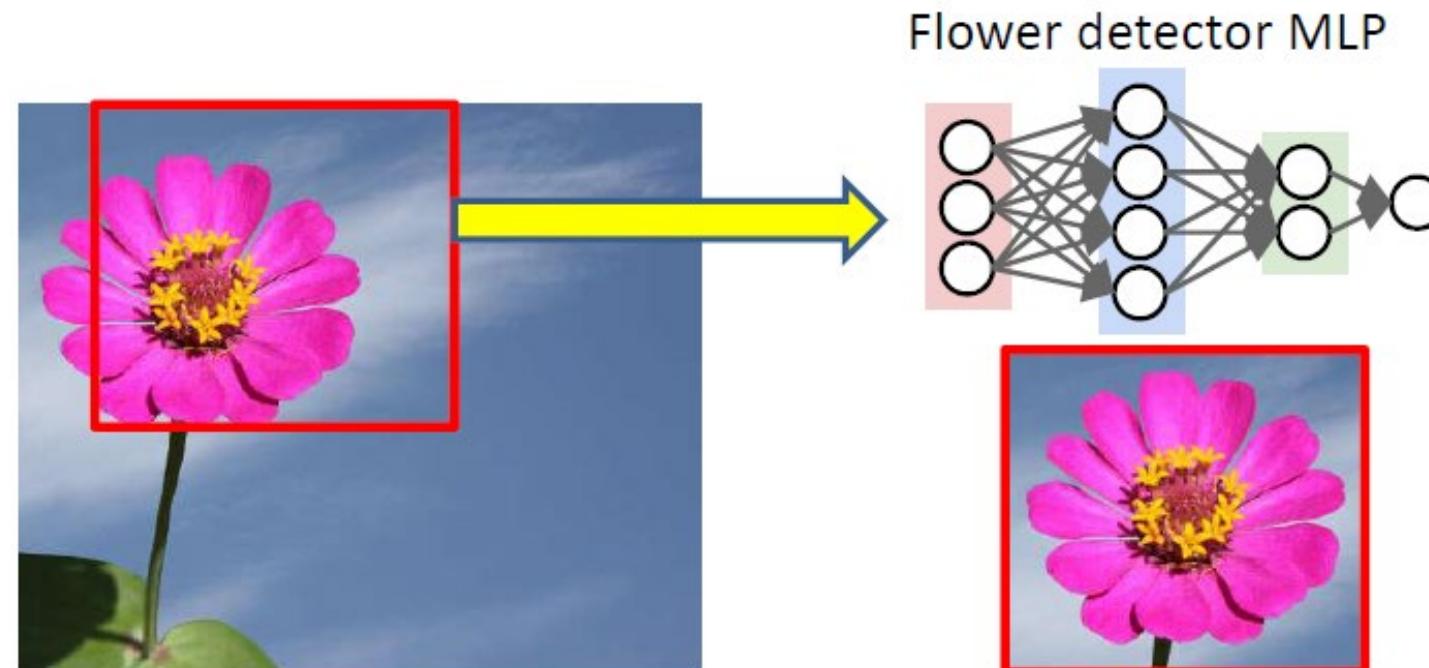
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



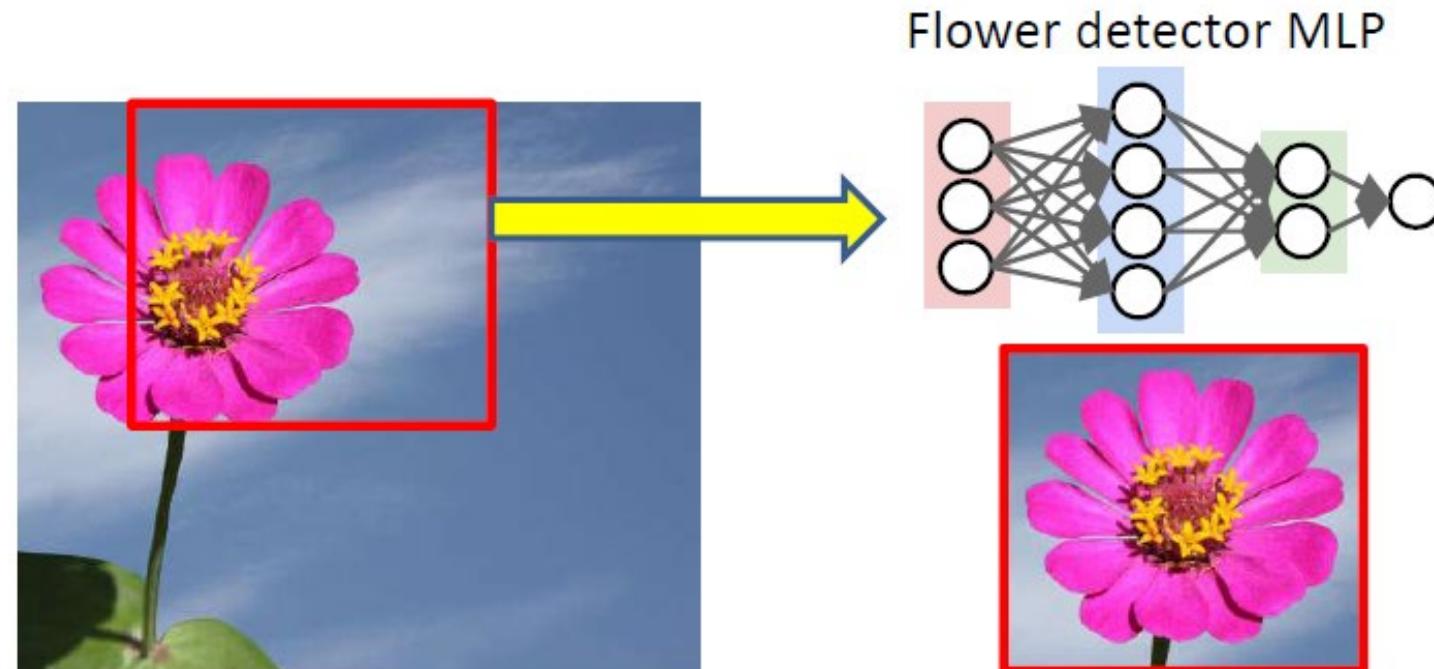
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



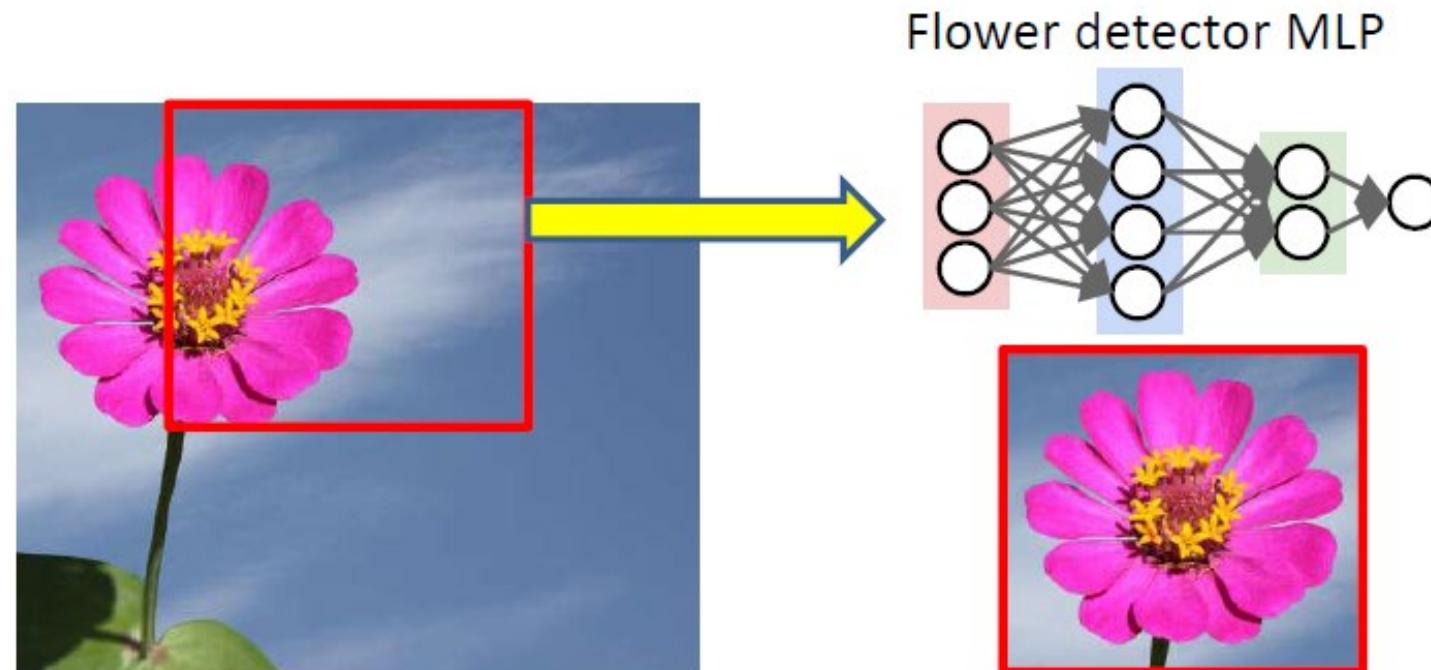
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



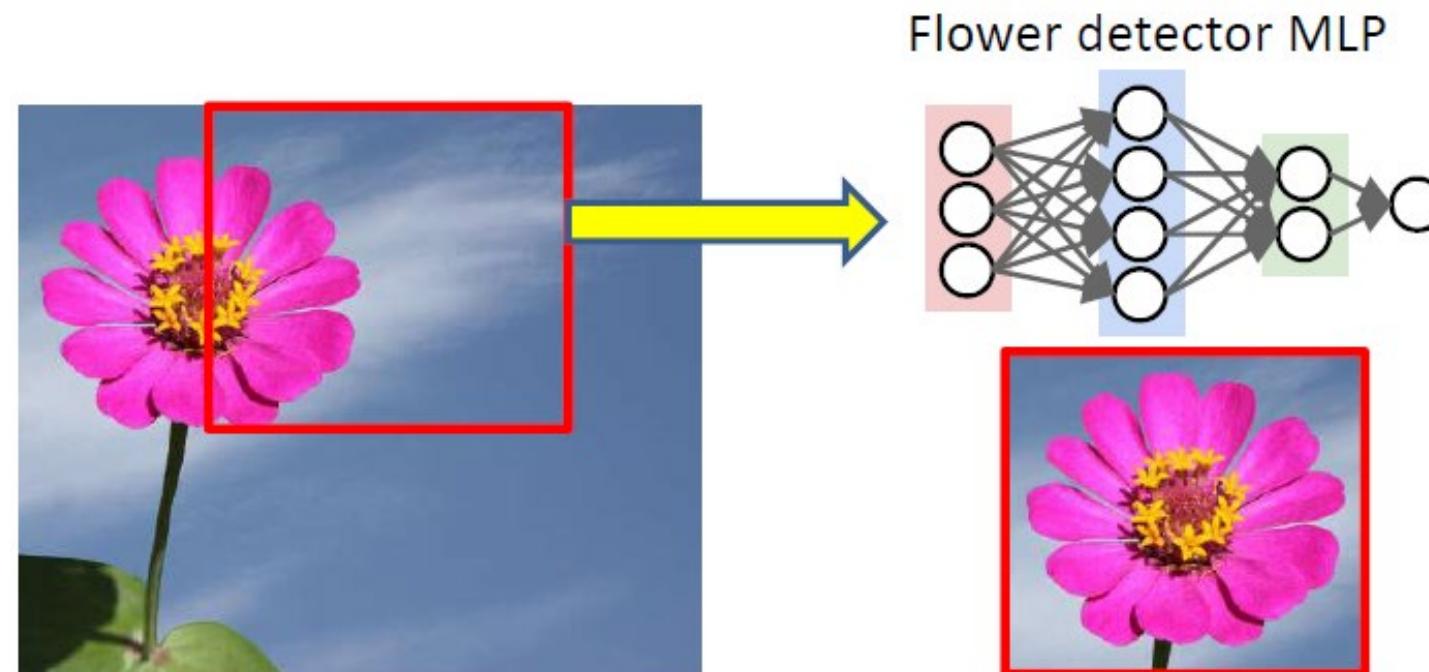
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



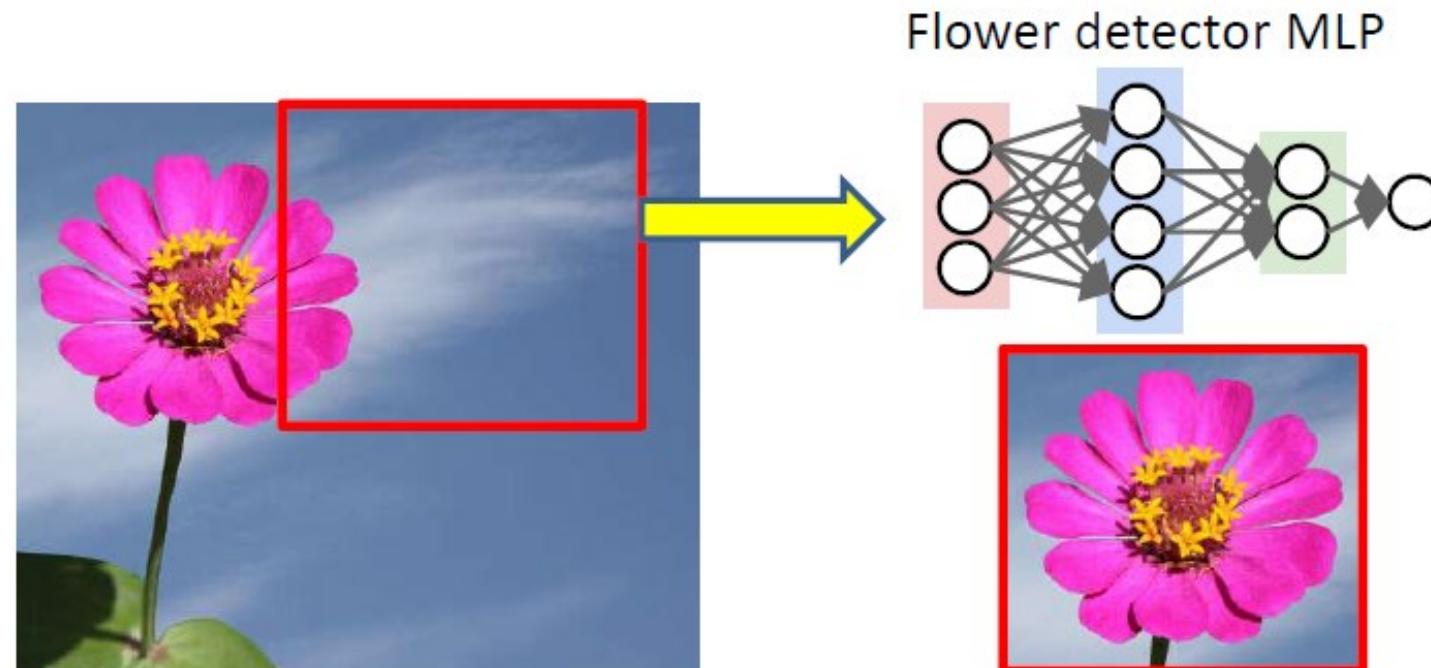
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



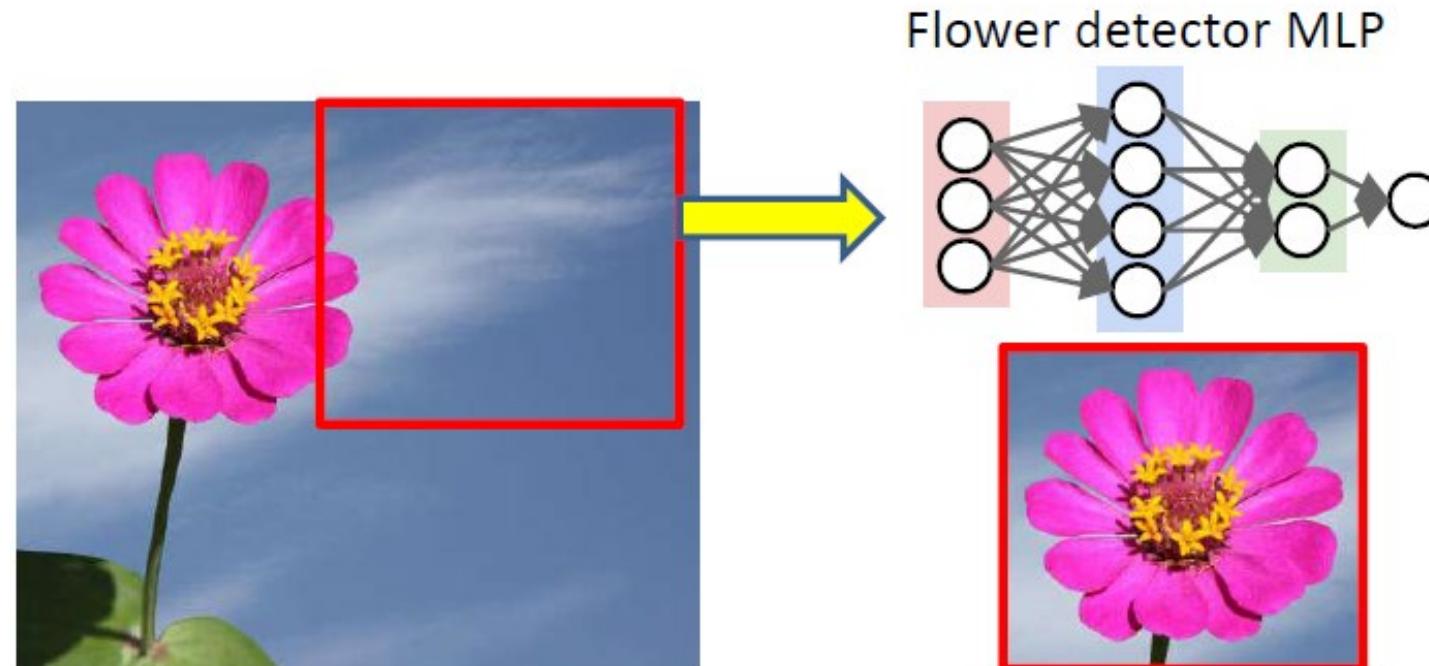
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



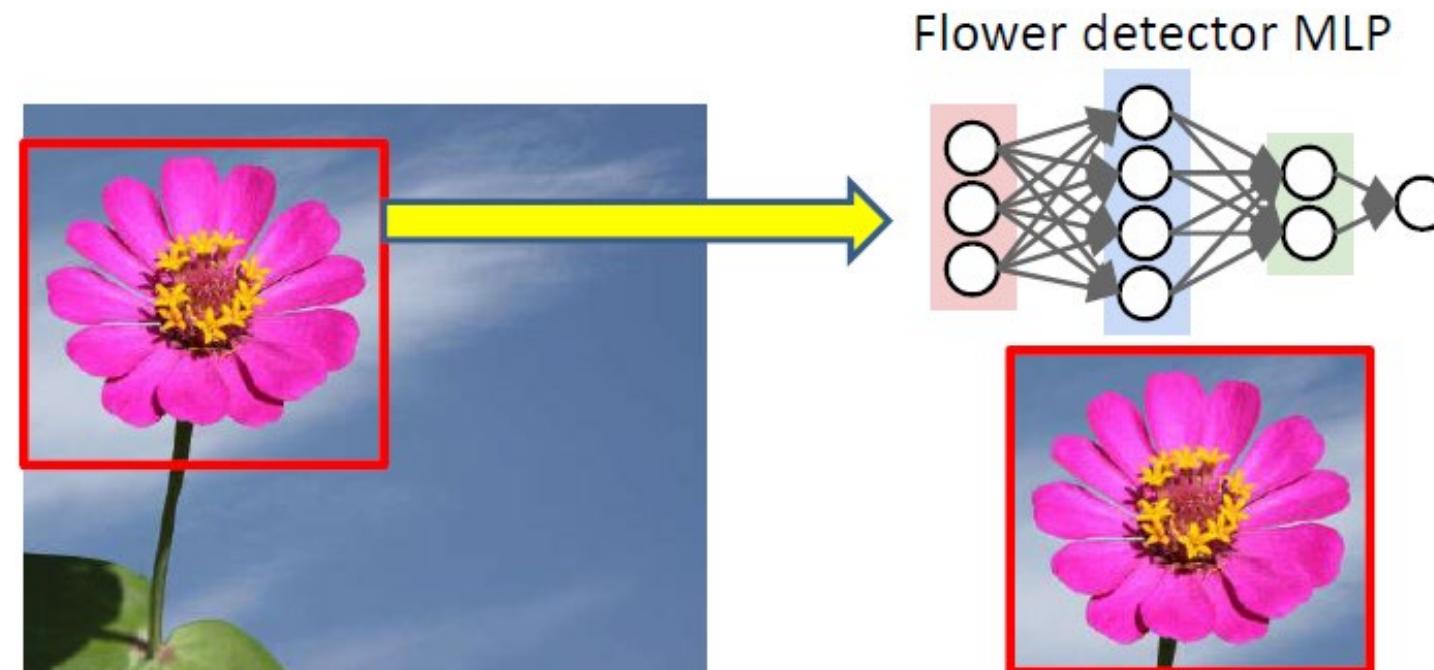
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



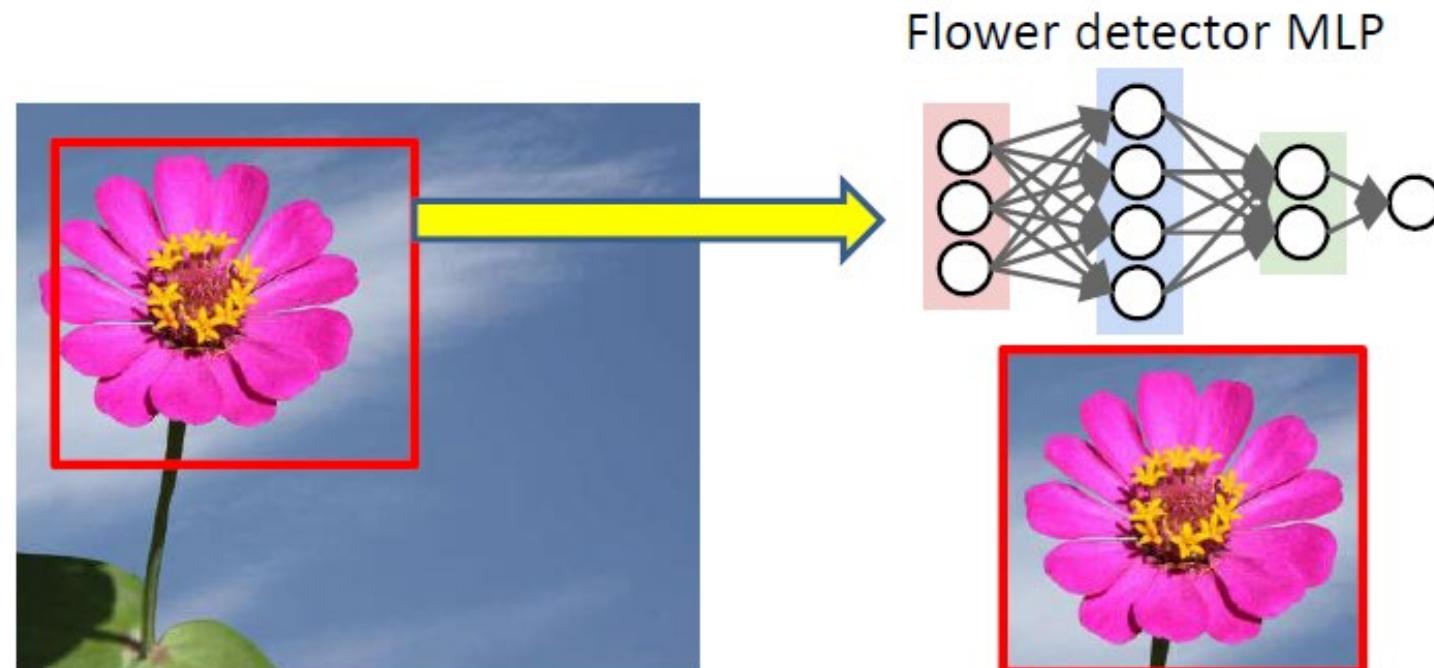
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



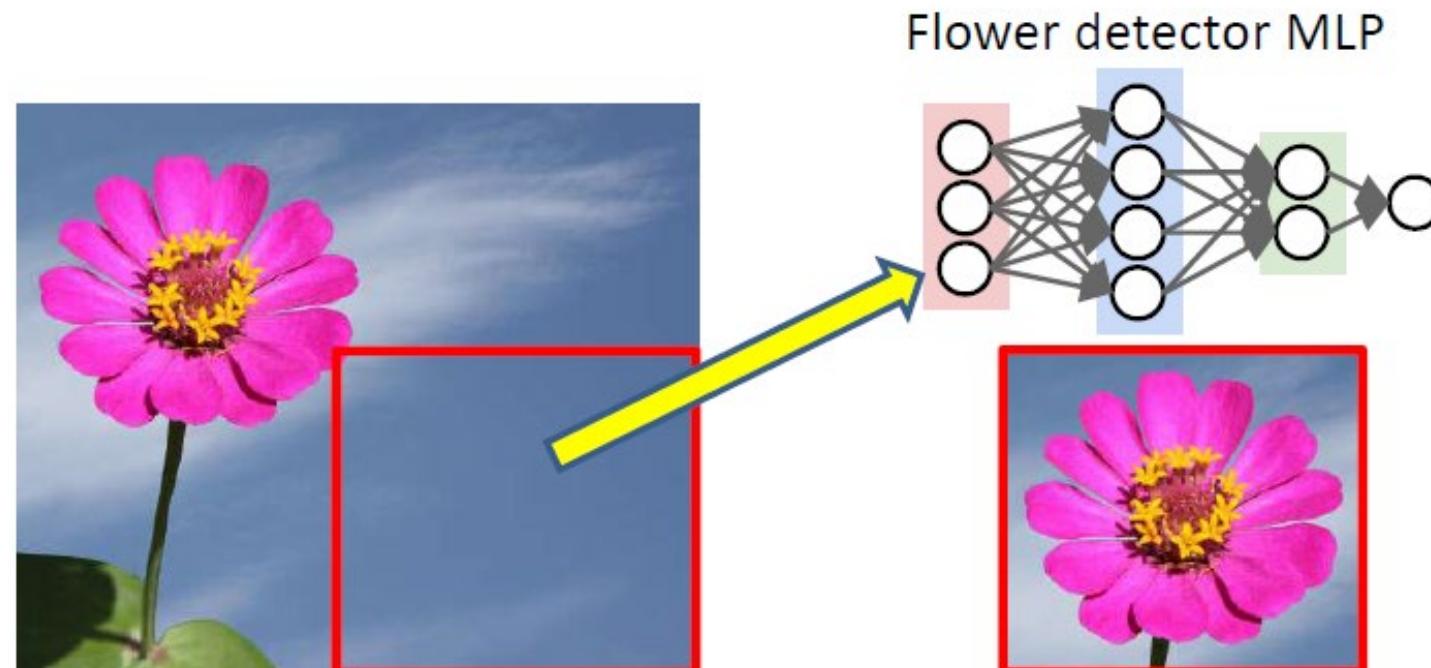
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



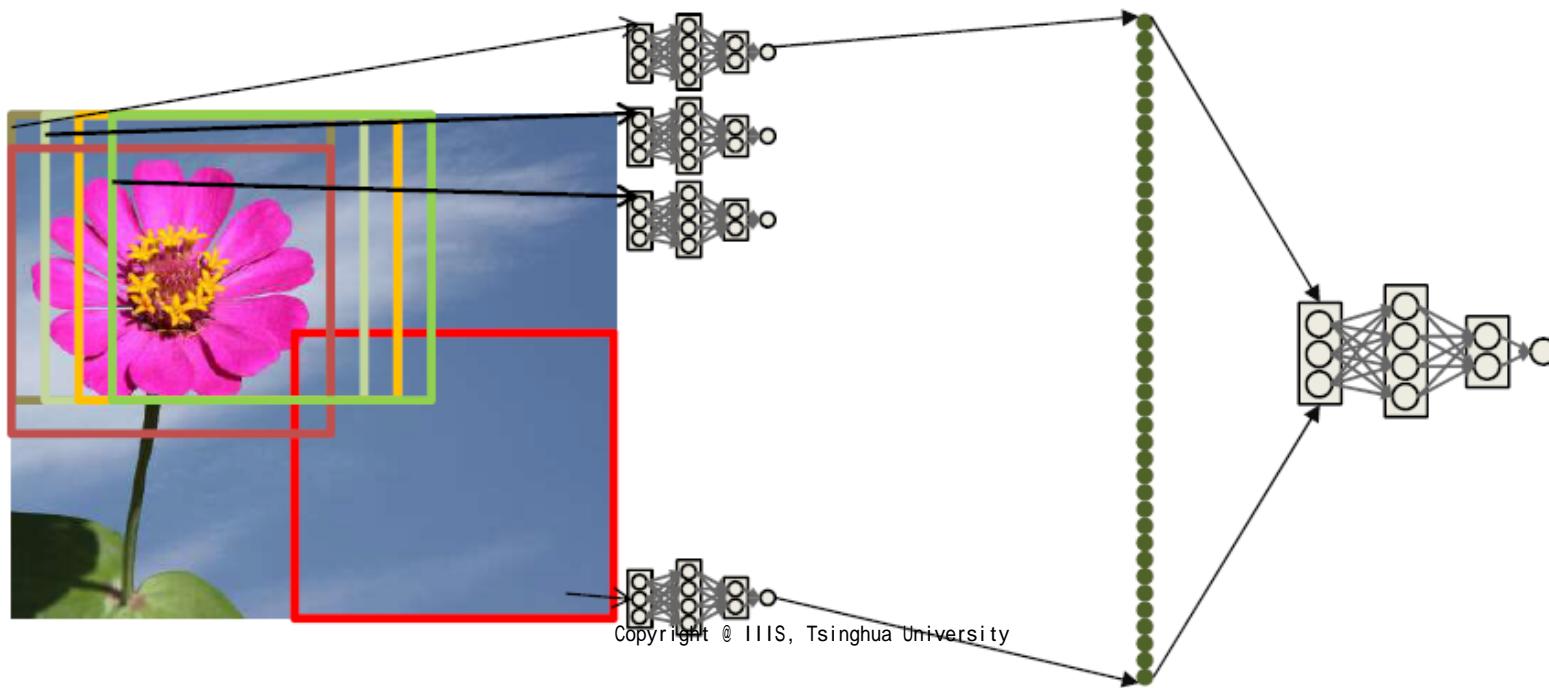
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions



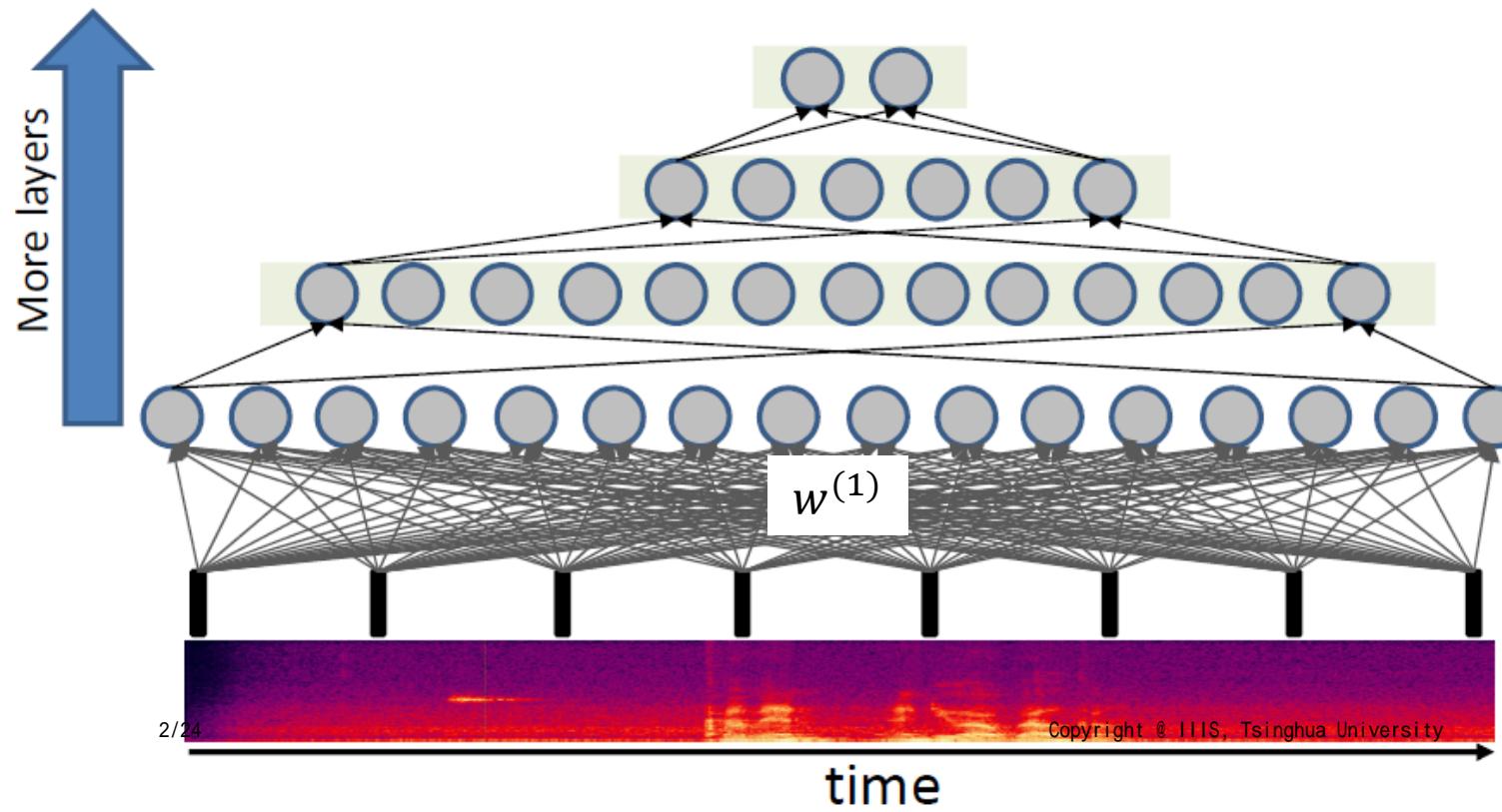
A Simple solution: Scan in 2D

- Detecting a flower using a **shared MLP**
 - Look at every possible positions
 - Send the local patch to the same MLP
 - take the outputs to a final MAX or MLP



Regular MLP v.s. Scanning MLP

- Regular MLP
 - Extremely dense & high-dimensional weight matrix (NM params per layer)

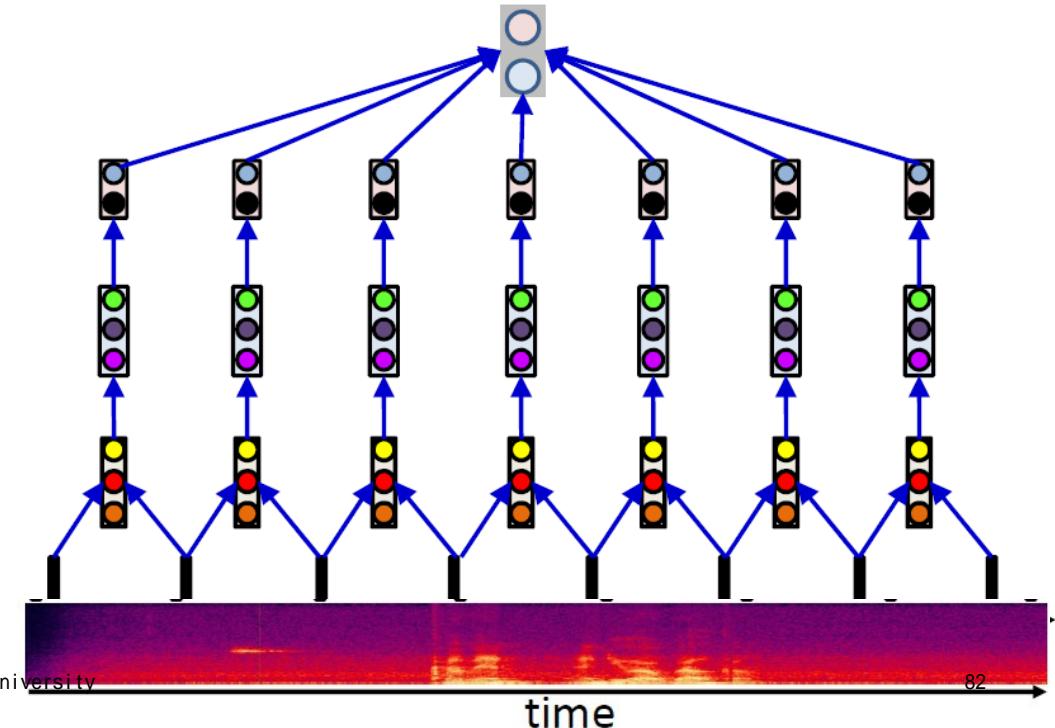


$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & \cdots & \cdots & \cdots & w_{1M} \\ w_{21} & w_{22} & w_{23} & w_{24} & \cdots & \cdots & \cdots & w_{2M} \\ w_{31} & w_{32} & w_{33} & w_{34} & \cdots & \cdots & \cdots & w_{3M} \\ w_{41} & w_{42} & w_{43} & w_{44} & \cdots & \cdots & \cdots & w_{4M} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & w_{N4} & \cdots & \cdots & \cdots & w_{NM} \end{bmatrix}$$

Regular MLP v.s. Scanning MLP

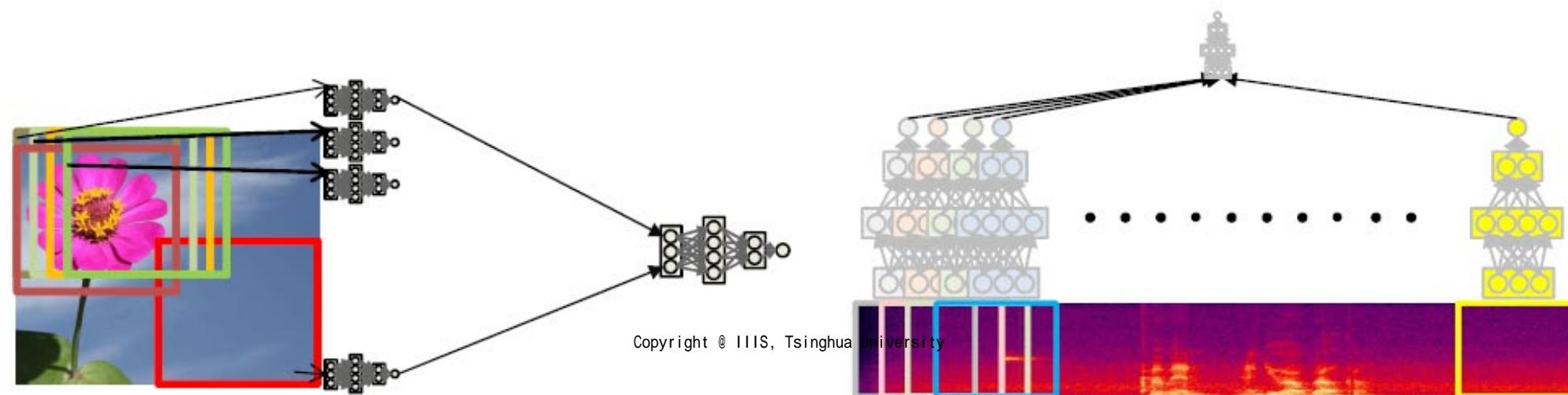
- Scanning MLP

- Only require a small amount of parameters (the shared MLP for local patch)
- *Effective in any situation where the data are expected to be composed of similar structures at different locations*
 - E.g. speech recognition, image recognition



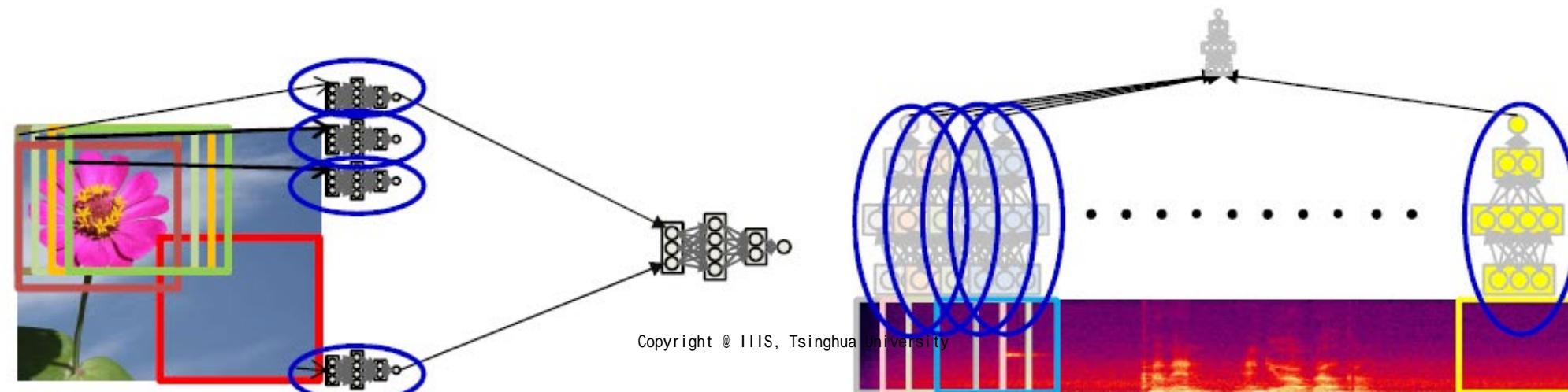
Training the Network

- Still Backpropagation!
 - Fully differentiable neural networks



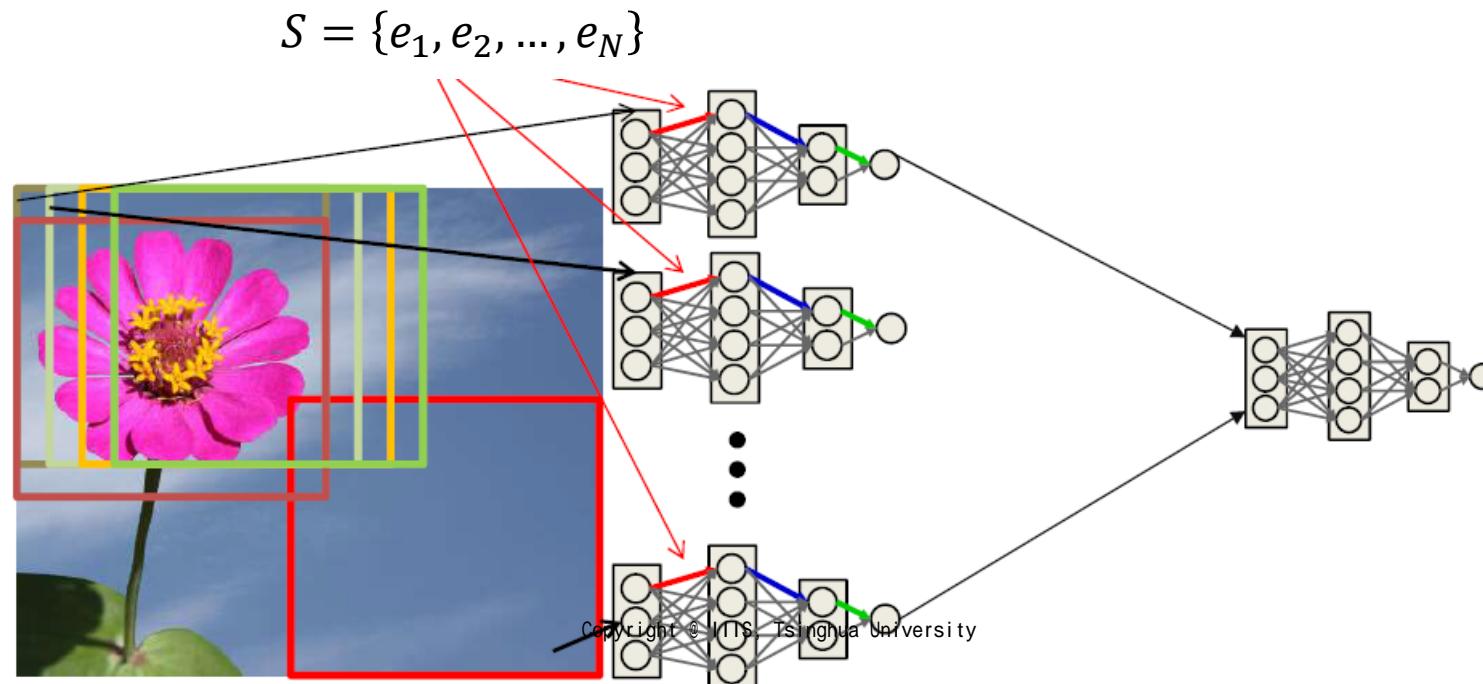
Training the Network

- Still Backpropagation!
 - Fully differentiable neural networks
 - But with constraints --- shared parameter models!



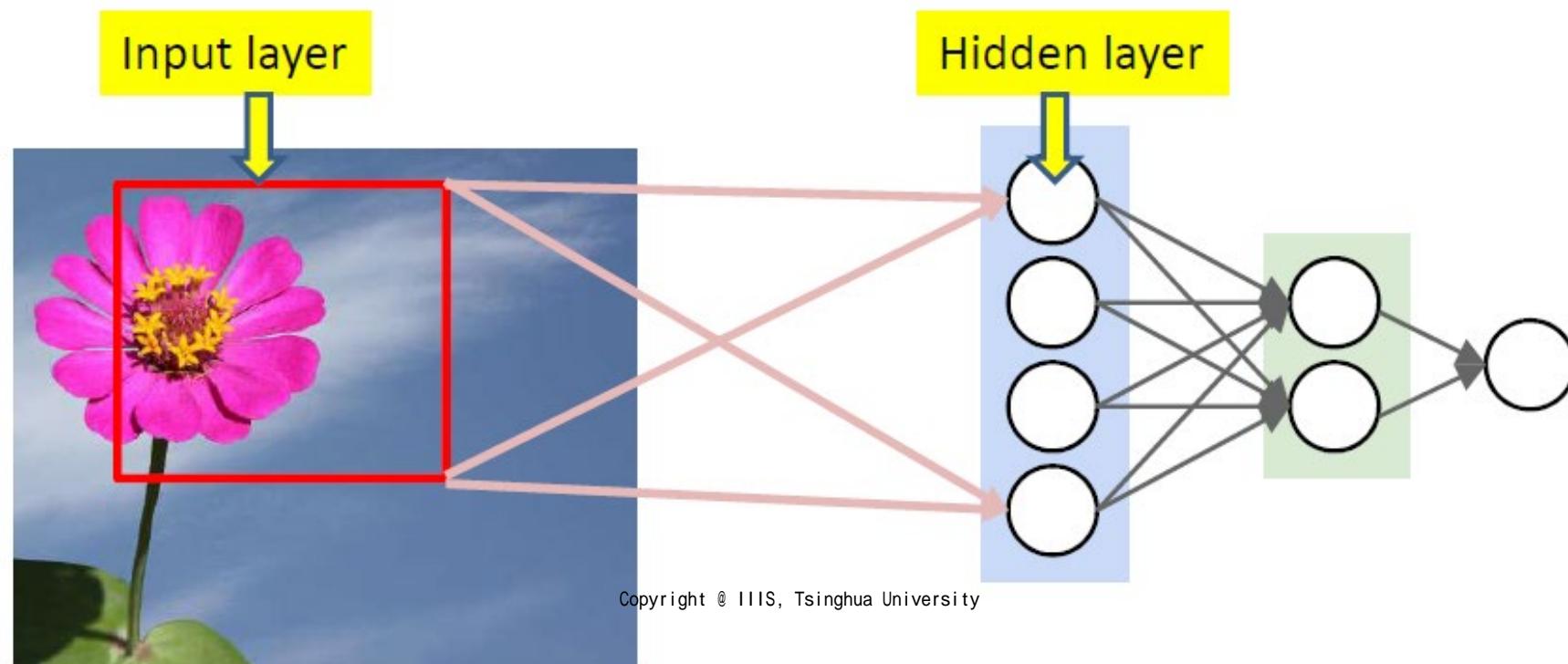
Training the Network

- Still Backpropagation!
 - Let S denotes the edges that have common value
 - $\nabla_S L(w) = \sum_{e \in S} \nabla_{w_e} L(w)$ *the effect can be summed up*
 - Your homework 😊



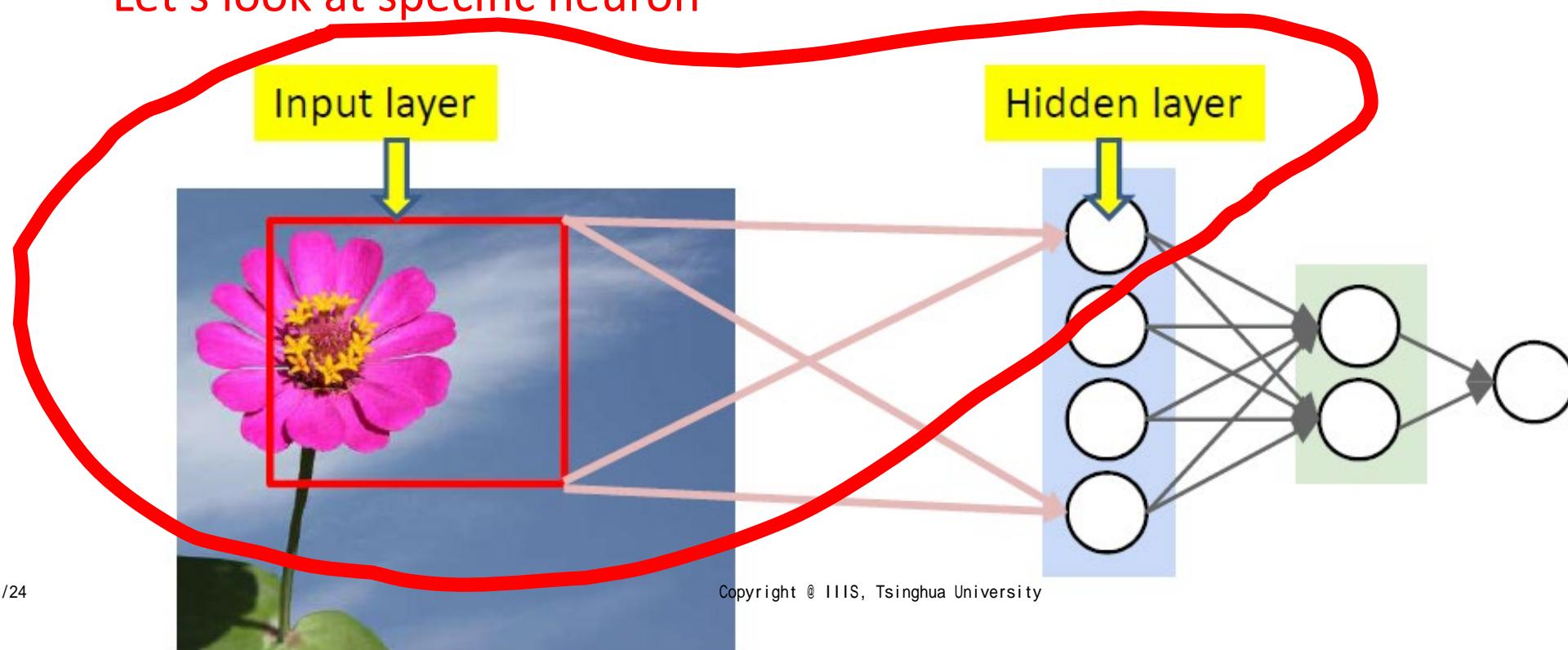
A Closer Look at 2D Scanning

- We scan the whole image for a desired pattern
- At each location, the patch is sent to an MLP



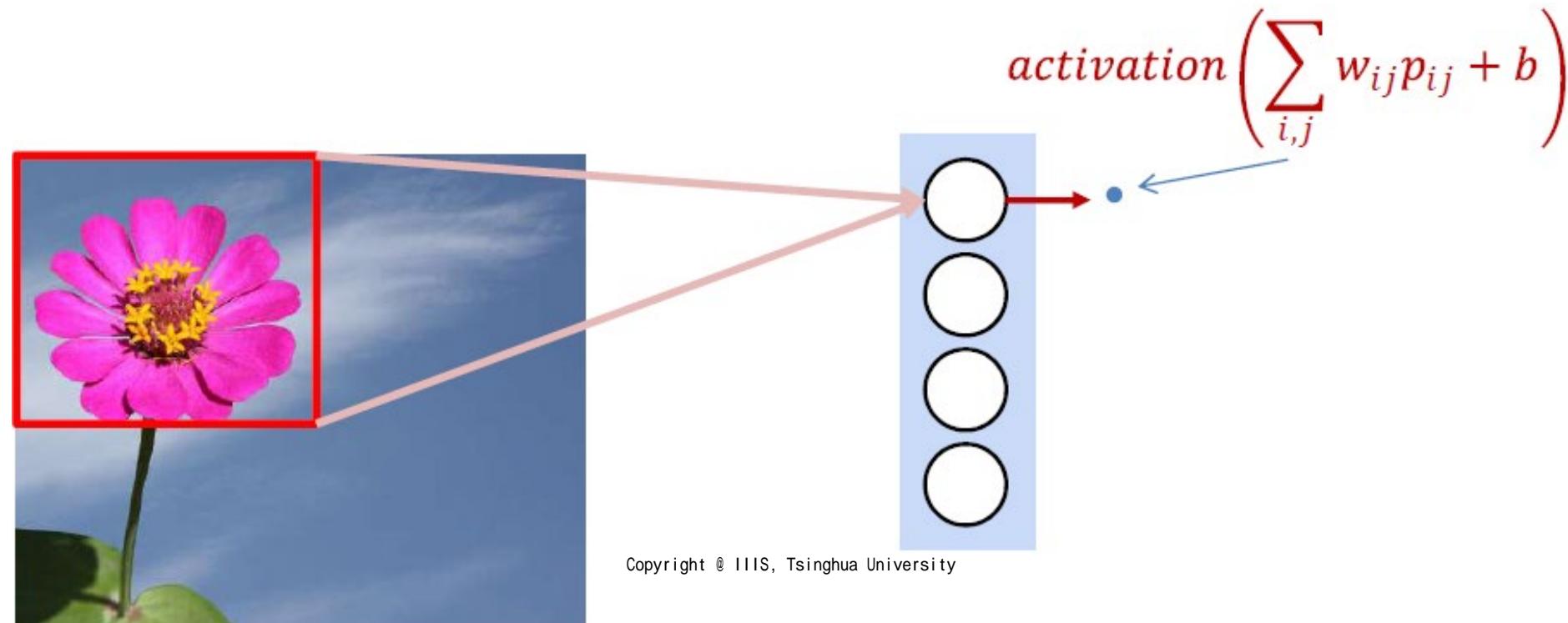
A Closer Look at 2D Scanning

- We scan the whole image for a desired pattern
- At each location, the patch is sent to an MLP
 - Let's look at specific neuron



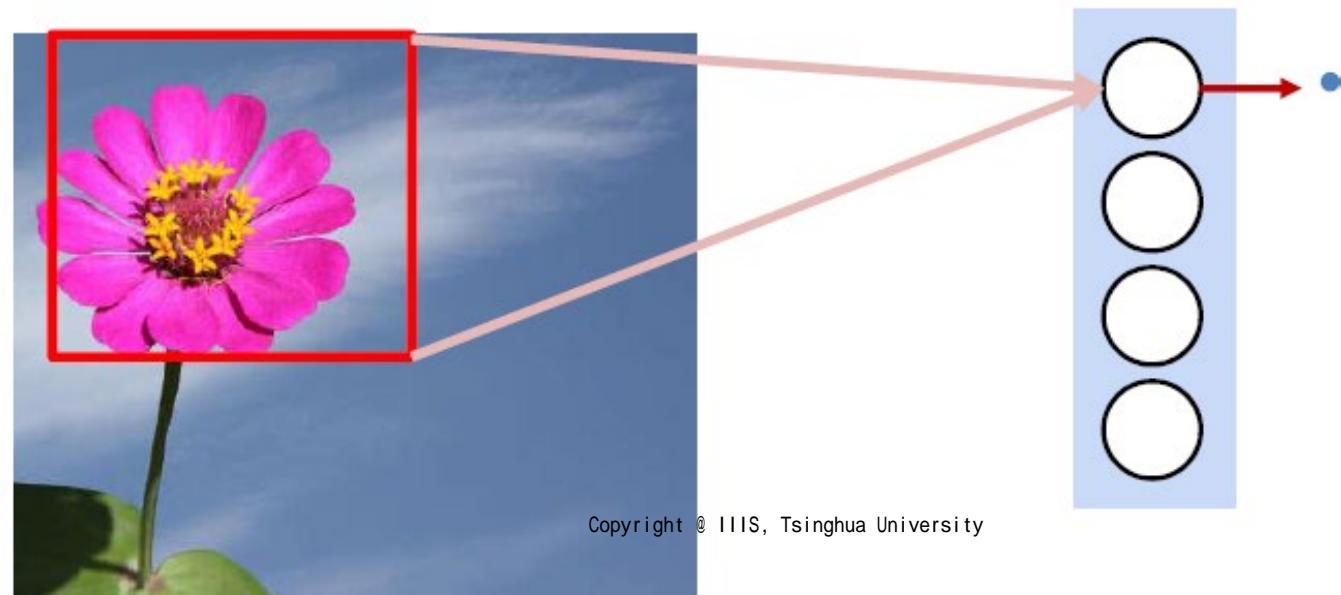
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



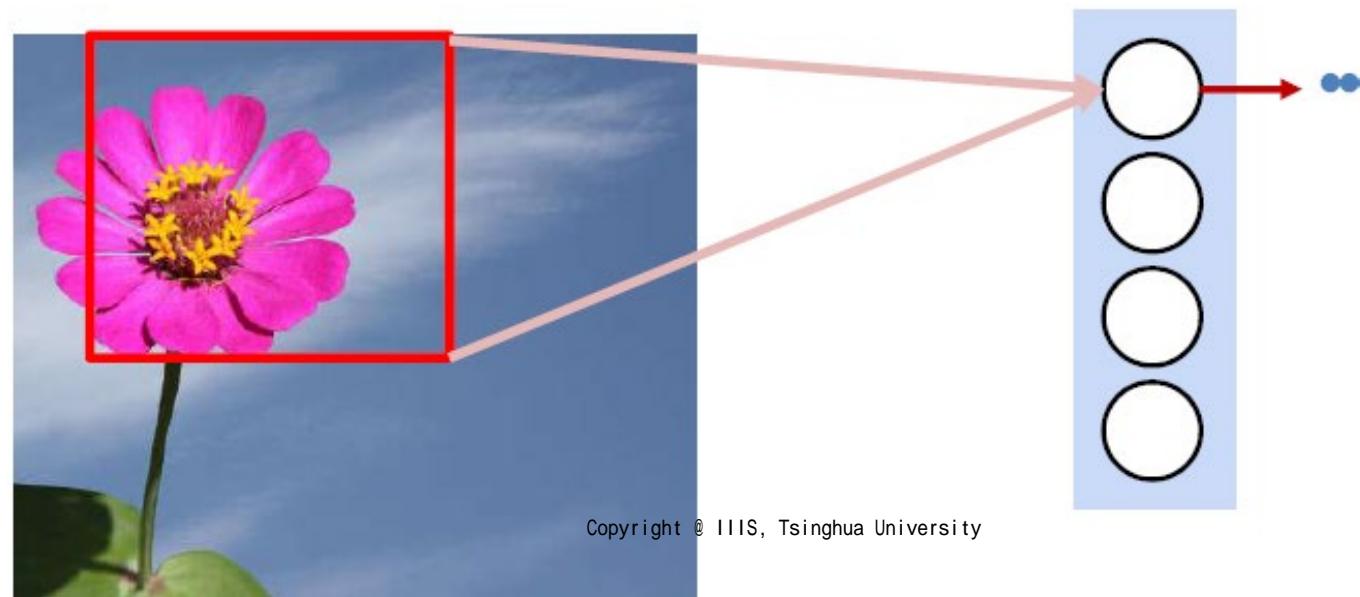
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



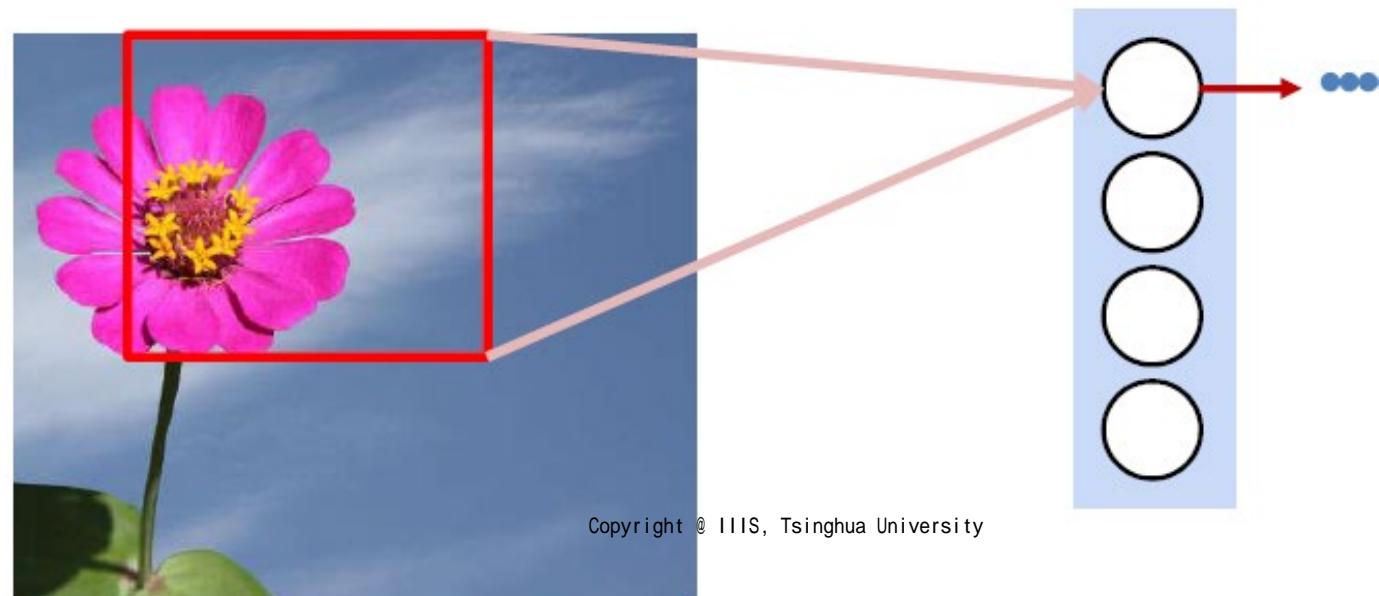
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



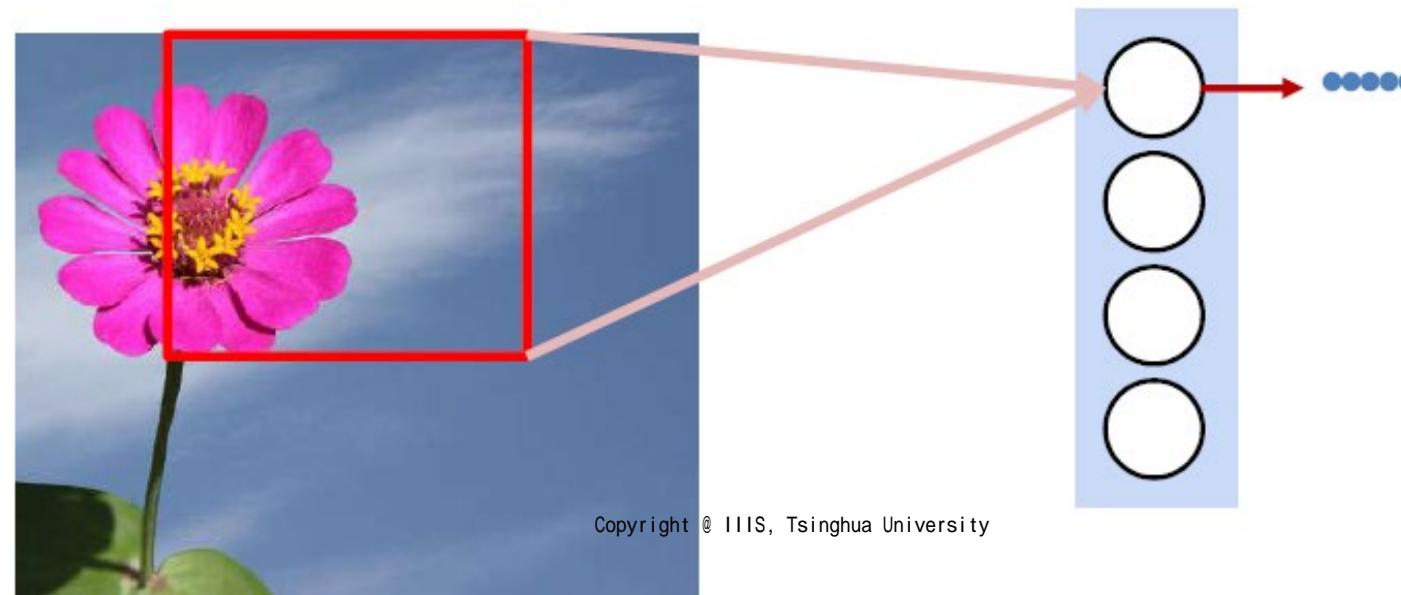
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



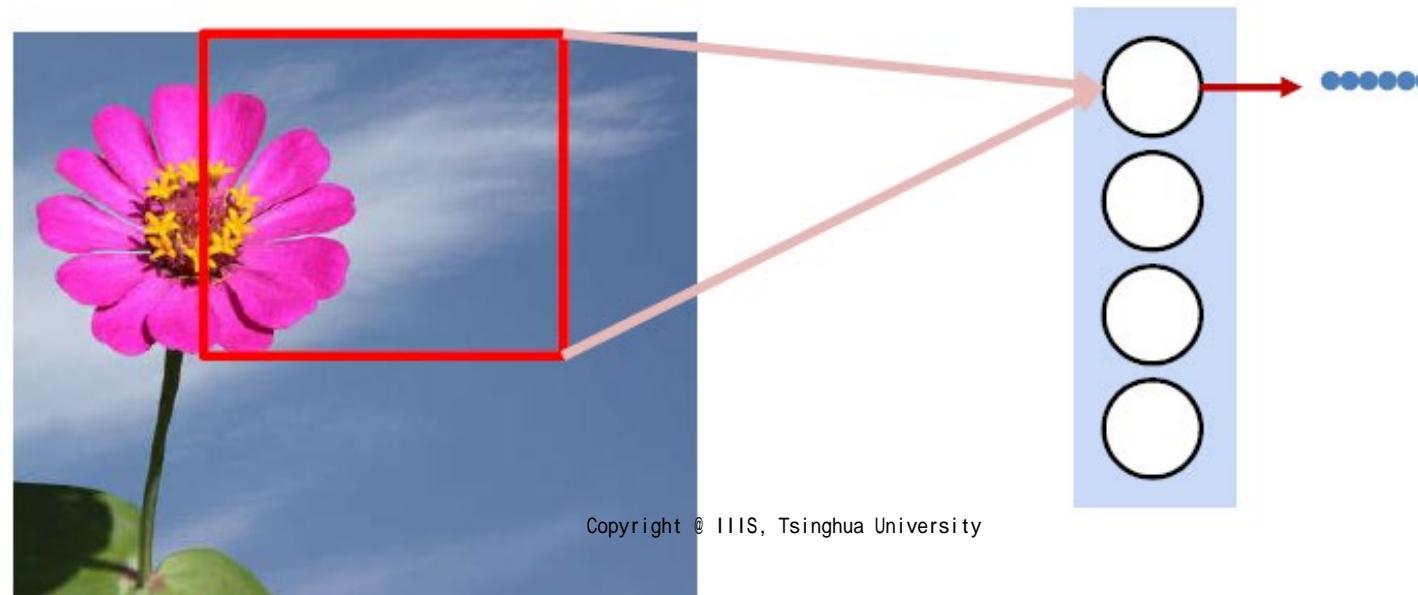
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



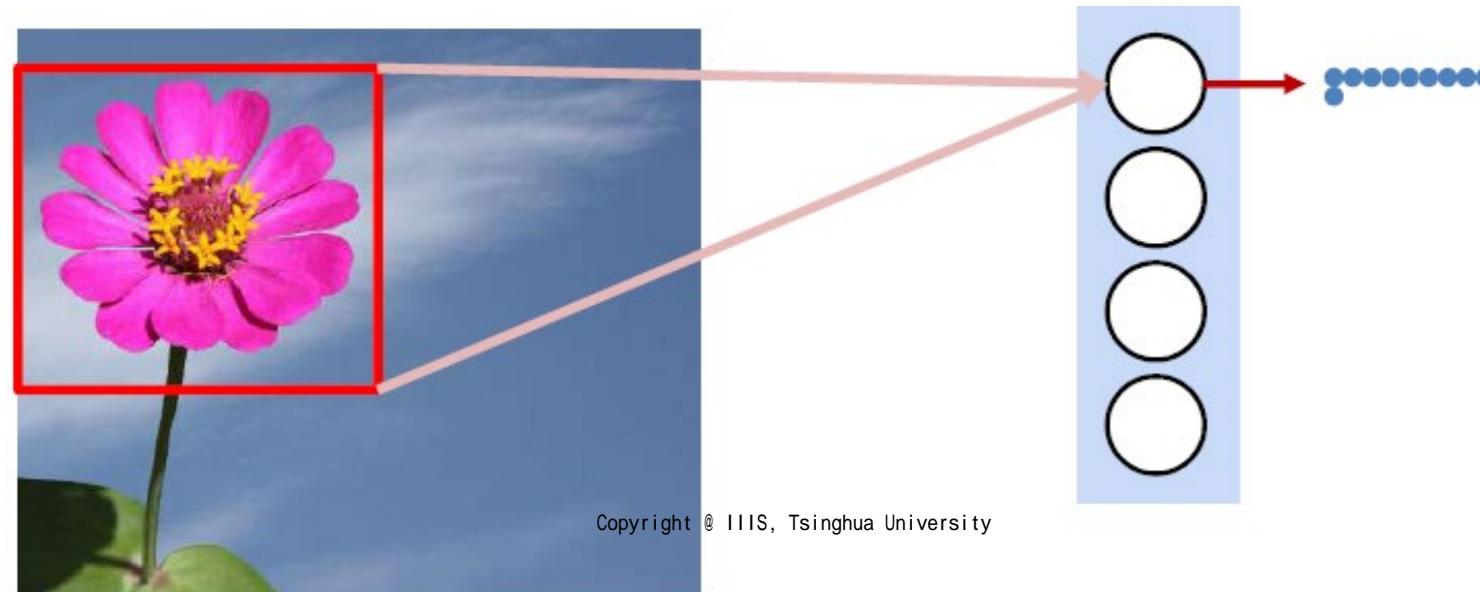
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



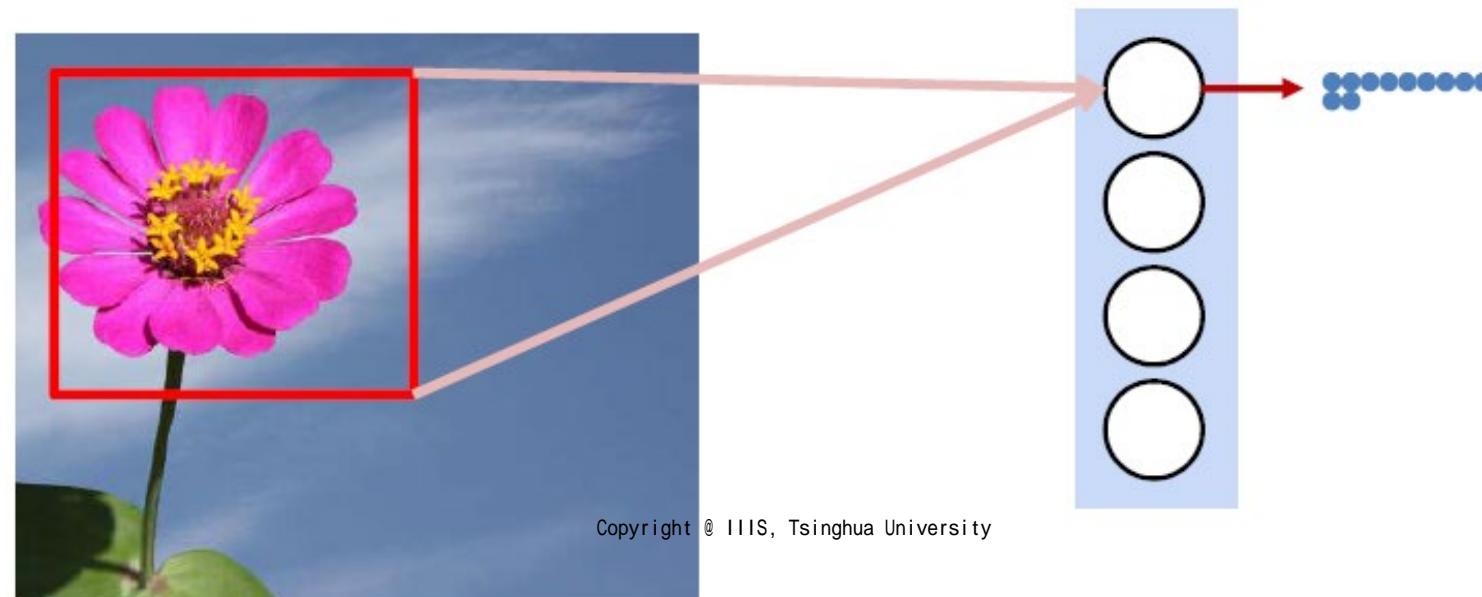
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



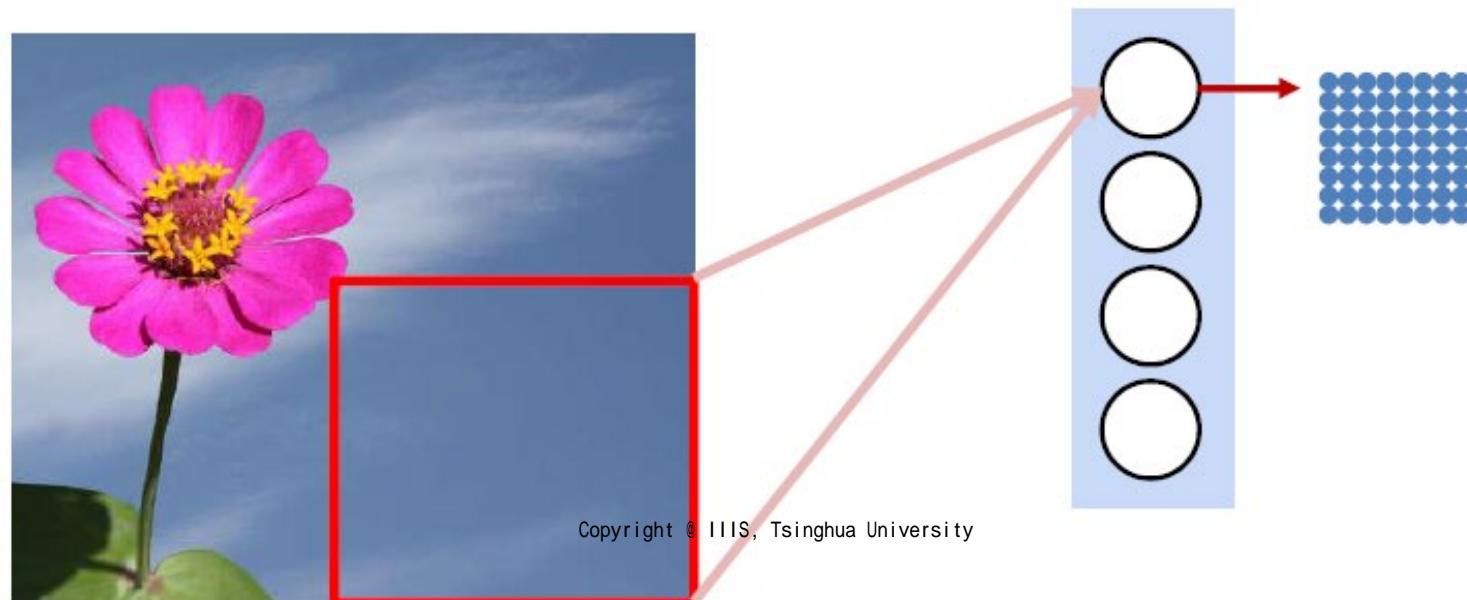
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations



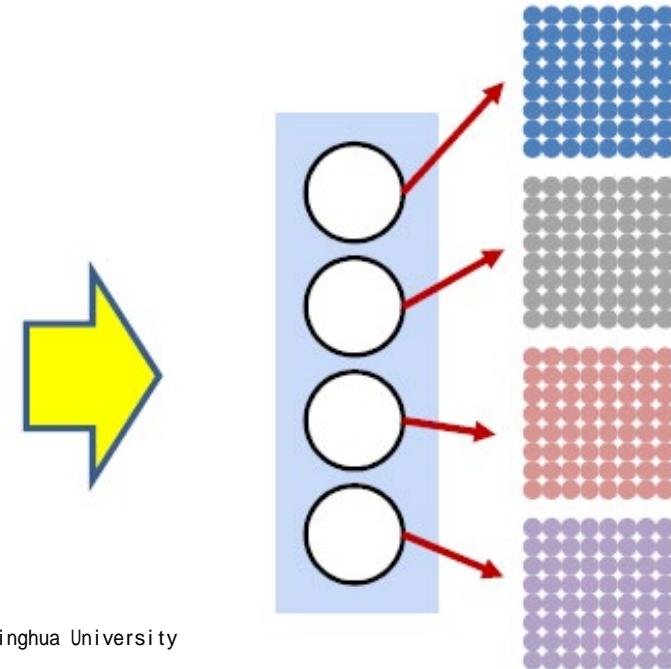
A Closer Look at 2D Scanning

- Let's consider a single neuron (a simple perceptron)
 - We can arrange the neuron outputs corresponding to the image locations
 - **We obtain a rectangle outputs!**



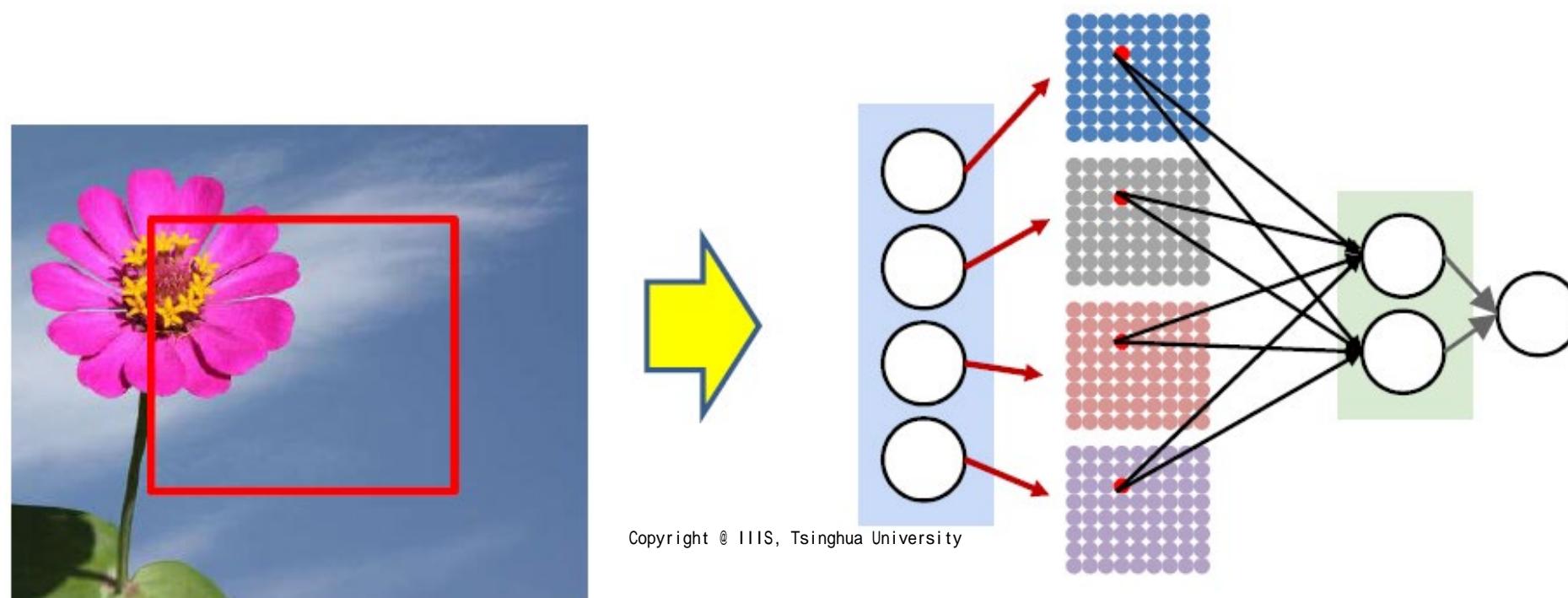
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly



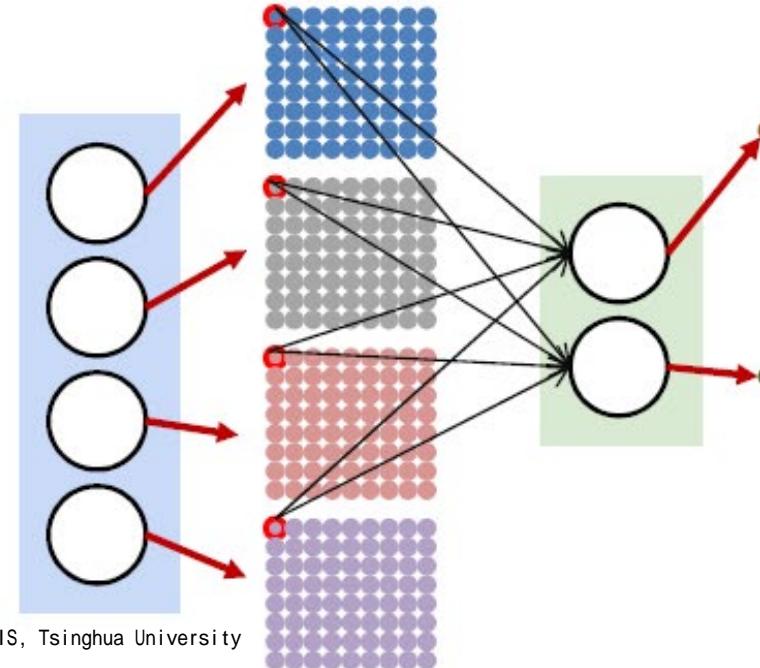
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification



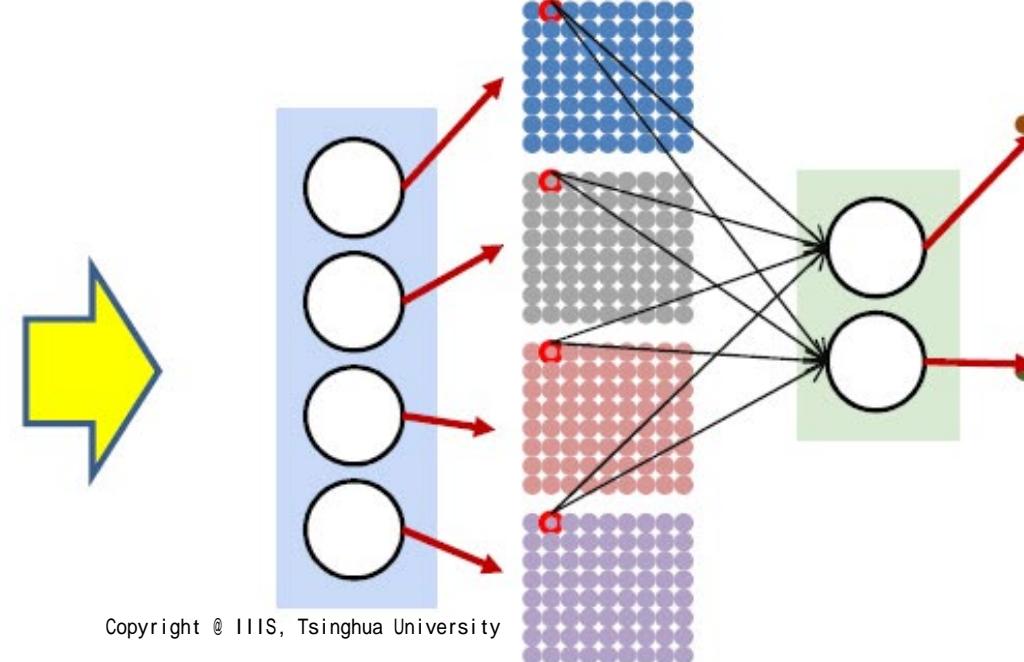
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



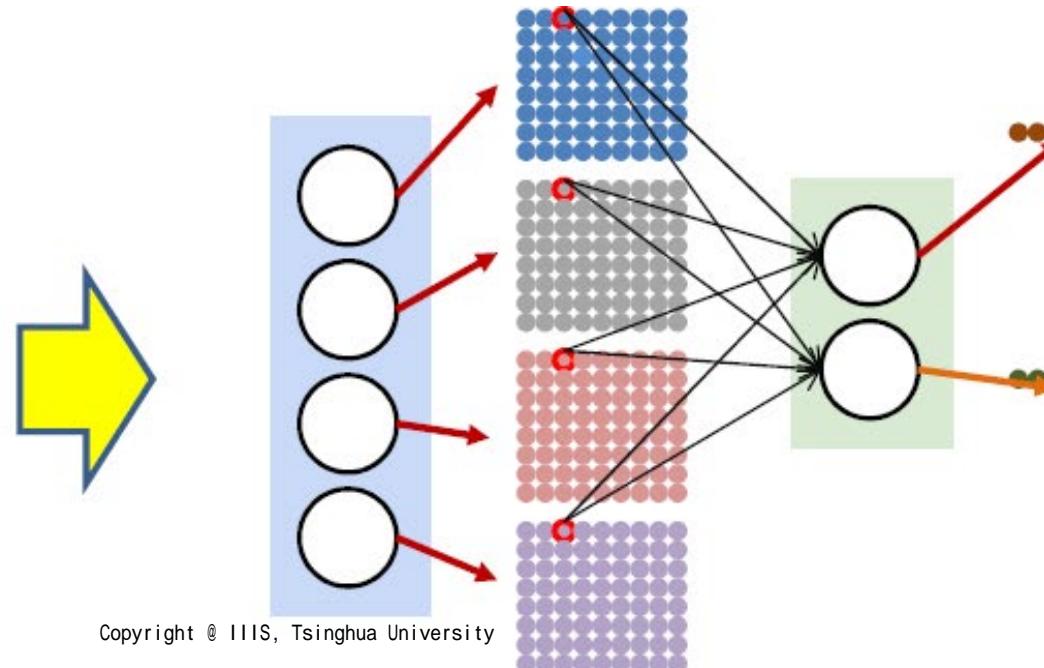
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



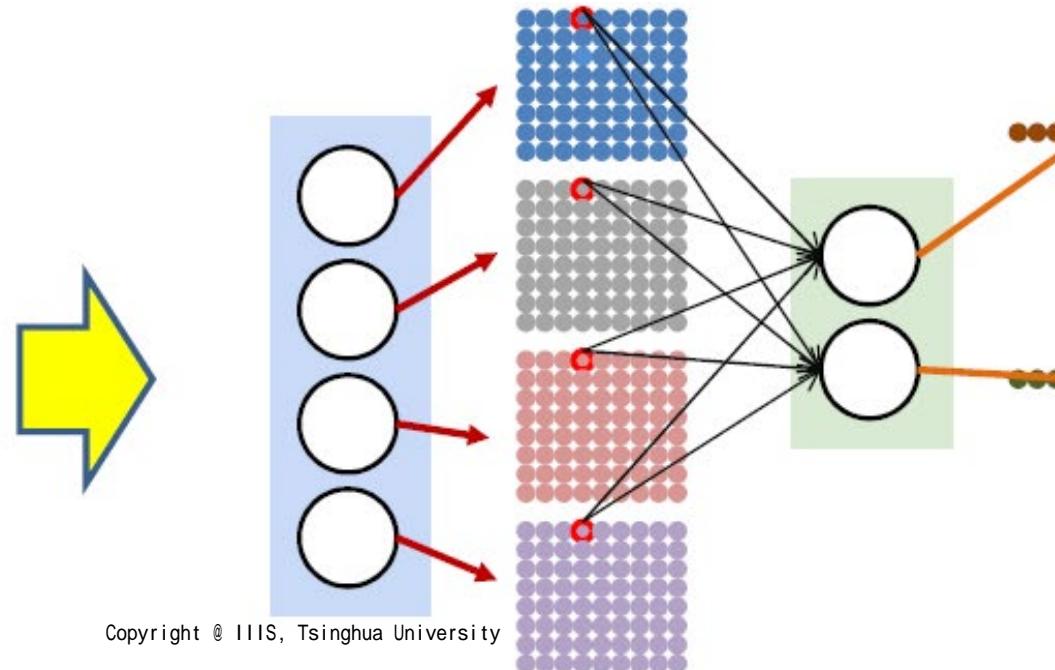
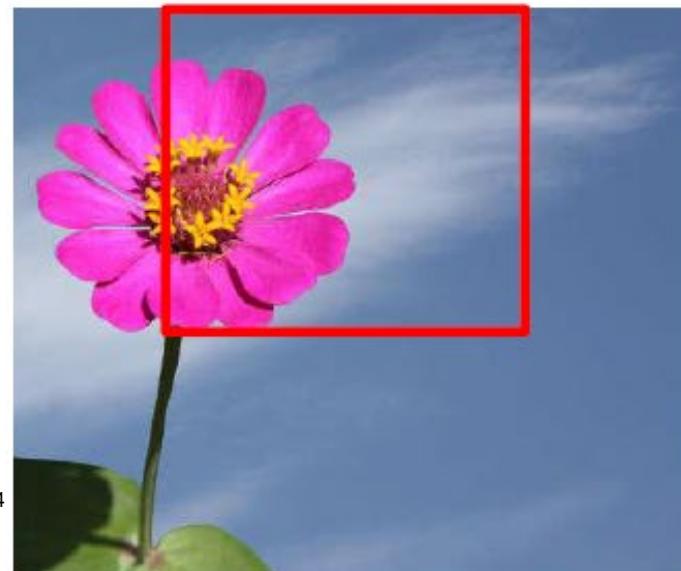
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



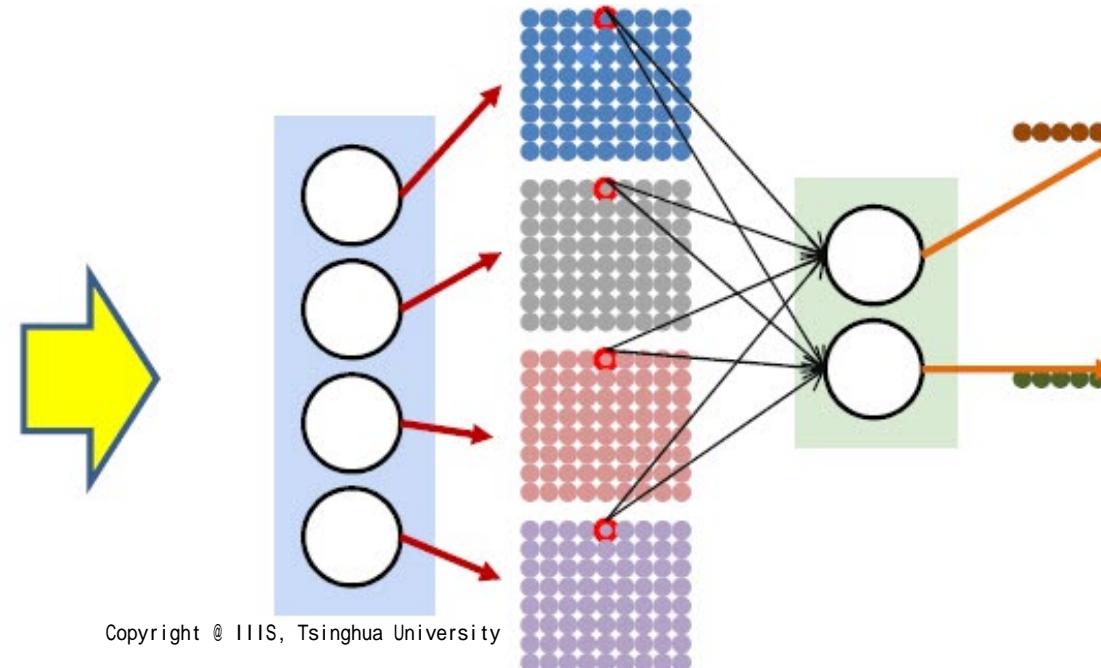
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



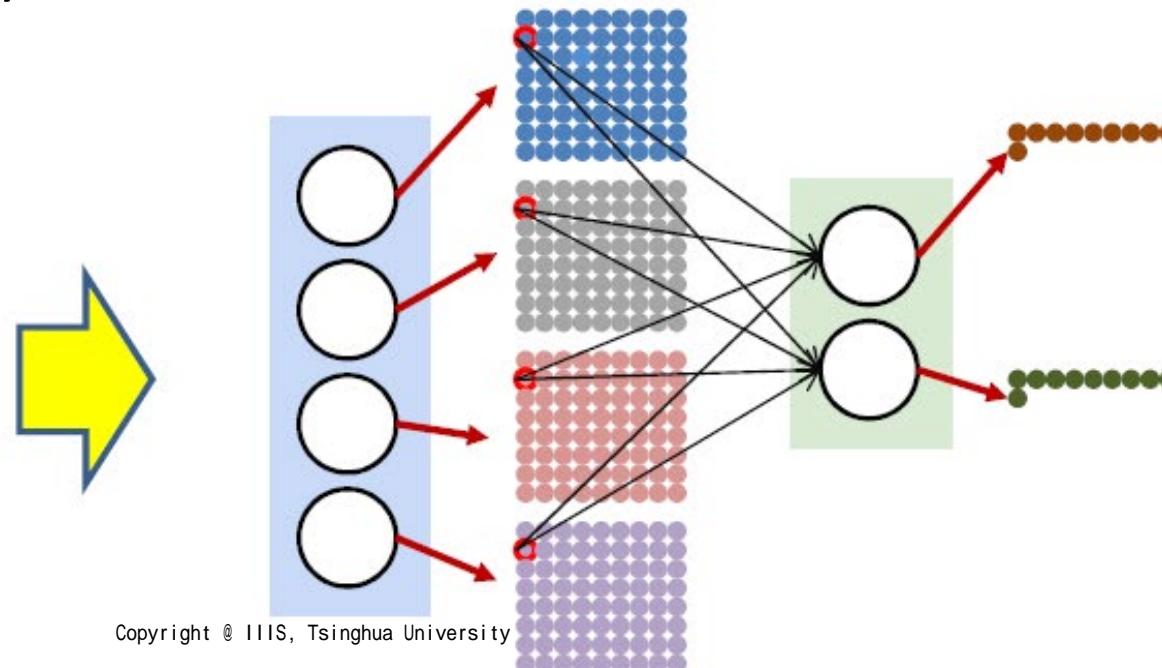
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



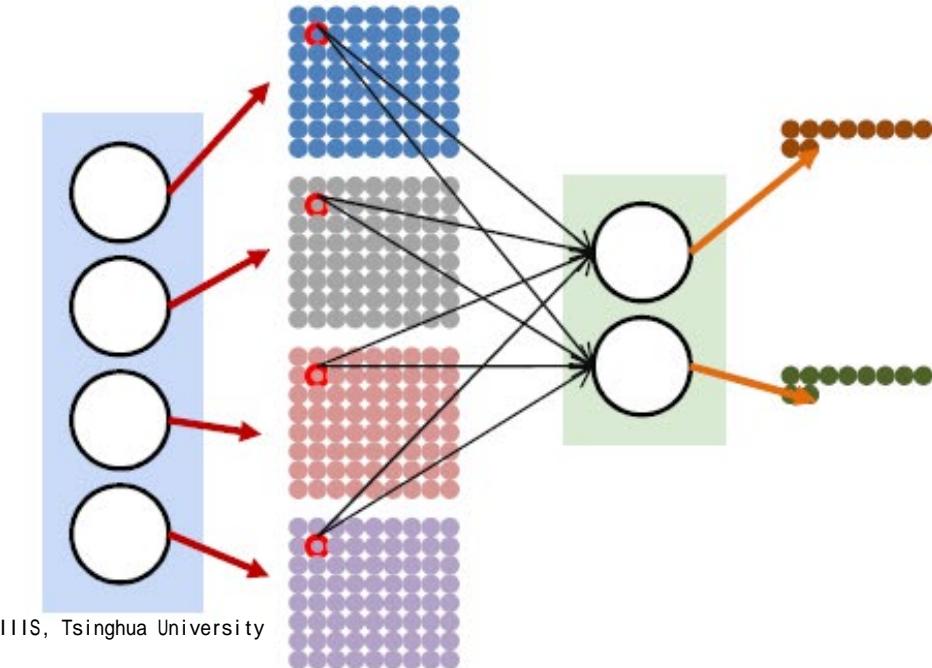
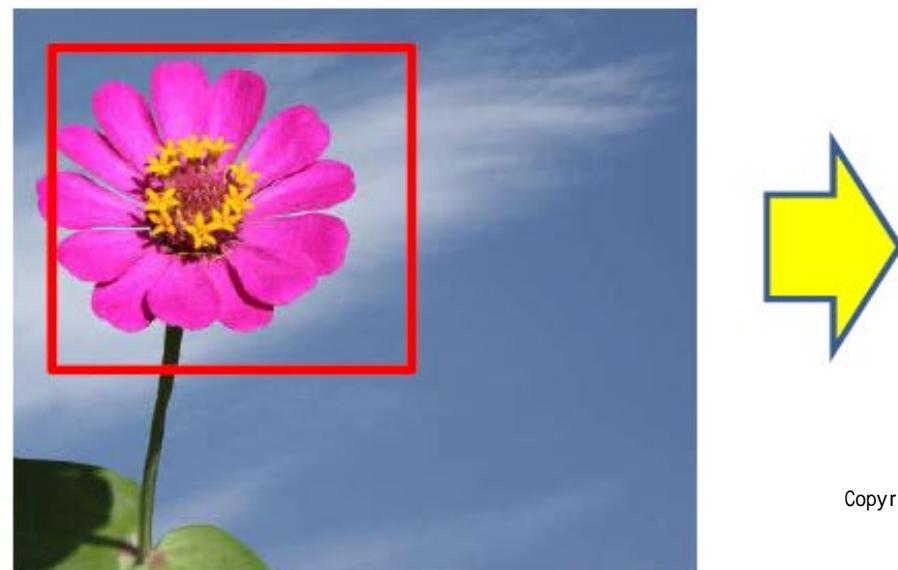
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



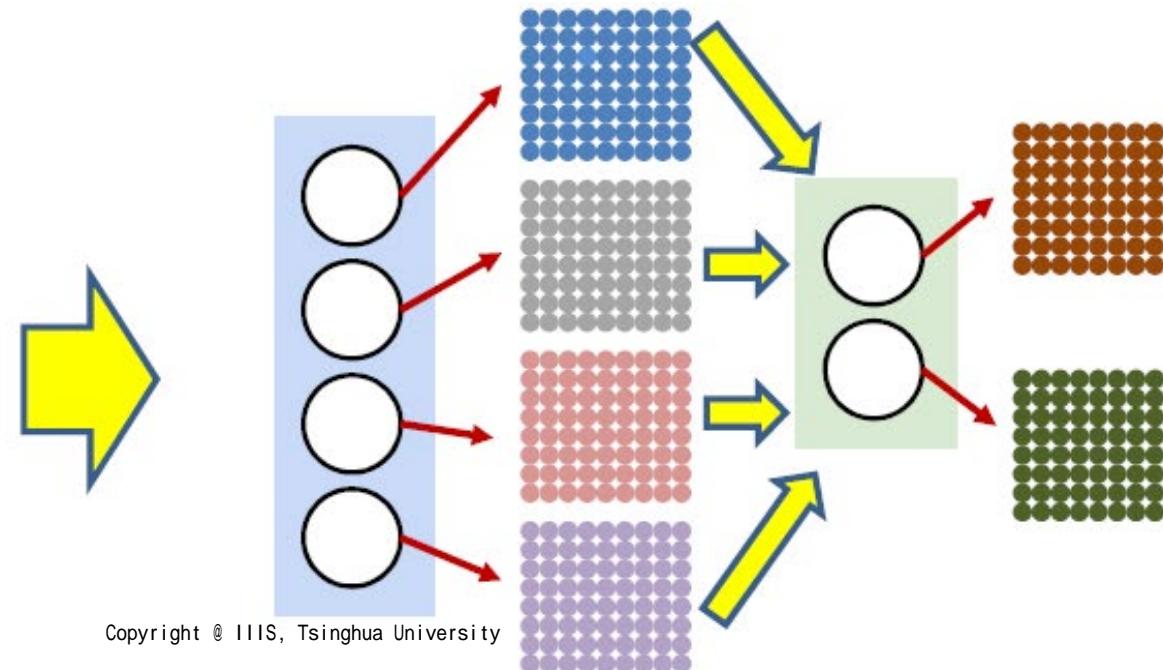
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer



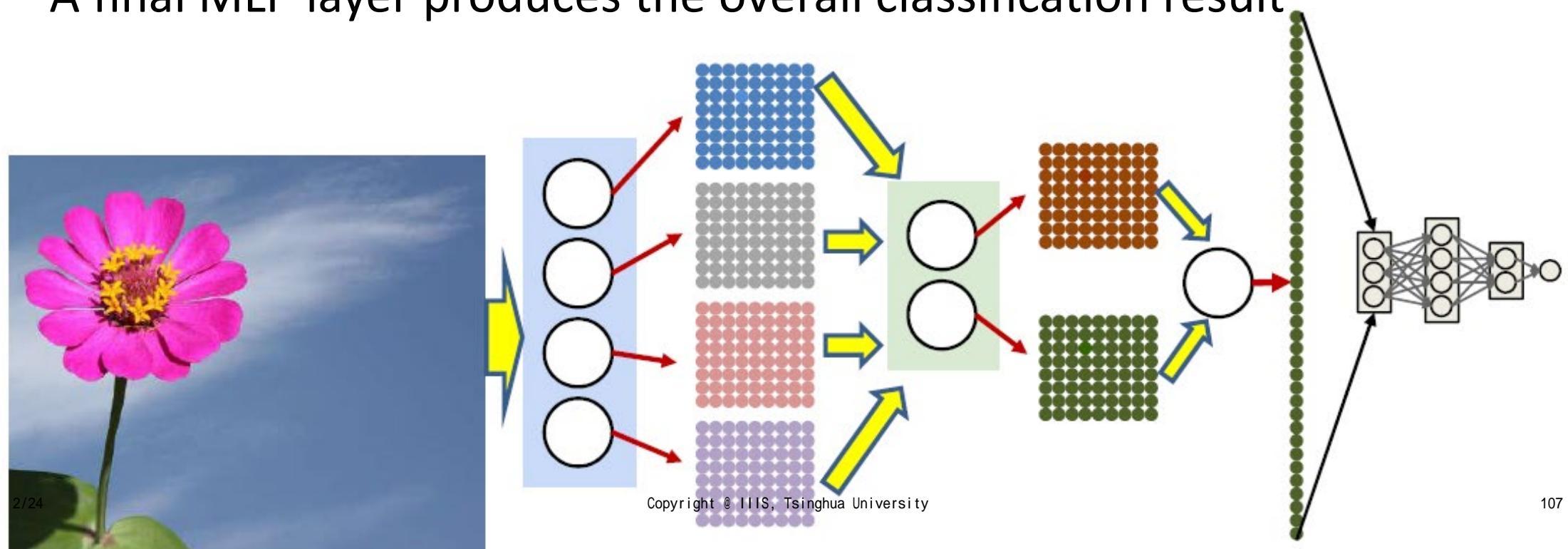
A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- And we send the neuron outputs to the second layer for classification
 - Each output location of the second layer takes the input from the same location from the first layer

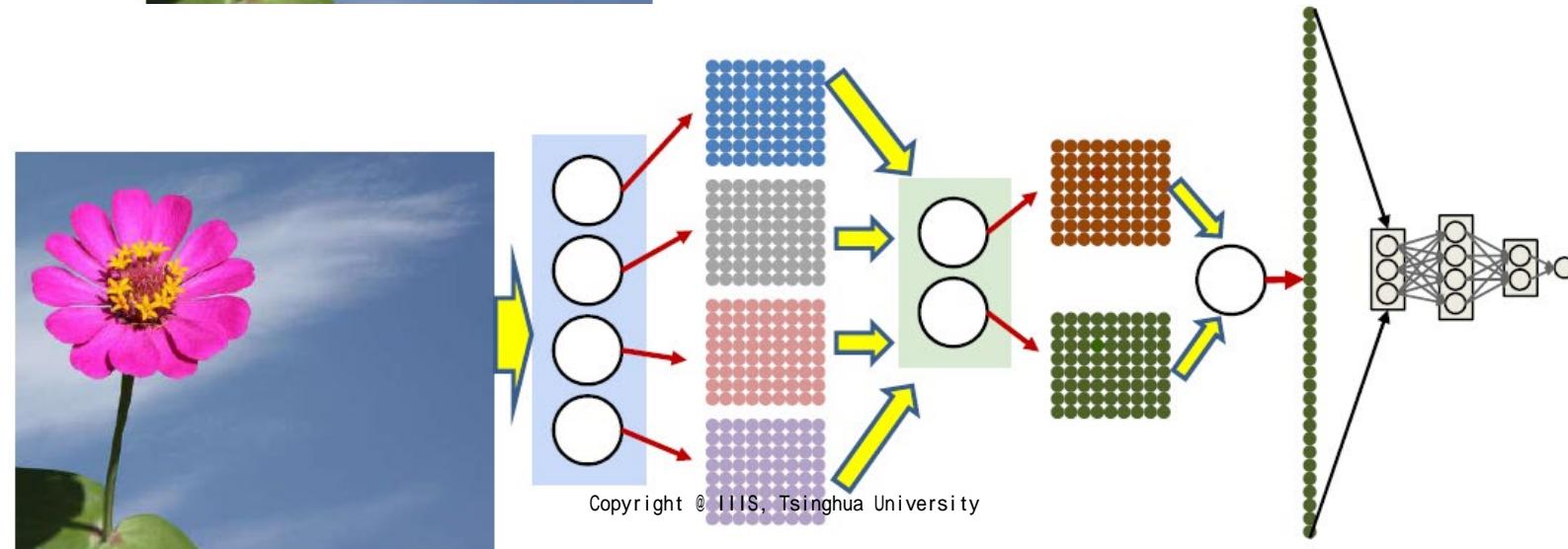
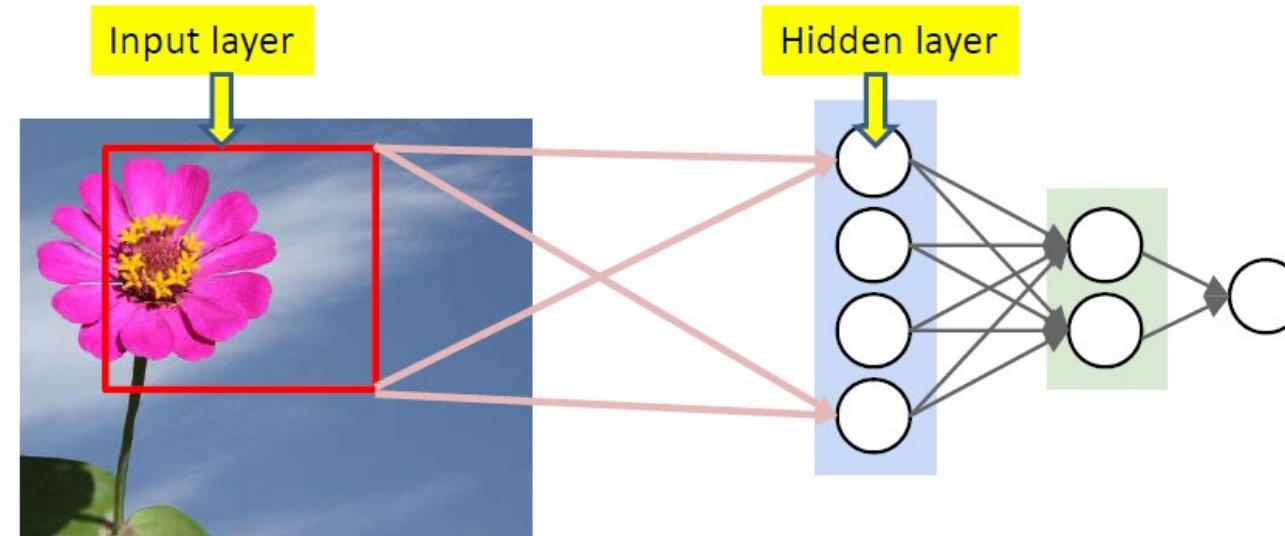


A Closer Look at 2D Scanning

- The output for each neuron can be organized as a rectangle similarly
- For each location, the outputs will be passed towards the final layer
- A final MLP layer produces the overall classification result

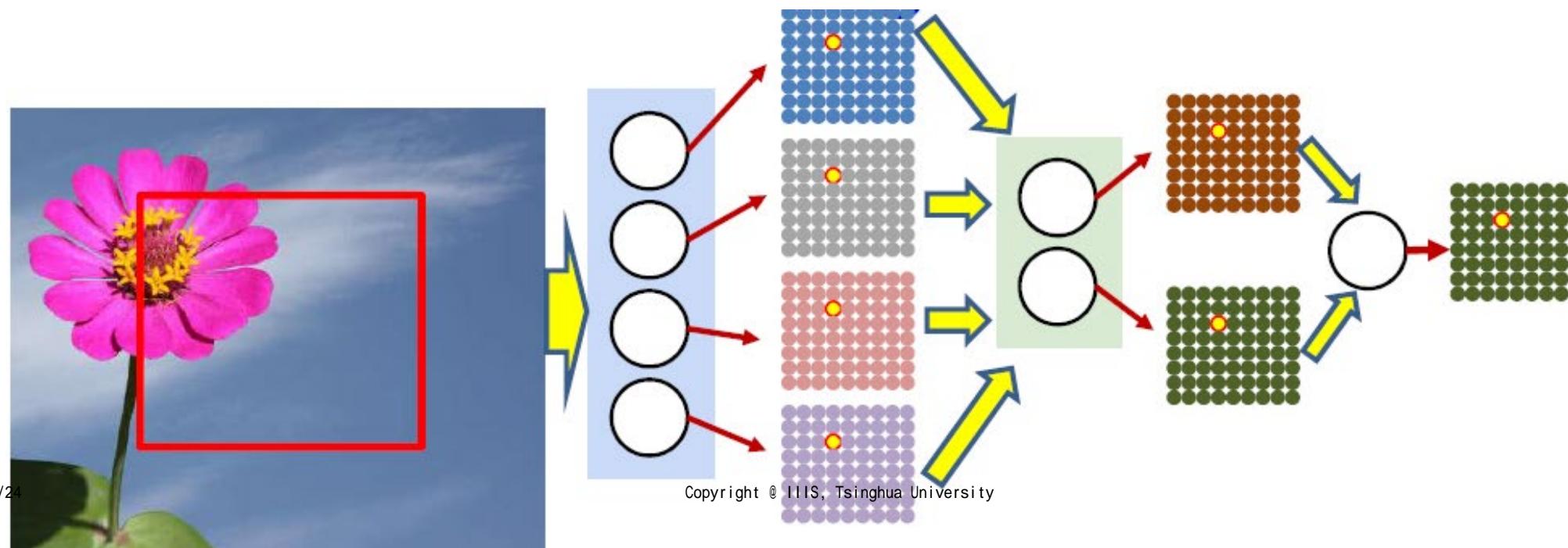


2D Scanning: a Spatial View



2D Scanning: a Spatial View

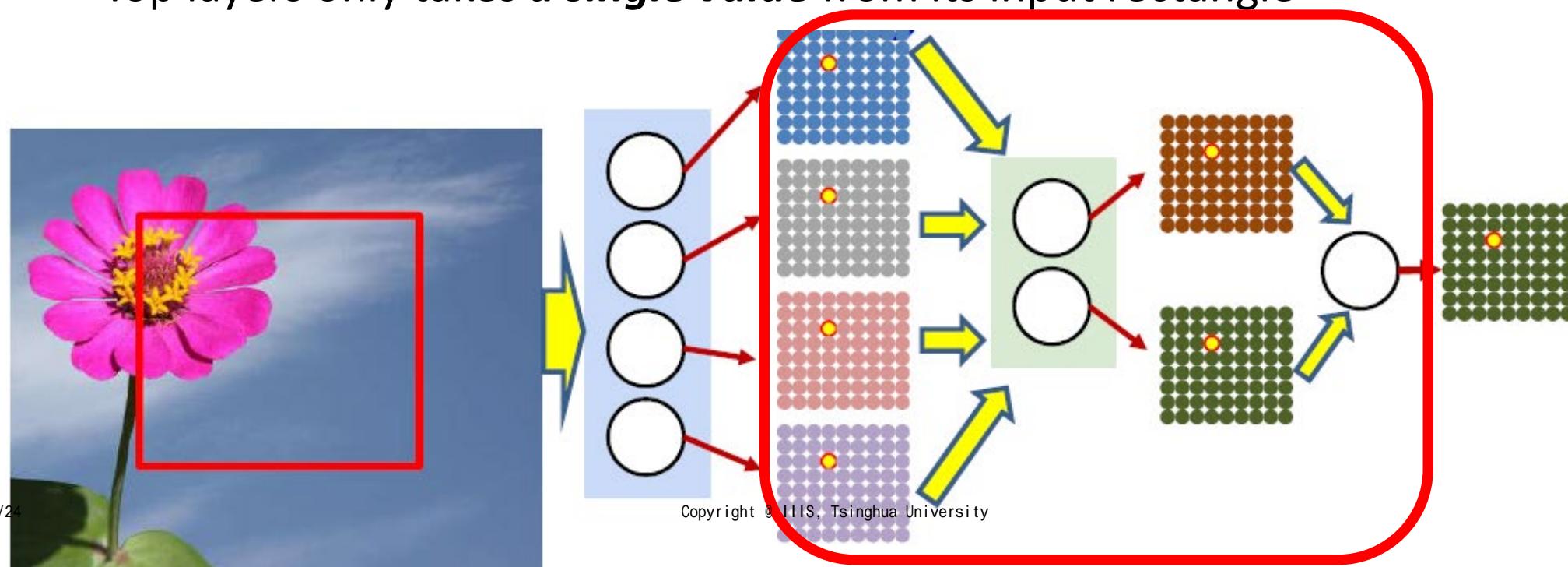
- Each position in the output neuron map corresponds to a patch position in the input image



2D Scanning: a Spatial View

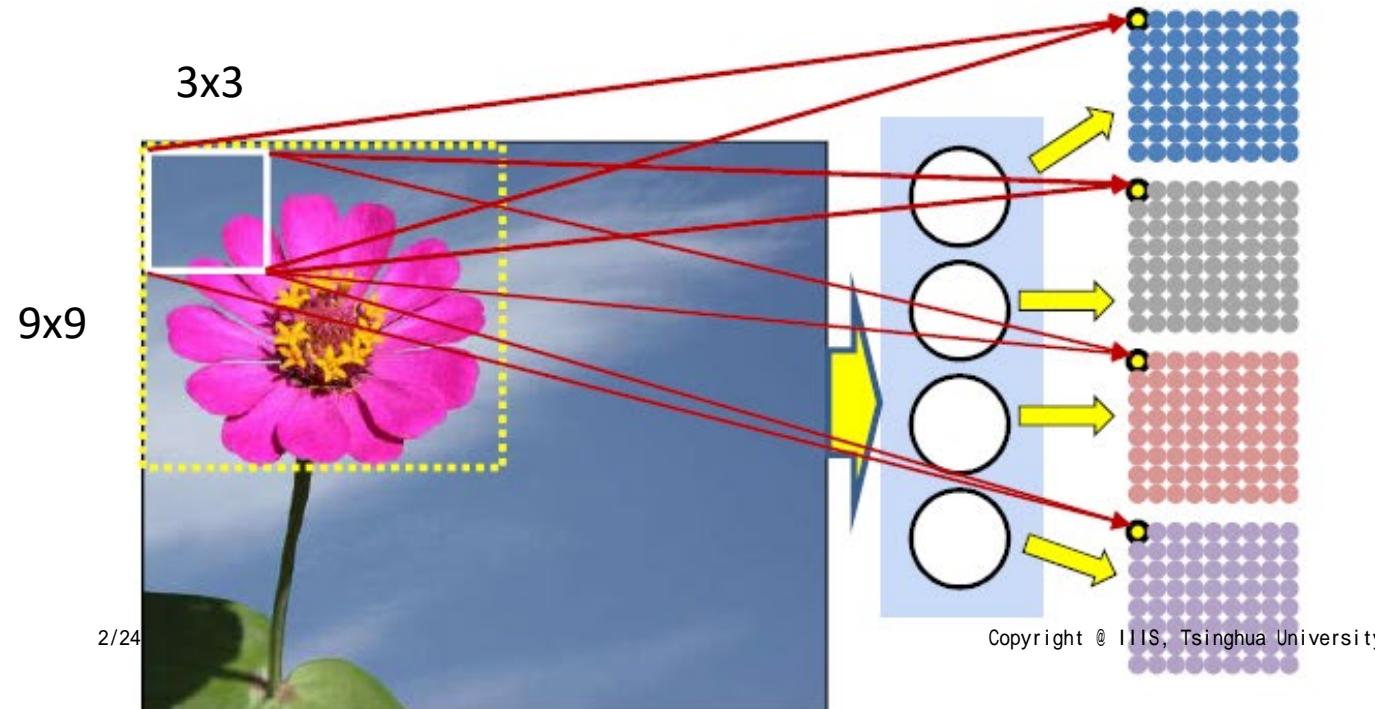
- Each position in the output neuron map corresponds to a patch position in the input image
 - The first layer takes care of the *entire patch*
 - Top layers only takes *a single value* from its input rectangle

Can distributed the scanning workload across layers?



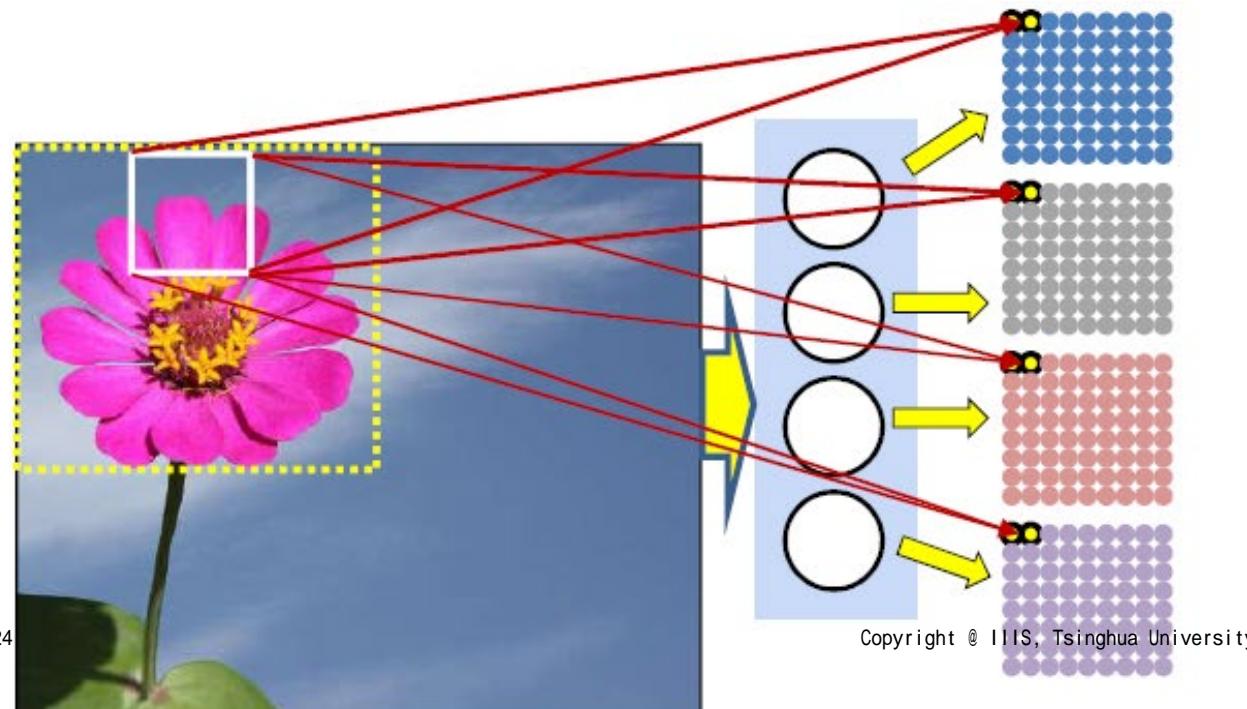
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9×9 patch scanning by 2 layers of 3×3 scanning
 - First layer for smaller patches and second layer for larger patches



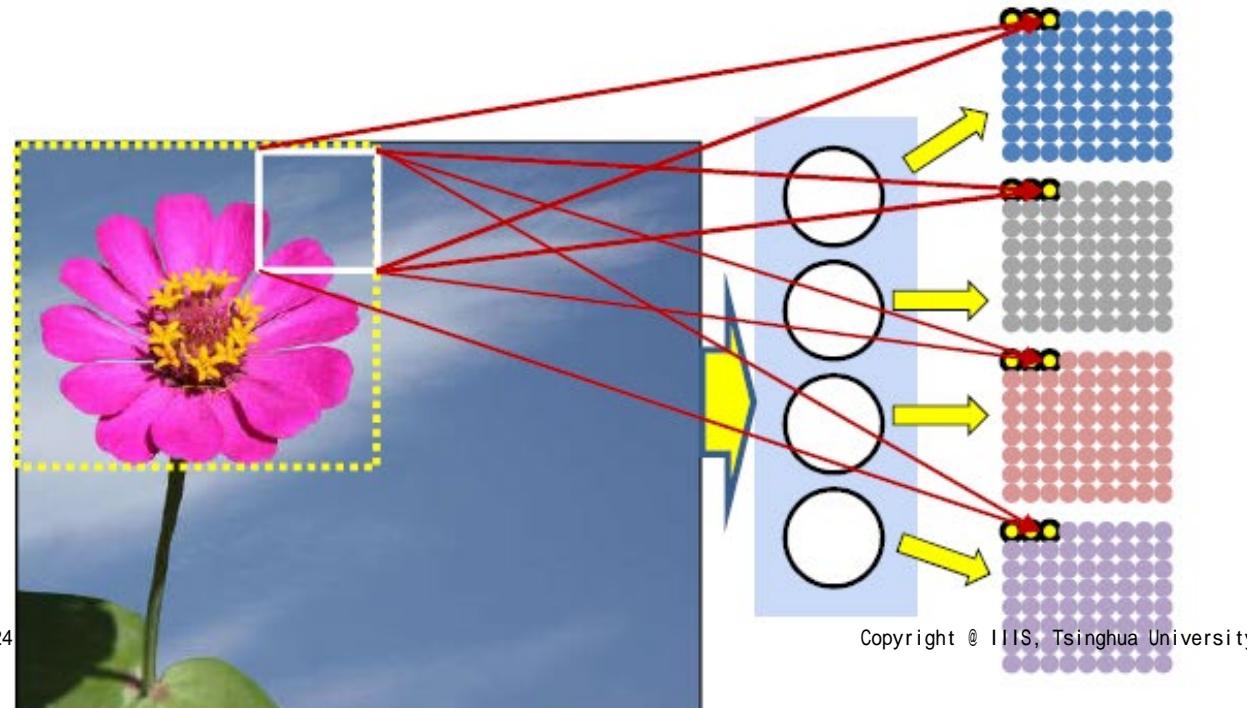
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



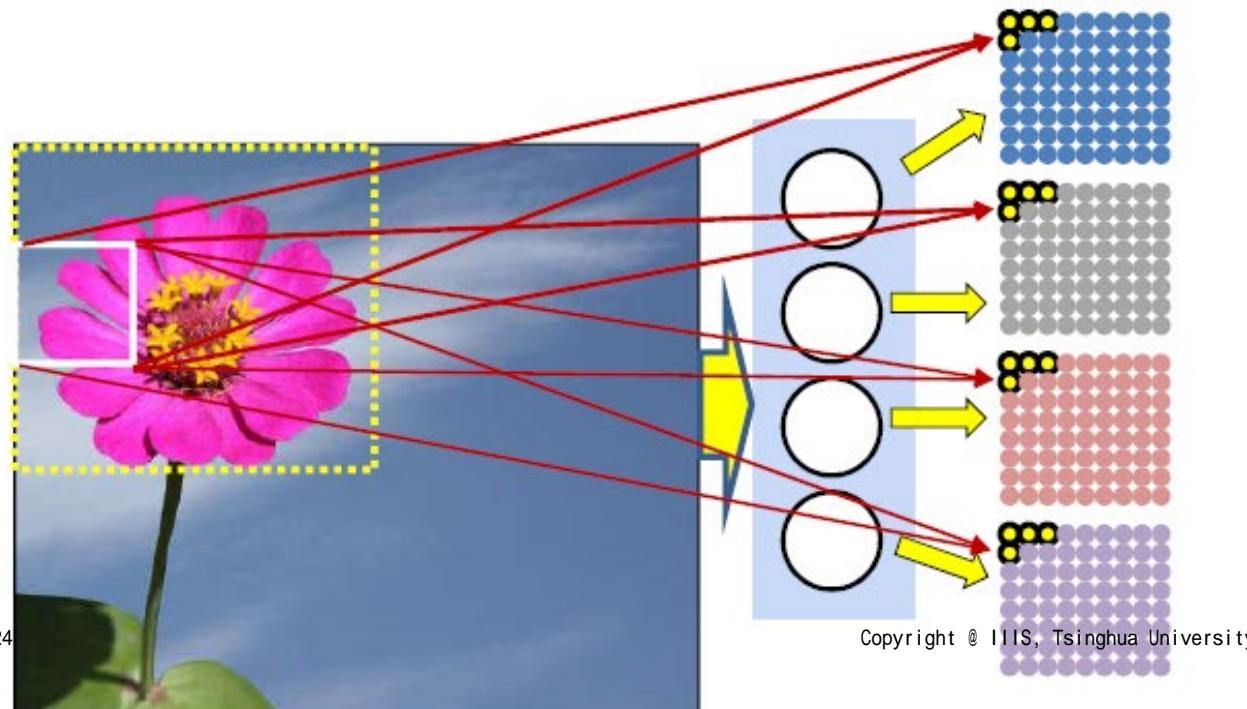
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



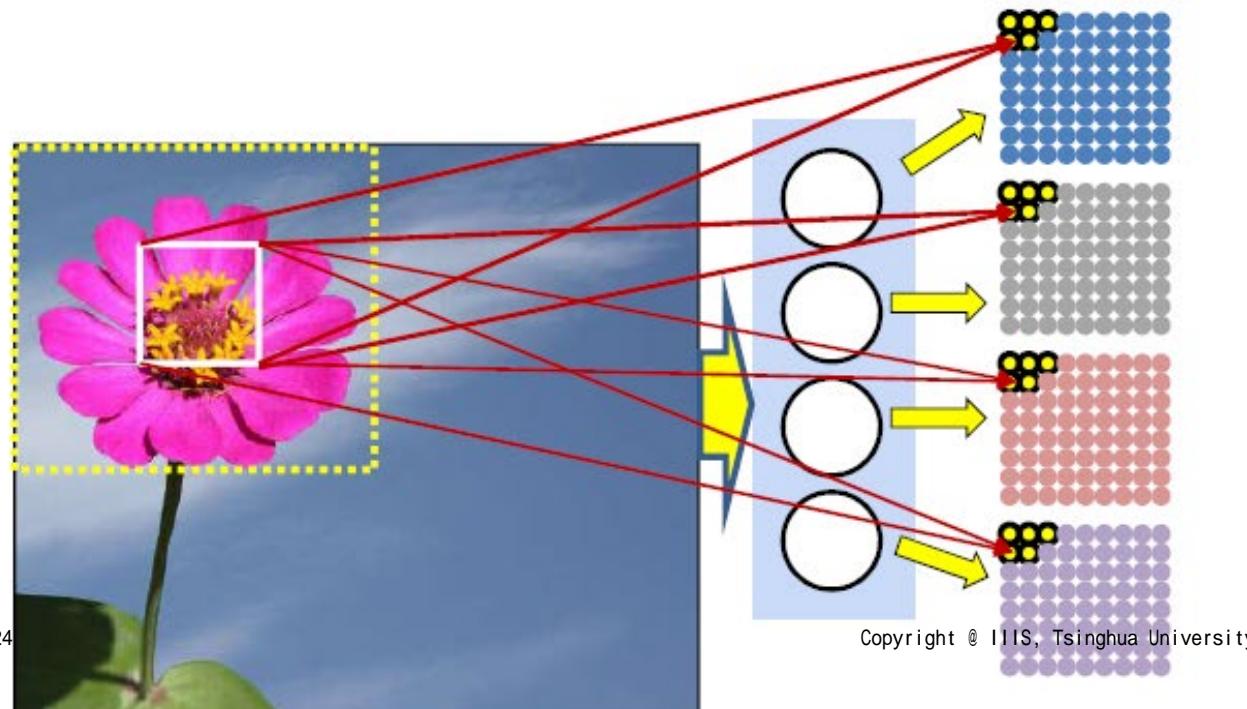
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



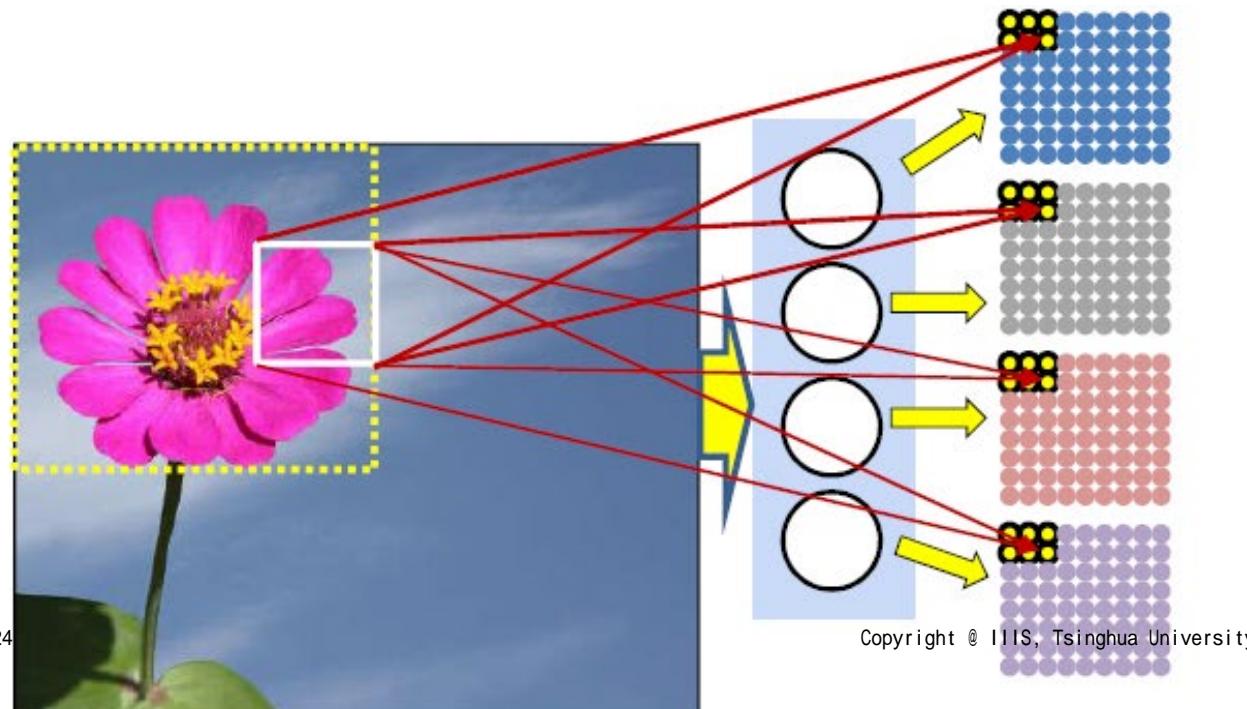
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



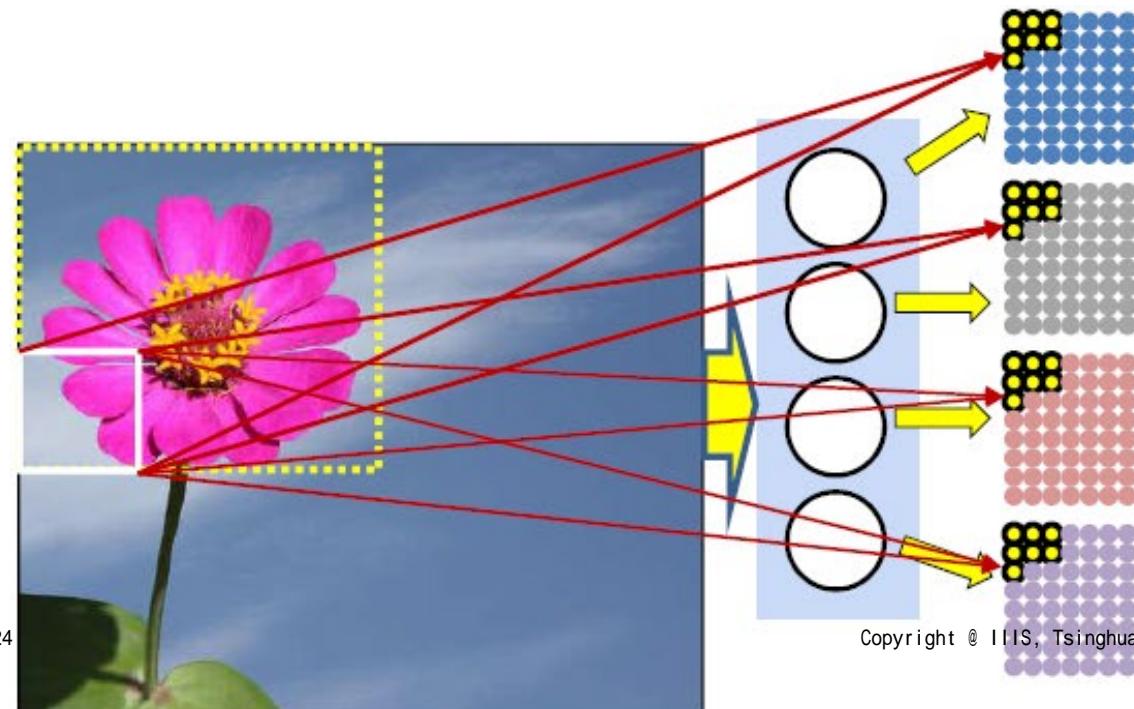
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



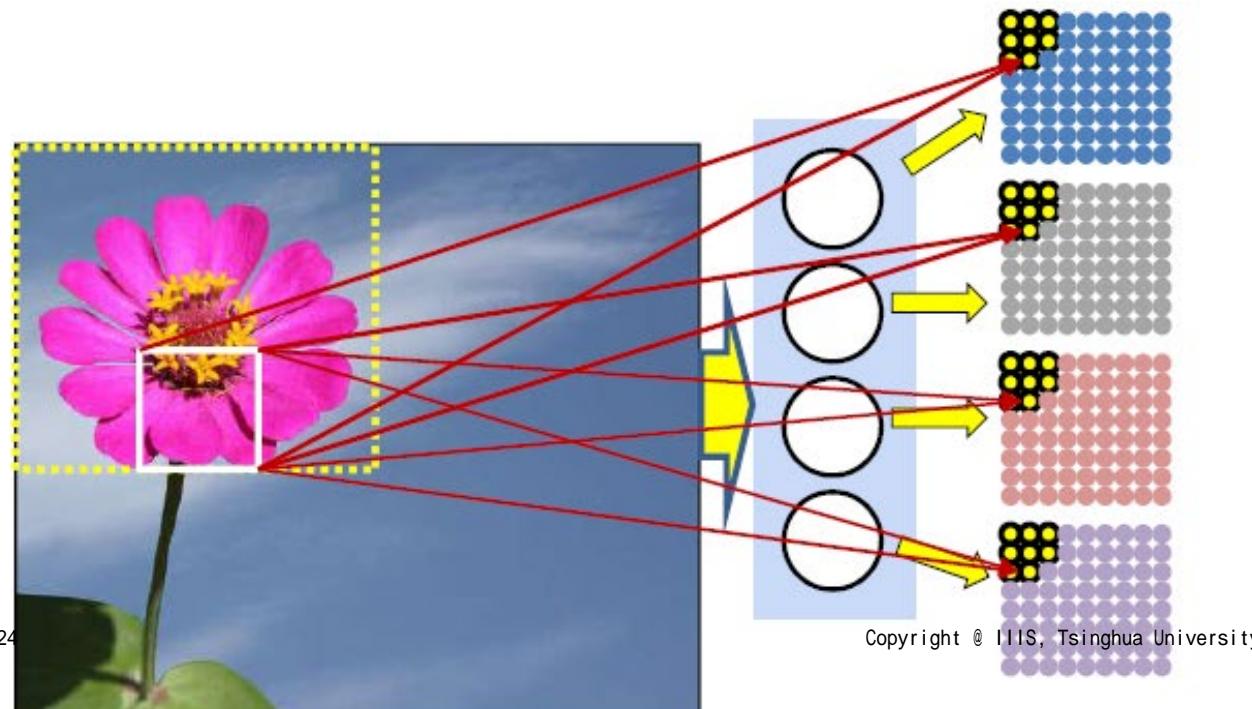
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



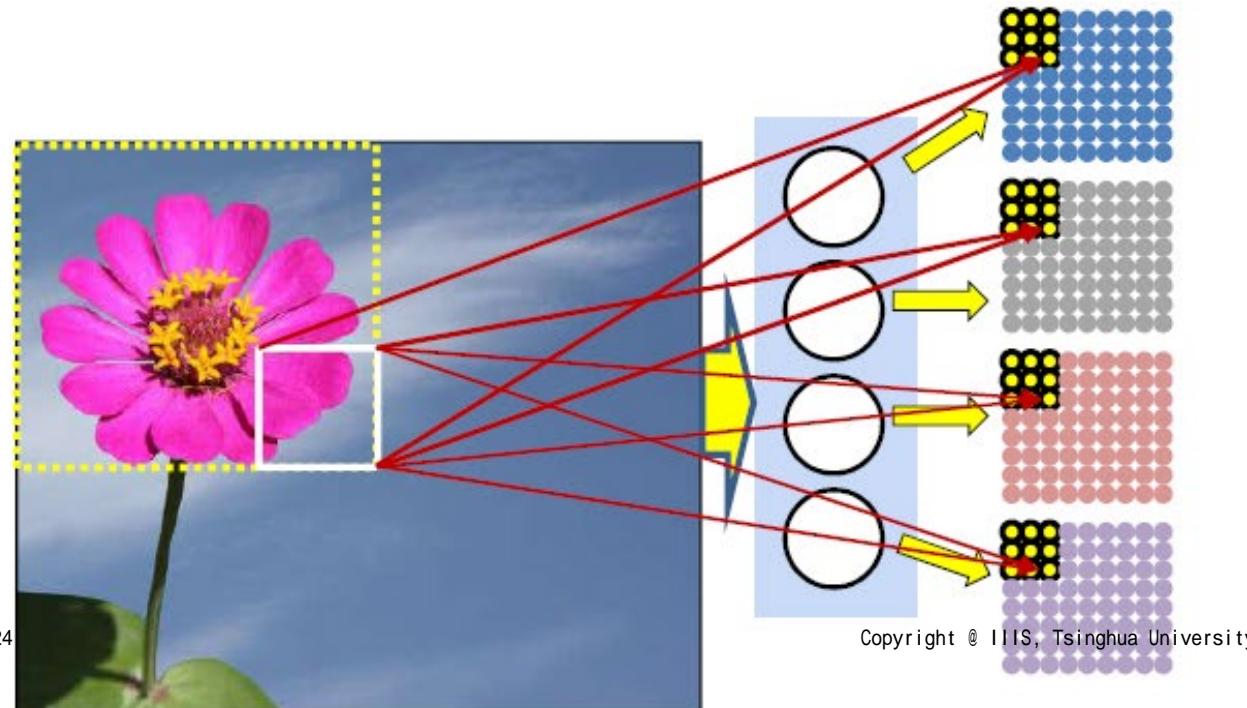
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



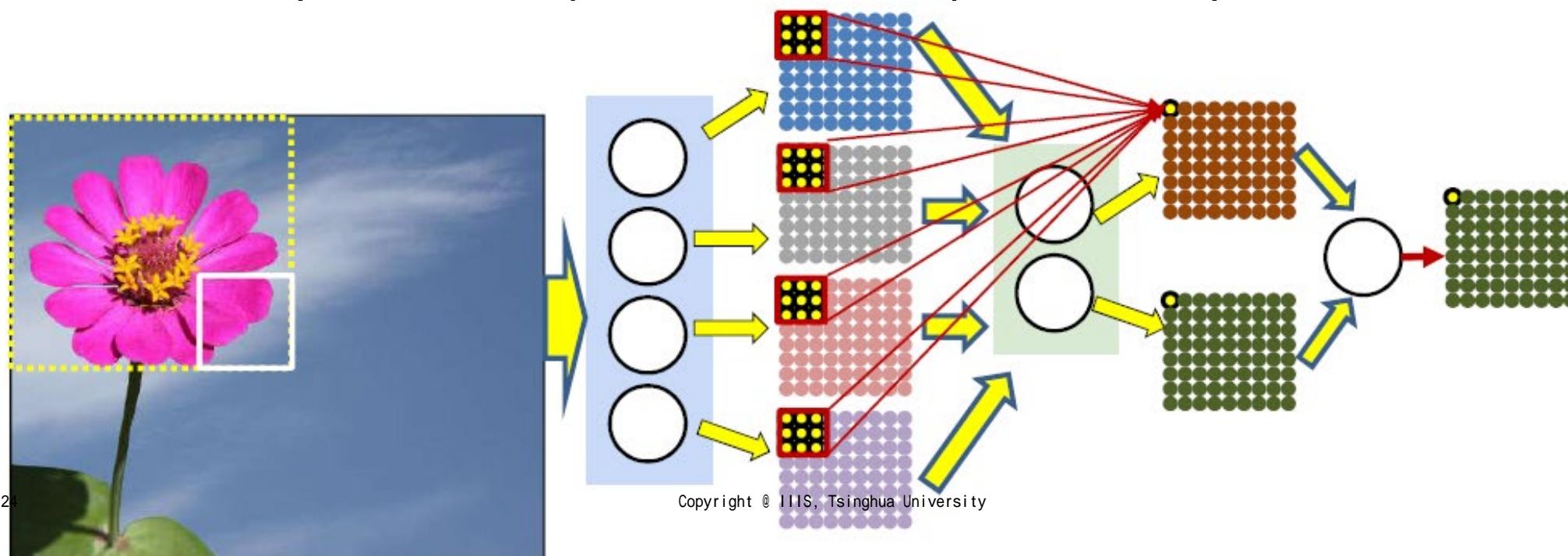
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches



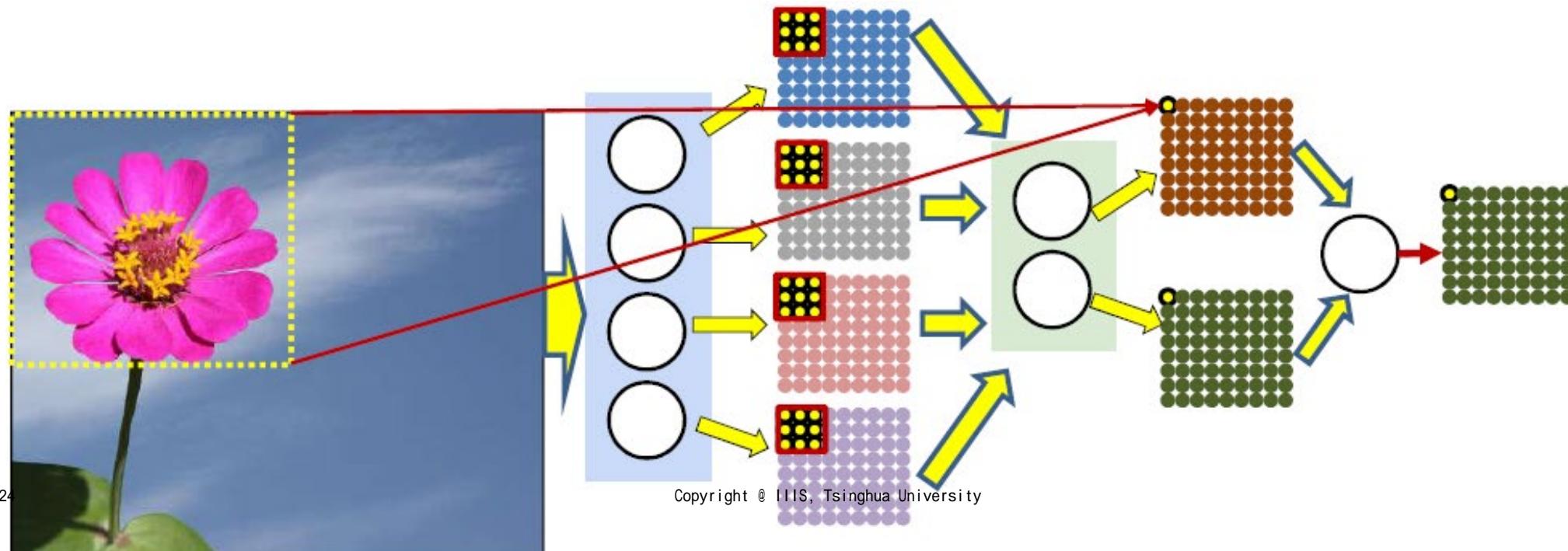
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches
 - Second layer scans the patch from the output of first layer



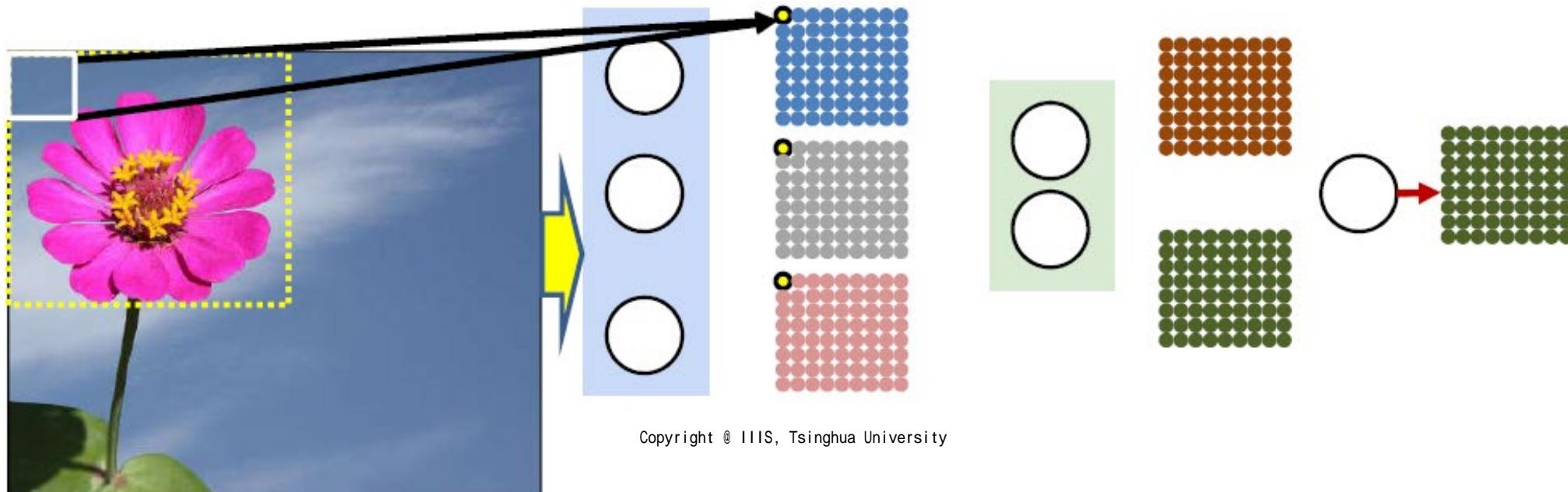
Distributed Scanning

- Let's distribute the pattern matching workload over 2 layers
 - E.g., perform 9x9 patch scanning by 2 layers of 3x3 scanning
 - First layer for smaller patches and second layer for larger patches
 - Second layer scans the patch from the output of first layer → **larger patch**



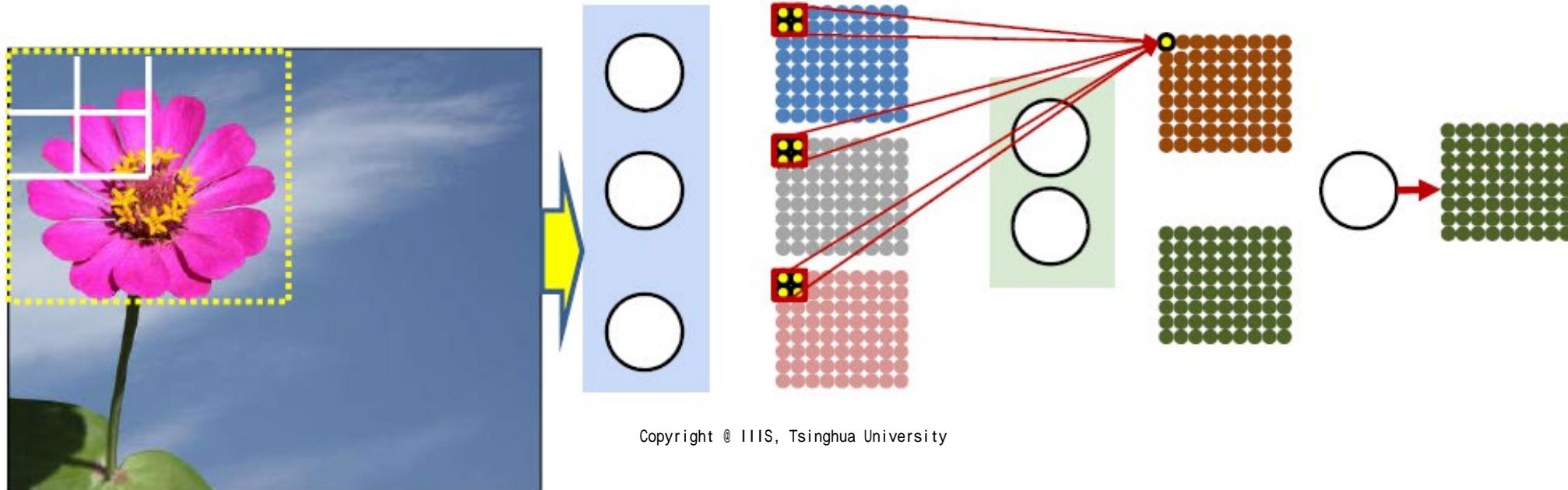
Distributed Scanning

- Let's learn the patterns over 3 layers
 - A similar recursive logic
 - First layer for small pattern



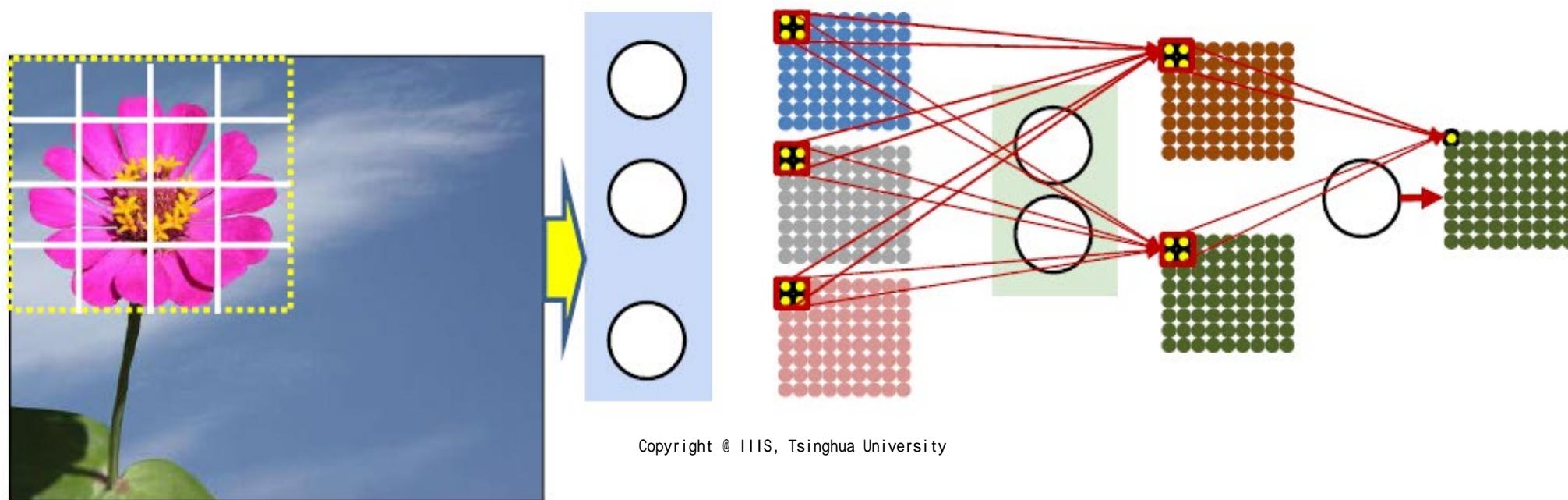
Distributed Scanning

- Let's learn the patterns over 3 layers
 - A similar recursive logic
 - Second layer for intermediate pattern



Distributed Scanning

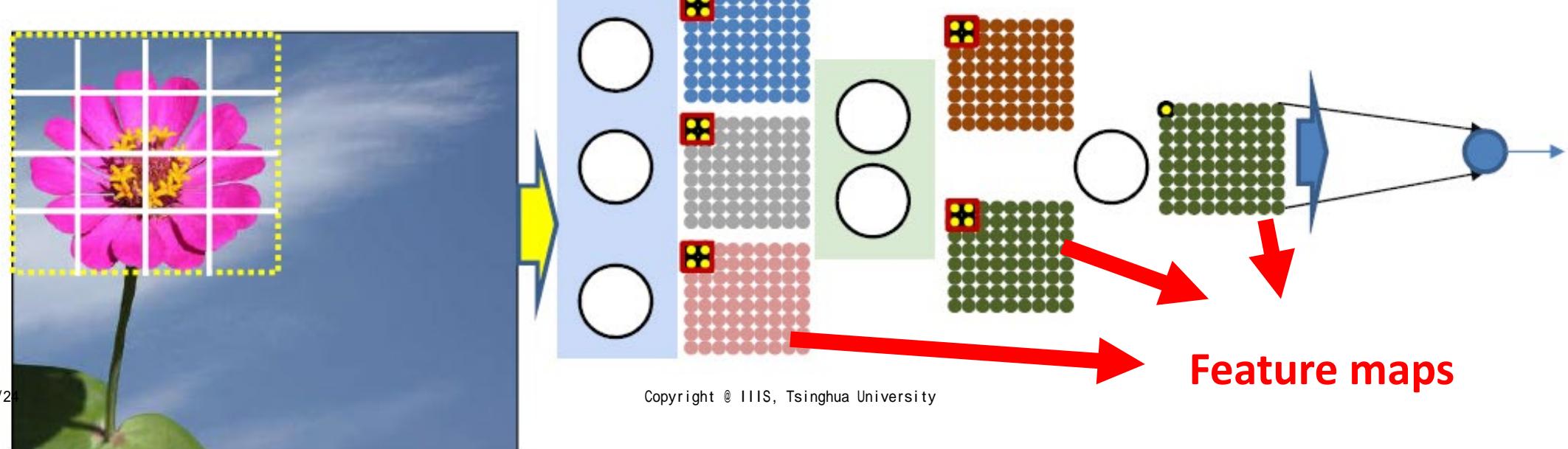
- Let's learn the patterns over 3 layers
 - A similar recursive logic
 - Top layer for the entire complex pattern



Distributed Scanning

- Let's learn the patterns over 3 layers
 - A similar recursive logic
 - Top layer for the entire complex pattern
 - Final MLP output for classification result over the entire image

A giant network building up a hierarchy of features!

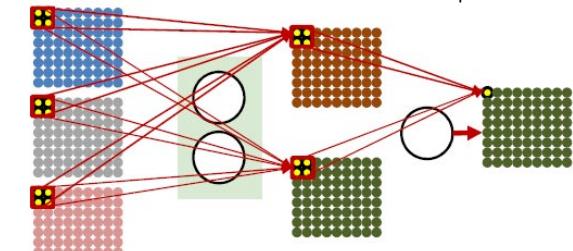
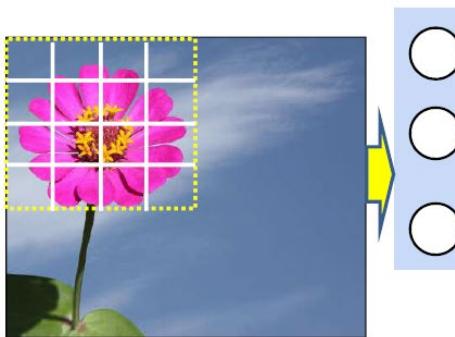


Pseudo-code

```

Y(0,:,:,: ) = Image
for l = 1:L
    for j = 1:Dl
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                z(l,j,x,y) = 0
                for i = 1:Dl-1
                    for x' = 1:Kl
                        for y' = 1:Kl
                            z(l,j,x,y) += w(l,i,j,x',y')
                            Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))
Y = softmax( Y(L,:,:1,...Y(L,:,W-K+1,H-K+1) ) )

```

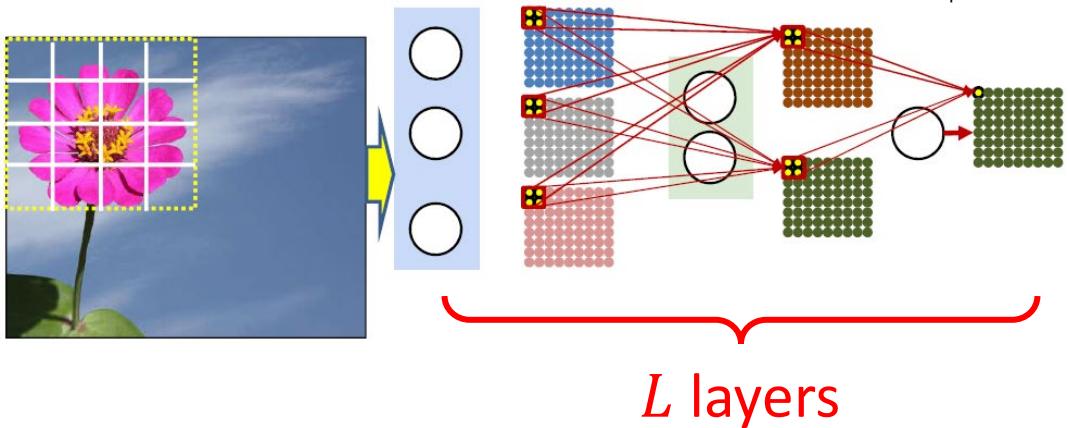


Pseudo-code

```

Y(0,:,:,: ) = Image
for l = 1:L # layers
    for j = 1:Dl
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                z(l,j,x,y) = 0
                for i = 1:Dl-1
                    for x' = 1:Kl
                        for y' = 1:Kl
                            z(l,j,x,y) += w(l,i,j,x',y')
                            Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))

```

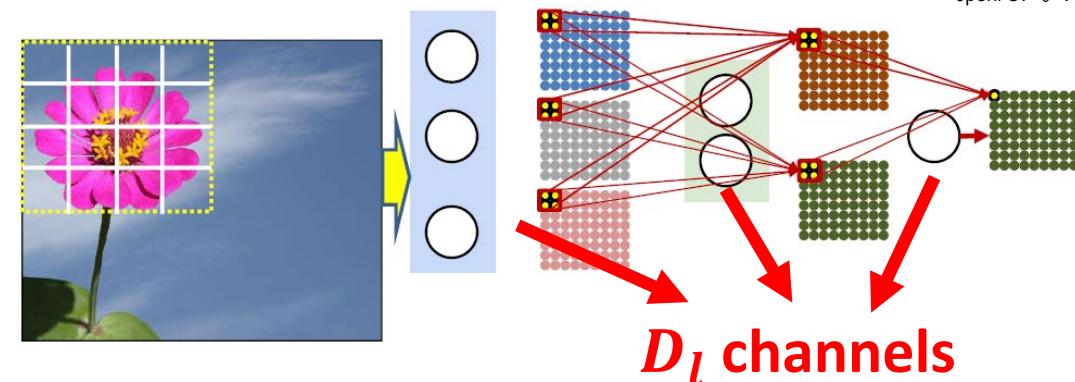


Pseudo-code

```

Y(0,:,:,: ) = Image
for l = 1:L # layers
    for j = 1:Dl # Dl is often called channels in computer vision
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                z(l,j,x,y) = 0
                for i = 1:Dl-1
                    for x' = 1:Kl
                        for y' = 1:Kl
                            z(l,j,x,y) += w(l,i,j,x',y')
                            Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))

```

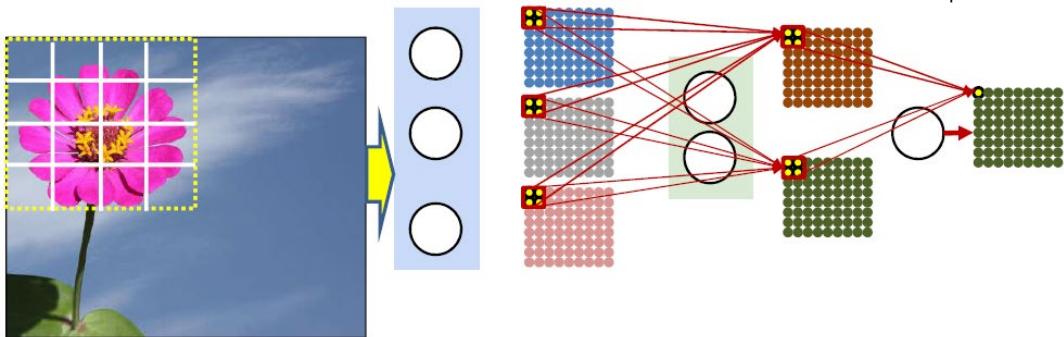


Pseudo-code

```

Y(0,:,:,: ) = Image
for l = 1:L # layers
    for j = 1:Dl # Dl is often called channels in computer vision
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                z(l,j,x,y) = 0 # compute feature at layer l, channel j at (x, y)
                for i = 1:Dl-1
                    for x' = 1:Kl
                        for y' = 1:Kl
                            z(l,j,x,y) += w(l,i,j,x',y')
                            Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))
Y = softmax( Y(L,:,1,1)...Y(L,:,W-K+1,H-K+1) )

```

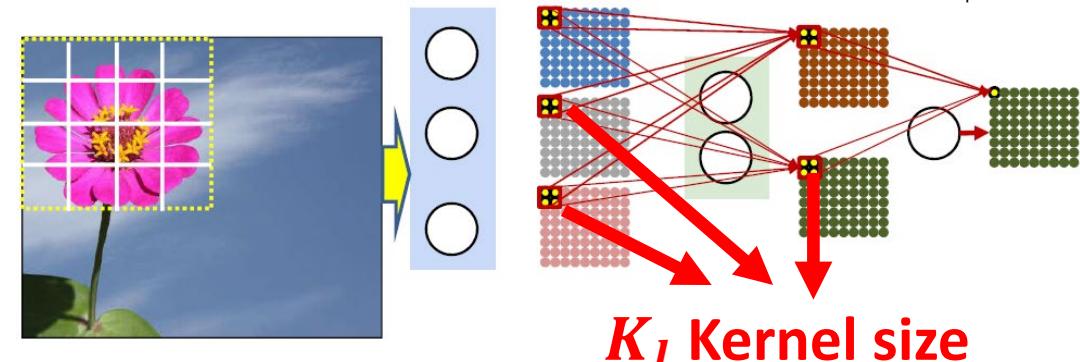


Pseudo-code

```

Y(0,:,:,: ) = Image
for l = 1:L # layers
    for j = 1:Dl # Dl is often called channels in computer vision
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                z(l,j,x,y) = 0 # compute feature at layer l, channel j at (x, y)
                for i = 1:Dl-1
                    for x' = 1:Kl # Kl is often called kernel size
                        for y' = 1:Kl
                            z(l,j,x,y) += w(l,i,j,x',y')
                            Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))

```



Pseudo-code

```

Y(0,:,:,: ) = Image
for l = 1:L # layers

```

for j = 1:D_l # D_l is often called channels in computer vision

```
for x = 1:Wl-1-Kl+1
```

for This operation is called “Convolution”

```
z(l,j,x,y) = 0 # compute feature at layer l, channel j at (x, y)
```

```
for i = 1:Dl-1
```

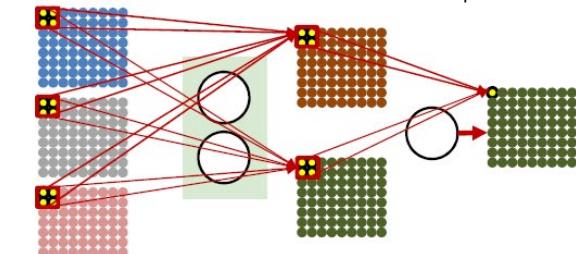
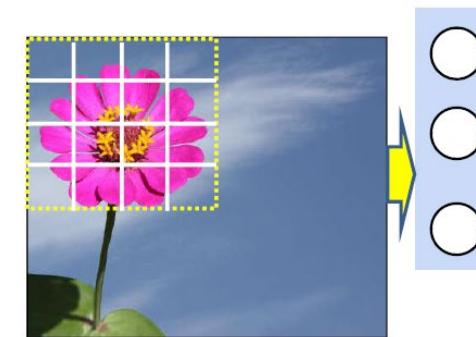
for x' = 1:K_l # K_l is often called kernel size

```
for y' = 1:Kl
```

```
z(l,j,x,y) += w(l,i,j,x',y')
```

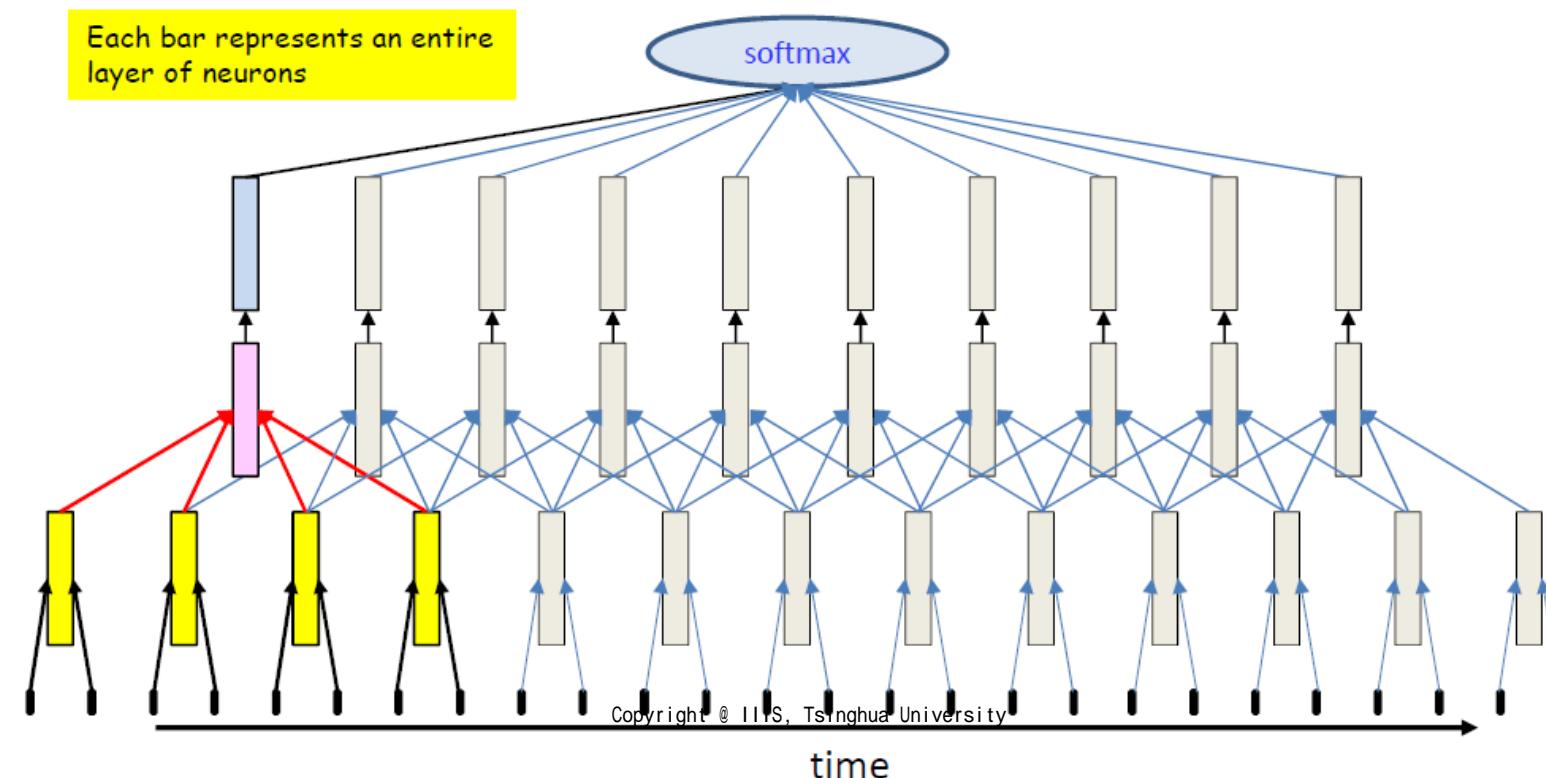
```
Y(l-1,i,x+x'-1,y+y'-1)
```

```
Y(l,j,x,y) = activation(z(l,j,x,y))
```



Distributed Scanning in 1D Case

- We can scan a 8-timestep patch with distributed scan over 2 layers
 - First layer: takes inputs over 2 timesteps and a stride of 2 timesteps
 - Second layer: takes input over 4 timesteps from first layer

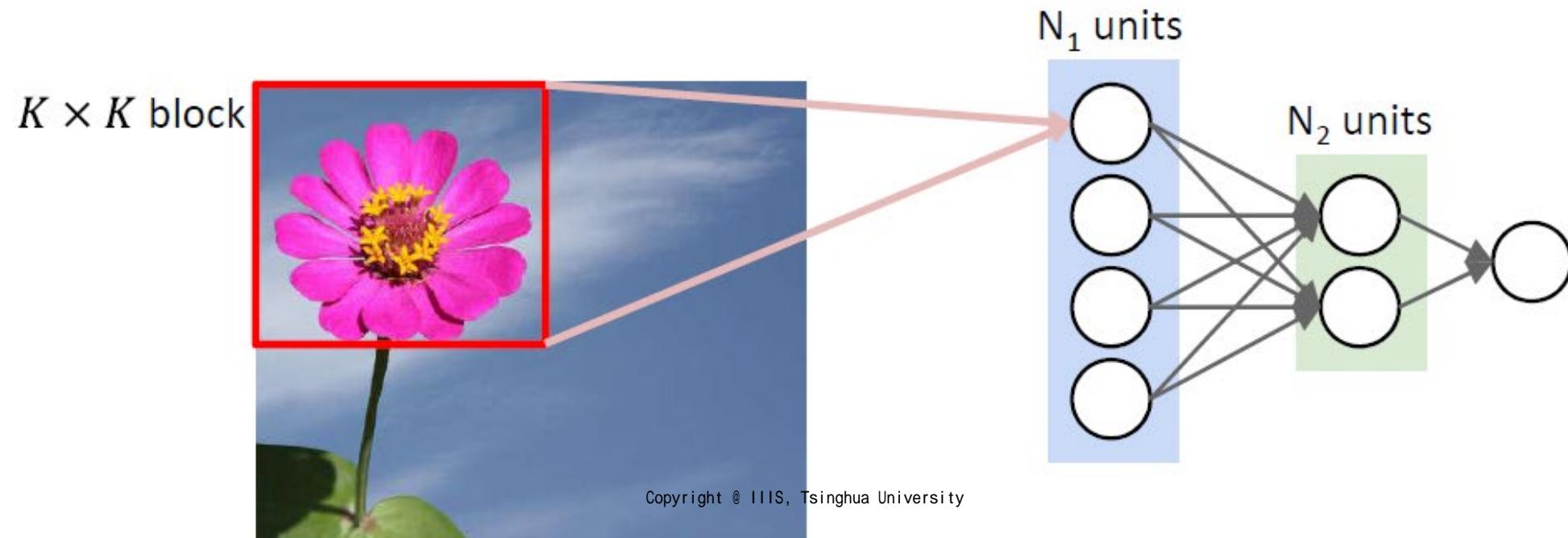


Why Distributed Scanning?

- Each layer focuses on localized patterns
 - Weights have lower dimensions
 - Easy to learn and more generalizable
- Number of Parameters!

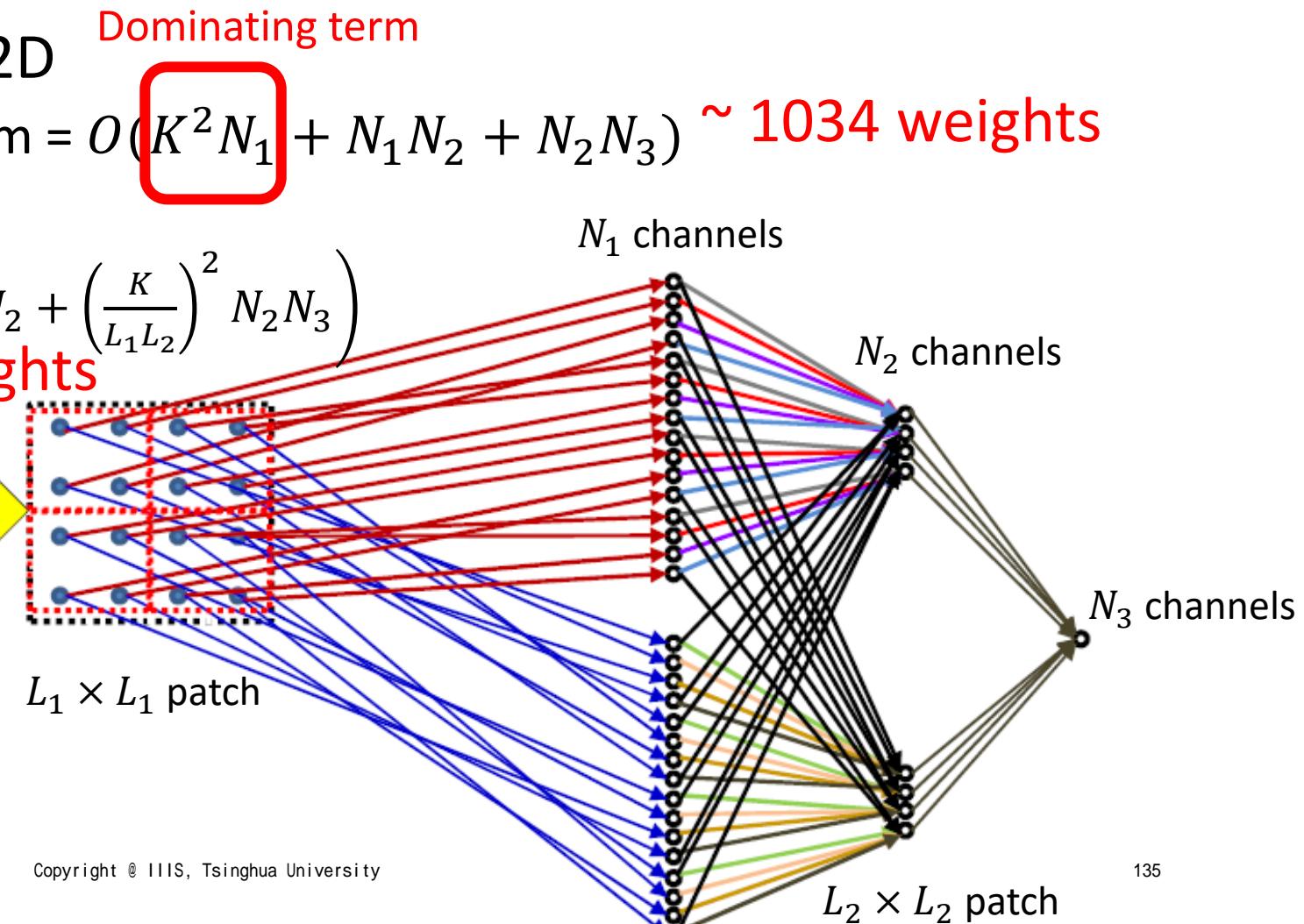
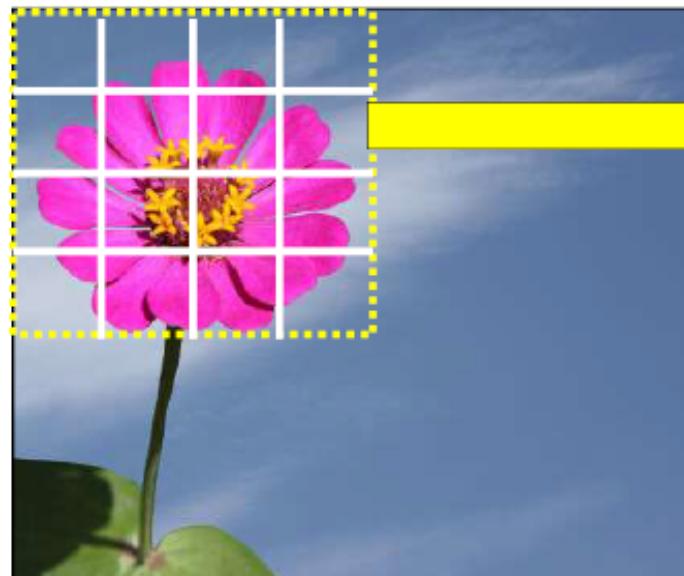
Why Distributed Scanning?

- Number of Parameters in 2D
 - Non-distributed net:
 - #Param = $O(K^2N_1 + N_1N_2 + N_2N_3)$



Why Distributed Scanning?

- Number of Parameters in 2D
 - Non-distributed net: #Param = $O(K^2 N_1 + N_1 N_2 + N_2 N_3)$ **Dominating term** $\boxed{K^2 N_1}$ ~ 1034 weights
 - Distributed over 3 layers:
 - #Param = $O\left(L_1^2 N_1 + L_2^2 N_1 N_2 + \left(\frac{K}{L_1 L_2}\right)^2 N_2 N_3\right)$ ~ 160 weights

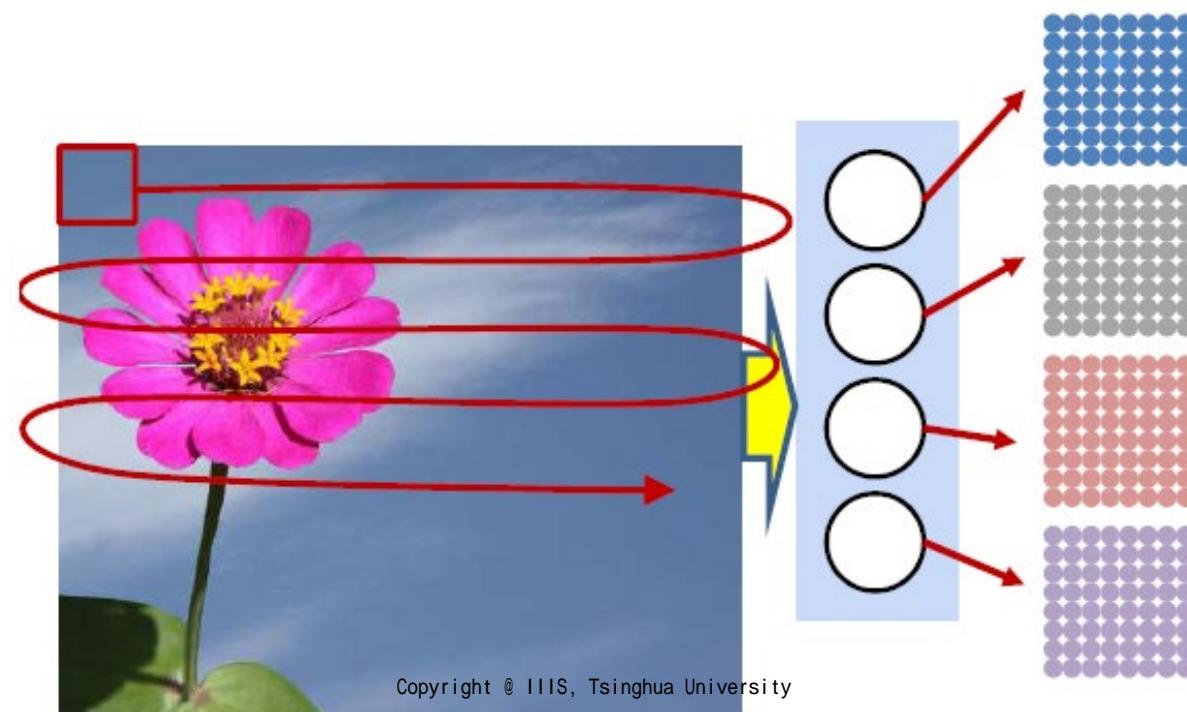


Why Distributed Scanning?

- Each layer focuses on localized patterns
 - Weights have lower dimensions
 - Easy to learn and more generalizable
- Number of Parameters!
 - Significantly reduce the amount of parameters when use more layers

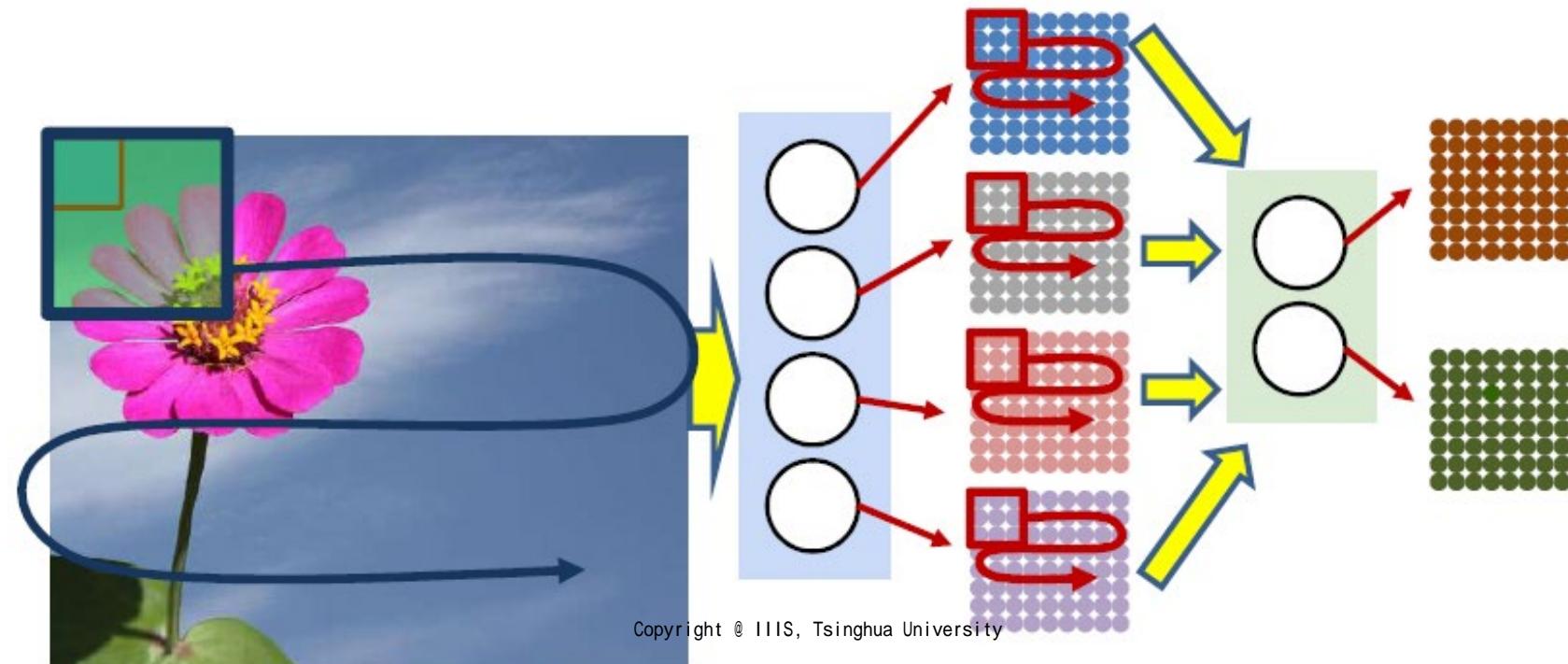
A More General Form of Scanning

- We do not necessarily need *precise* distribution over layers
- Let's re-examine the convolution operators
 - First layer scans small local sub-regions



A More General Form of Scanning

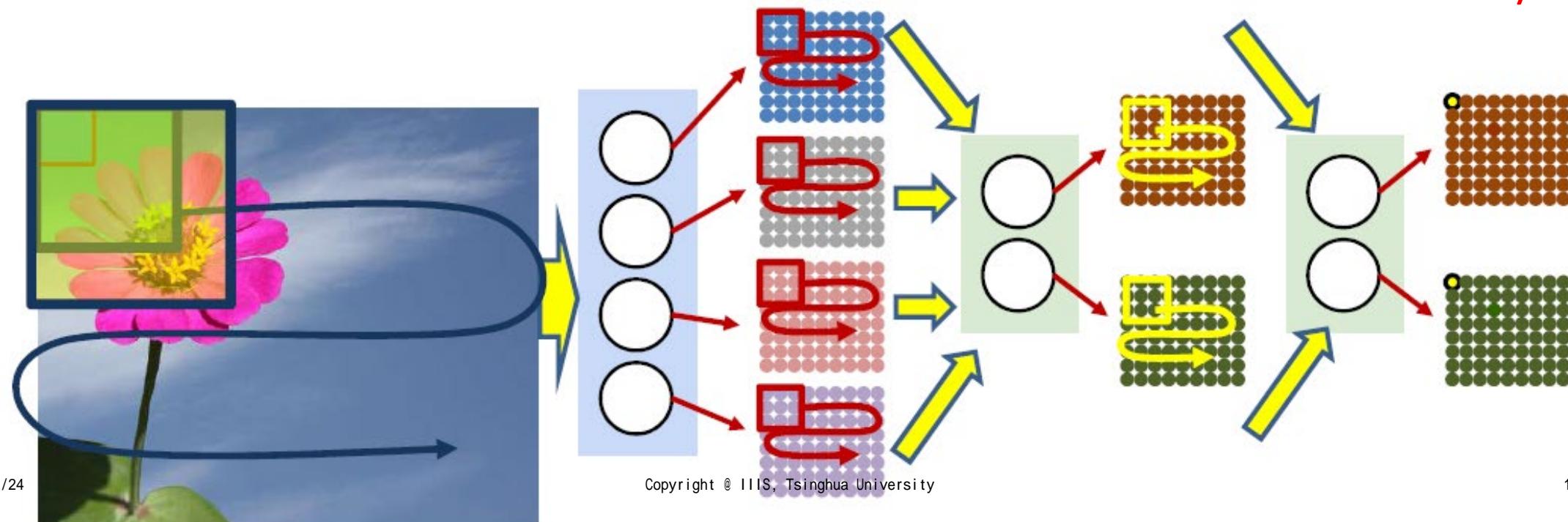
- We do not necessarily need *precise* distribution over layers
- Let's re-examine the convolution operators
 - Second layer scans subregions from the first layer → larger patch in the image



A More General Form of Scanning

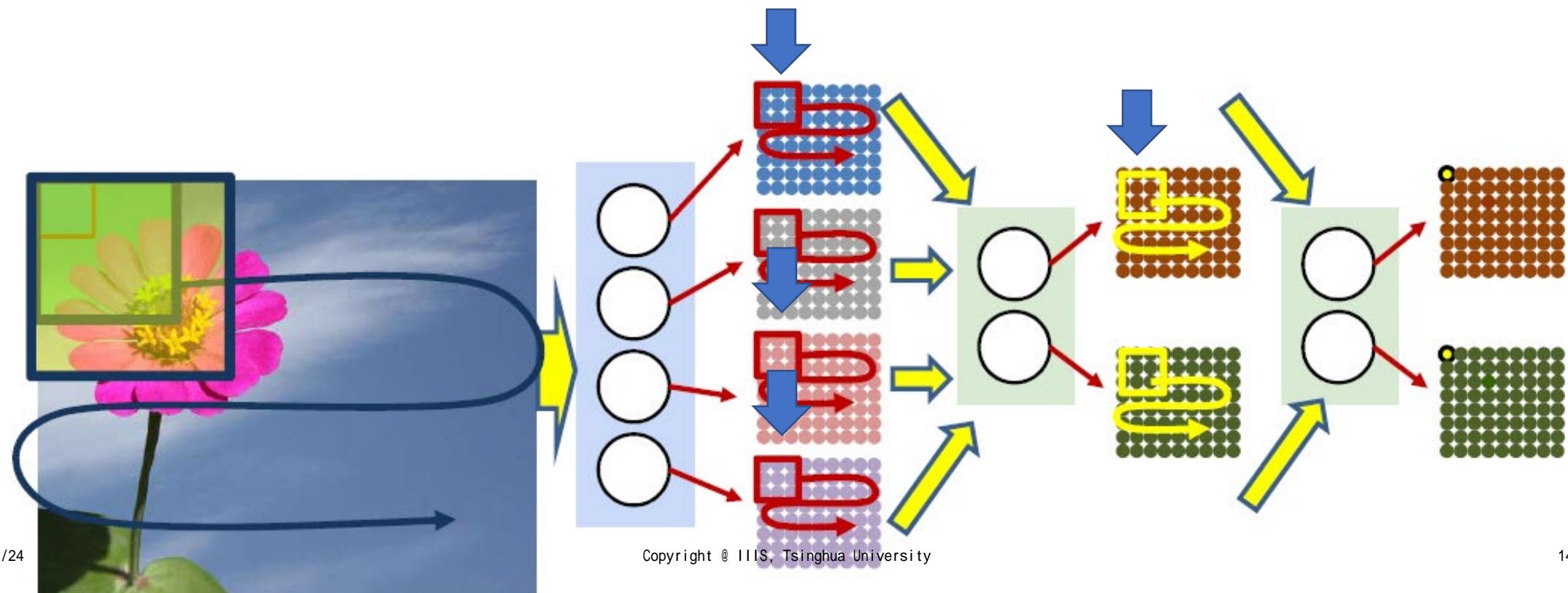
- We do not necessarily need *precise* distribution over layers
- Let's re-examine the convolution operators
 - Third layer just scans subregions from the second layer

We can have
arbitrarily many
layers



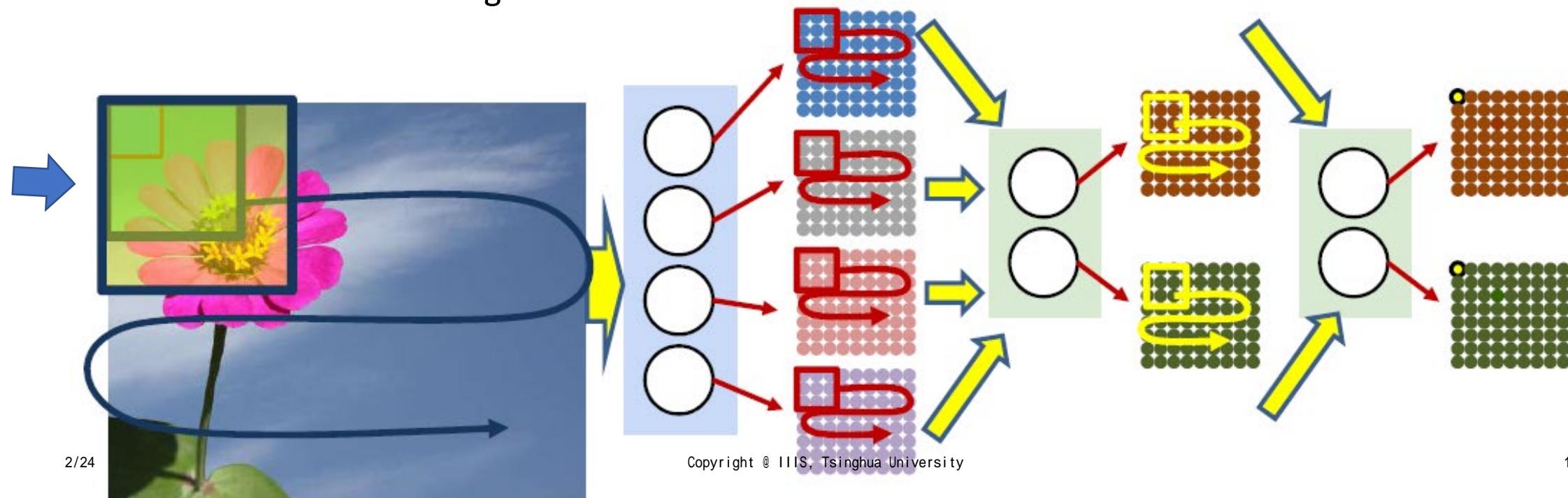
A More General Form of Scanning

- Terminology
 - Filters: scans for a pattern on the map from the previous layer



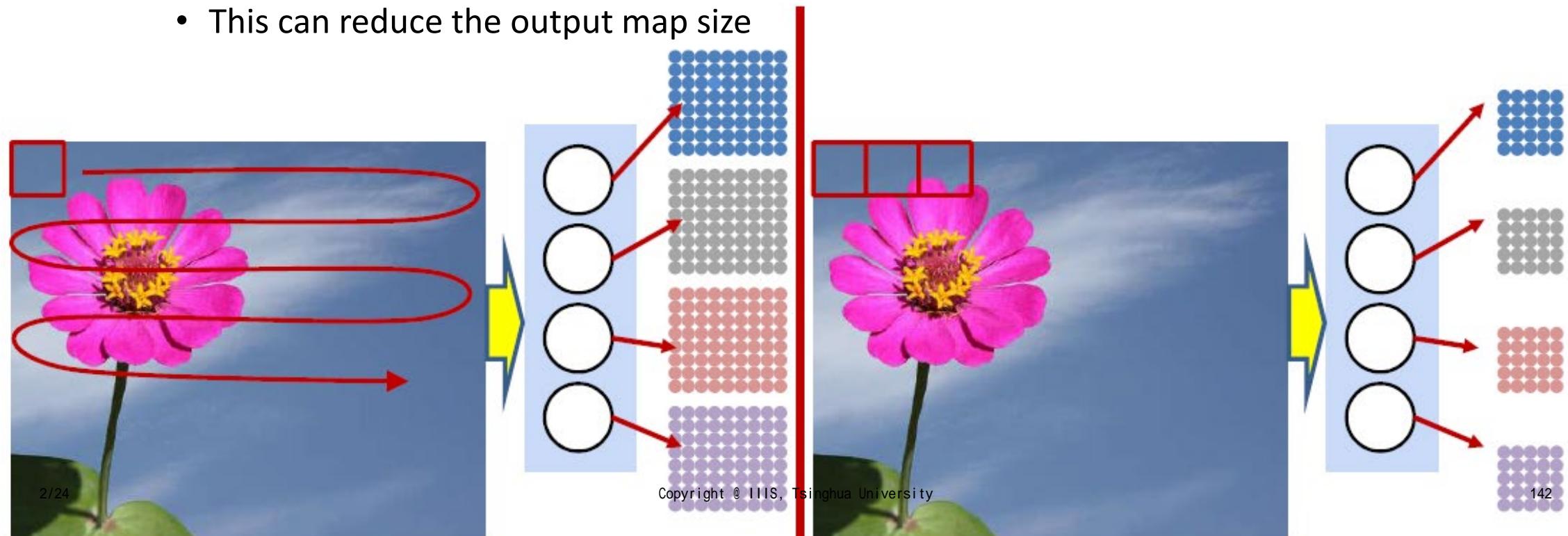
A More General Form of Scanning

- Terminology
 - Filters
 - Receptive Fields: the corresponding patch in the *input* image
 - Non-trivial for high-level filters



A More General Form of Scanning

- Terminology
 - Filters, Receptive Fields
 - Strides: the scanning “hops” for each filter
 - This can reduce the output map size



A More General Form of Scanning

- Size of Output Map
 - Filter Size M ; Input Map Size N ; Stride S
 - Convolution often reduces the map size even with $S = 1$

$M \times M$



Filter

Size : $N \times N$

$\left[\frac{N-M}{S} \right] + 1$

Output Size

?

A More General Form of Scanning

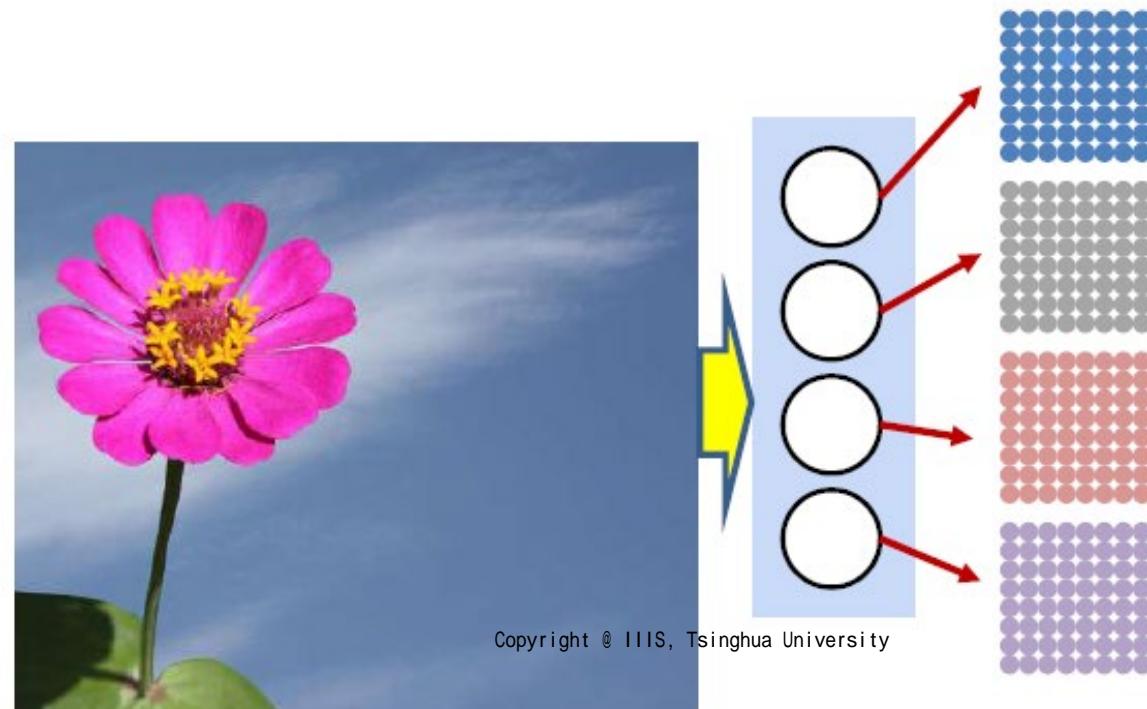
- Size of Output Map
 - Filter Size M ; Input Map Size N ; Stride S
 - Convolution often reduces the map size even with $S = 1$
 - Solution: ***zero-padding***
 - Pad 0 all around the map
 - It ensures the output size is $\left\lceil \frac{N}{S} \right\rceil$
 - For stride=1,
the output map remains the same size as input map



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

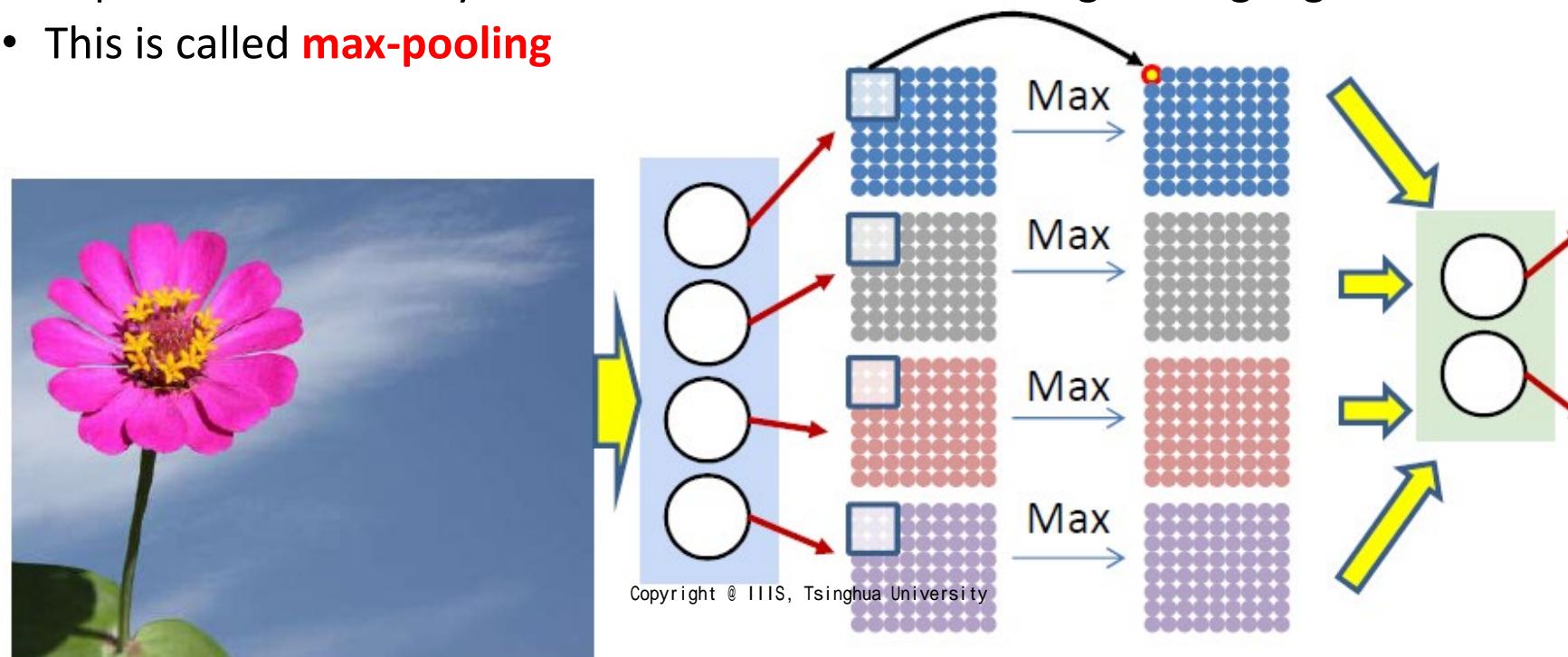
A More General Form of Scanning

- Account for jittering
 - If a pattern in the image shifts for 1 pixel, can we still detect it?
 - Even with a 1-pixel shift, a large portion of the feature map changes



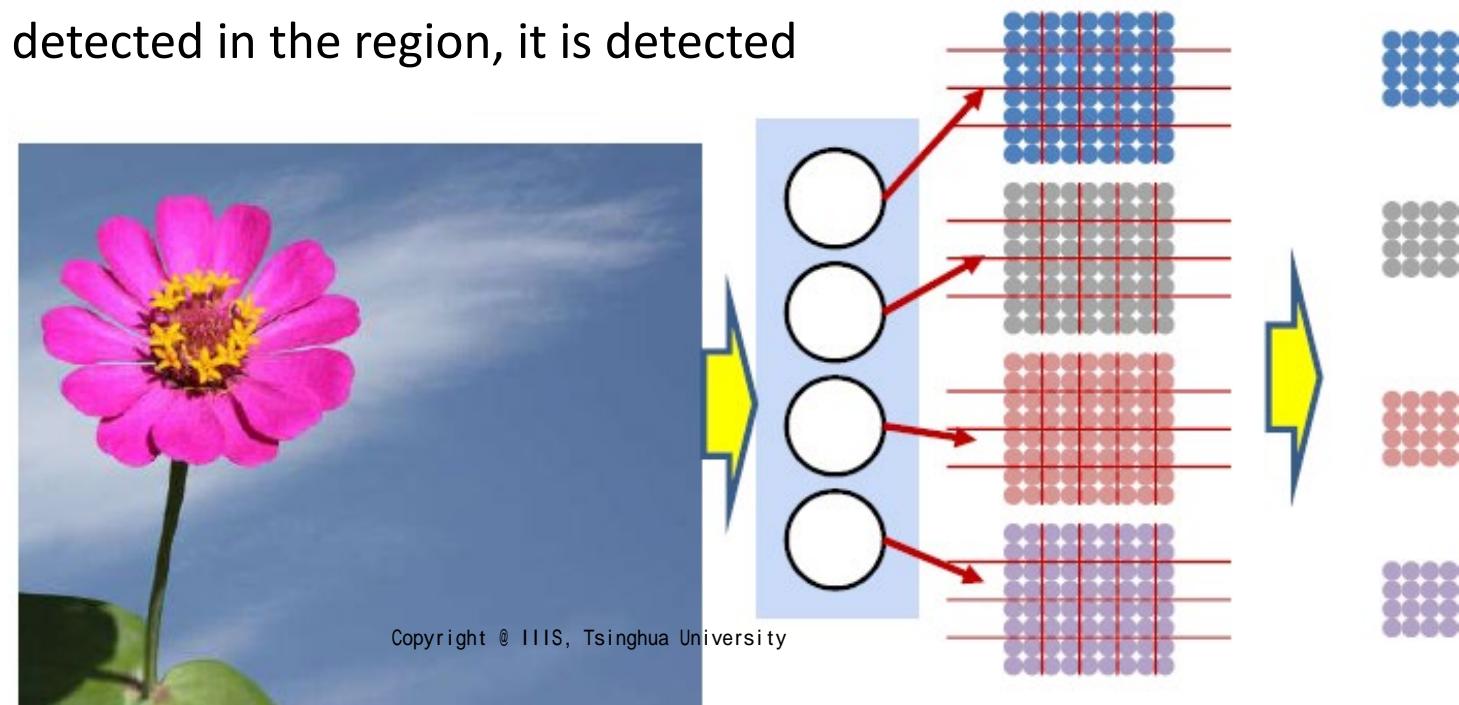
A More General Form of Scanning

- Account for jittering
 - If a pattern in the image shifts for 1 pixel, can we still detect it?
 - Small jittering is acceptable!
 - Replace each value by the maximum over a small neighboring region
 - This is called **max-pooling**



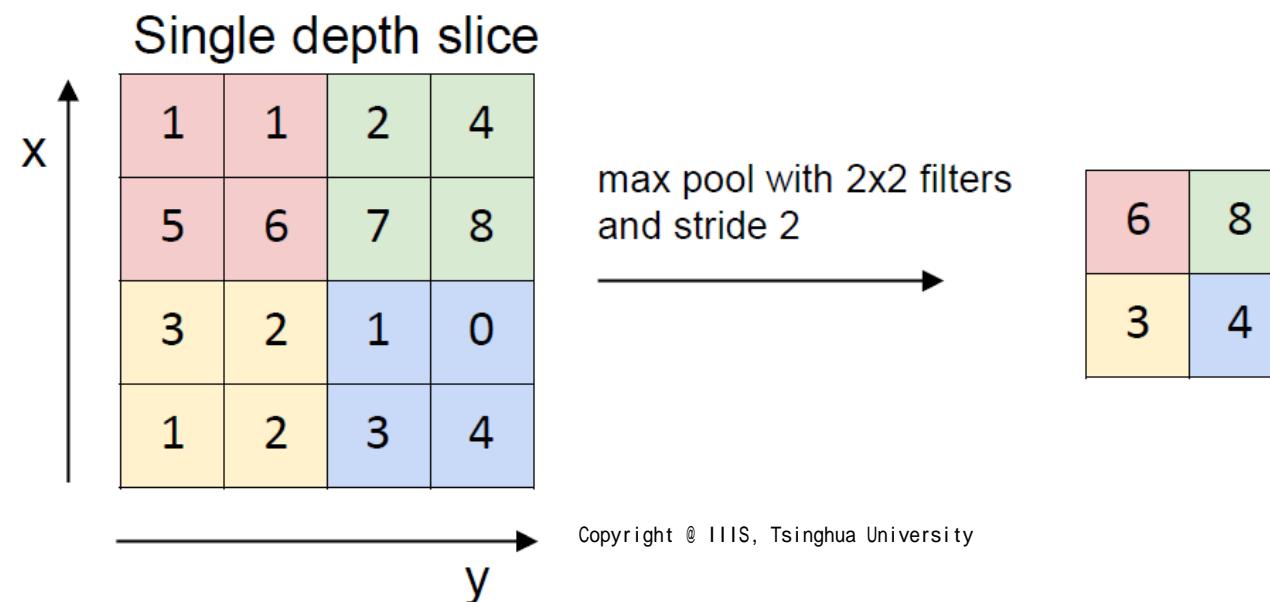
A More General Form of Scanning

- Max-Pooling typically has non-overlap strides
 - Stride = max-pooling size
 - It partitions the output map into blocks
 - Each block only maintain the highest value
 - Any pattern detected in the region, it is detected



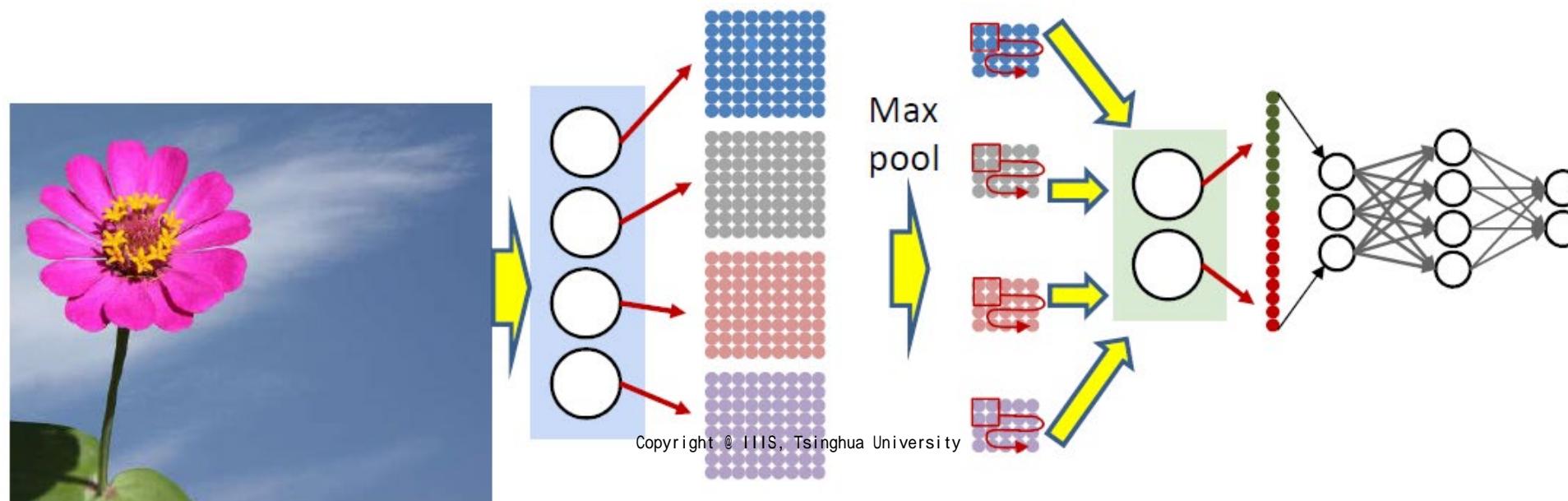
A More General Form of Scanning

- Max-Pooling typically has non-overlap strides
 - Stride = max-pooling size
 - It partitions the output map into blocks
 - Each block only maintain the highest value
 - Any pattern detected in the region, it is detected



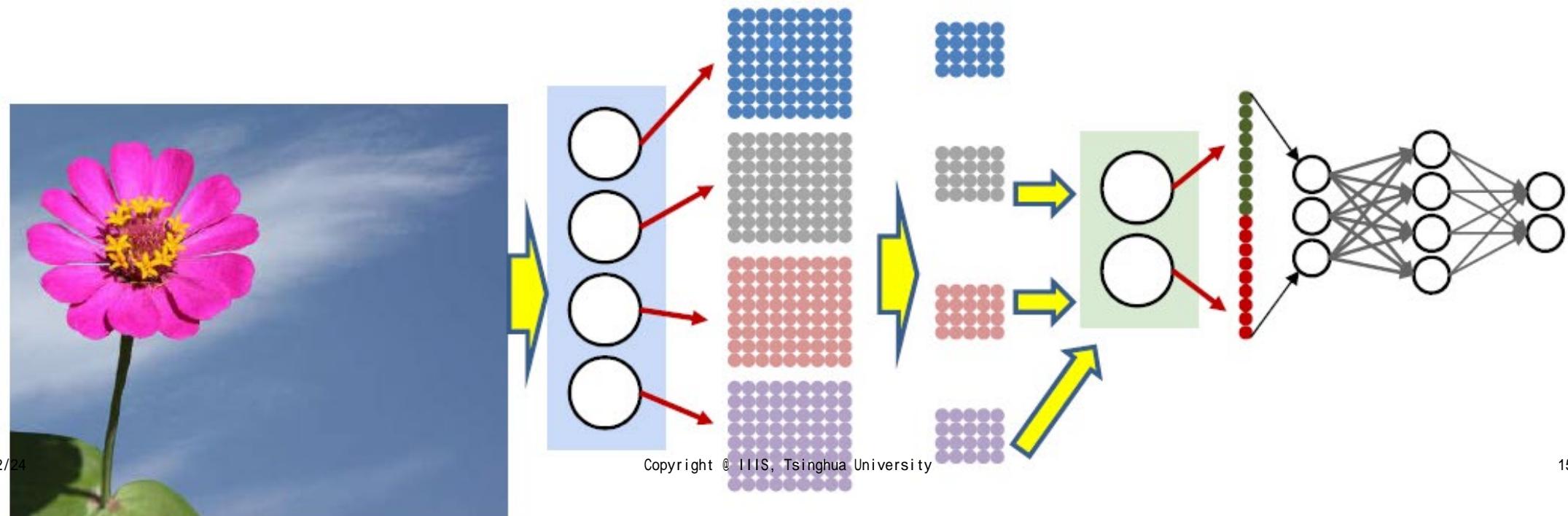
A More General Form of Scanning

- Max-Pooling typically has non-overlap strides
 - Stride = max-pooling size
 - It partitions the output map into blocks
 - The next layer works on pooled map



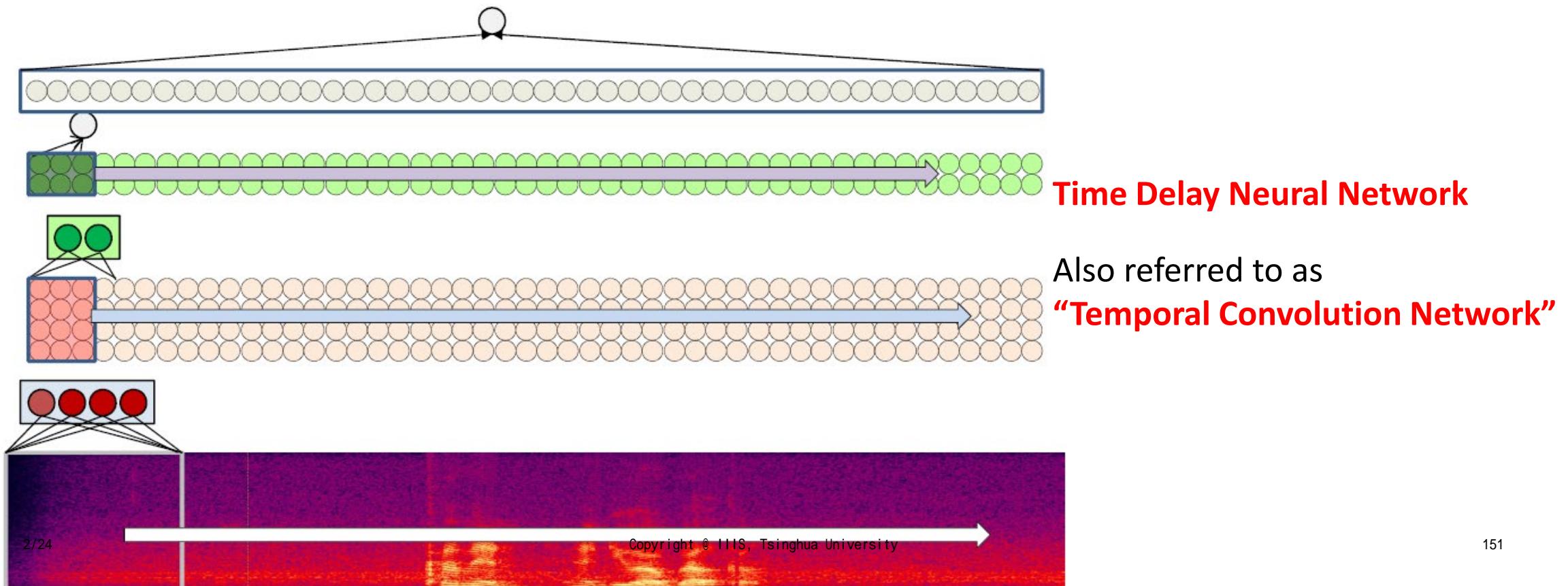
Convolutional Neural Networks

- The entire architecture is called CNN
 - Convolution layer
 - Max-Pooling layer
 - Final MLP layer for classification output



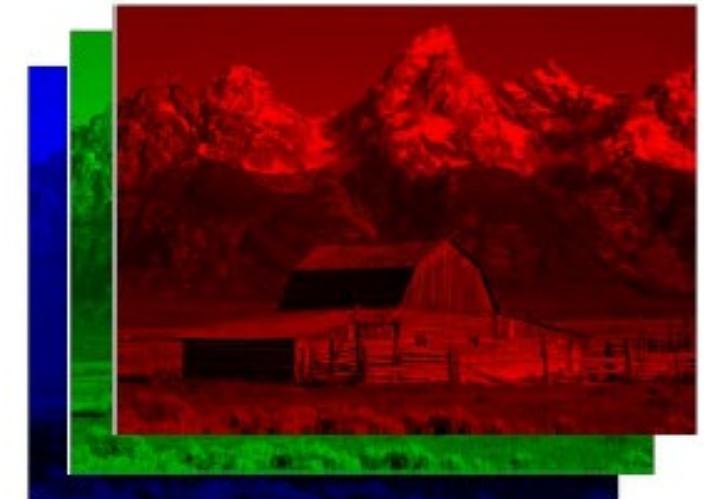
Back to the 1D Case

- 1D Convolution + Pooling + MLP → Classification
 - Whether the voice has “Welcome” in it



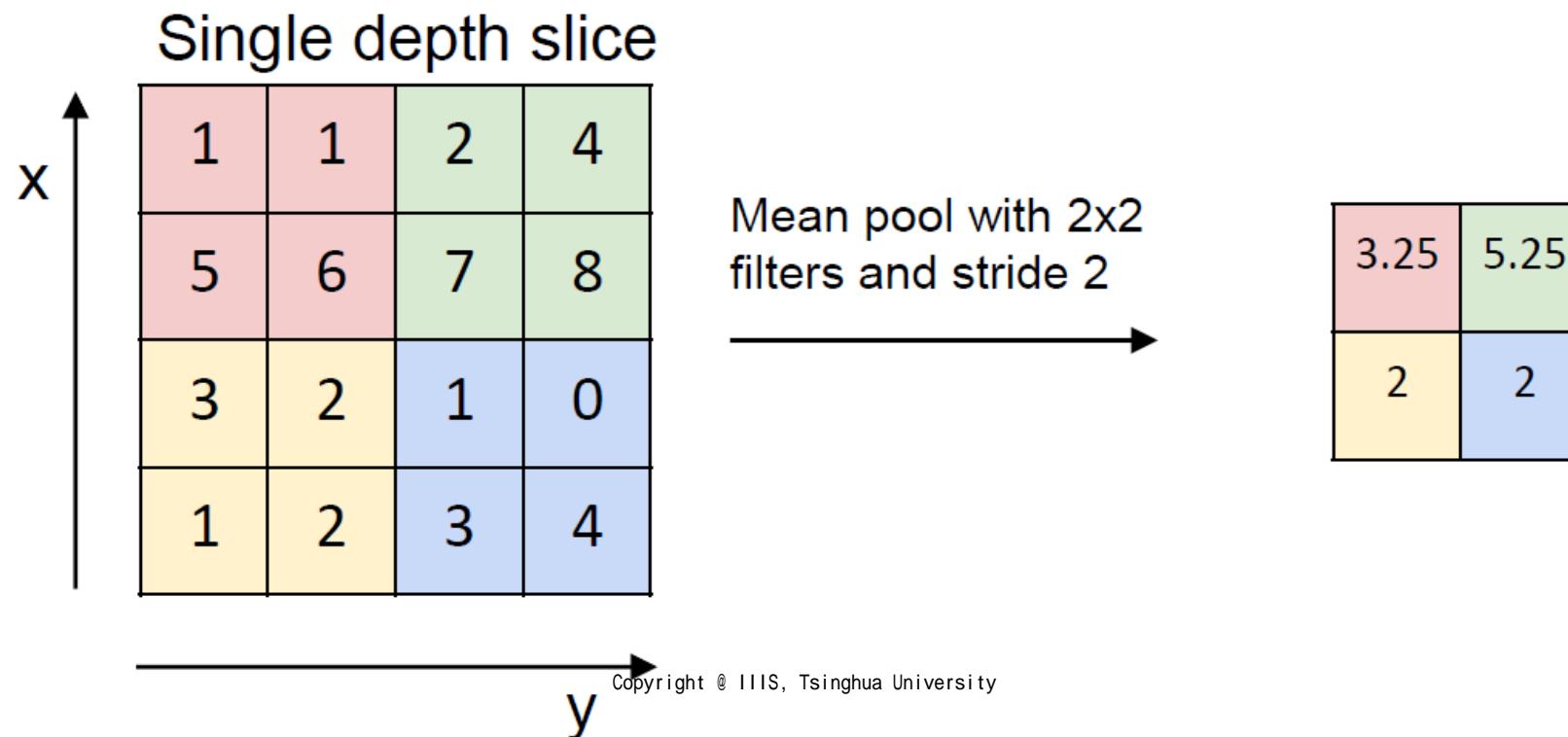
Additional Remarks

- The input channels
 - 1 (black-white) or 3 (RGB)



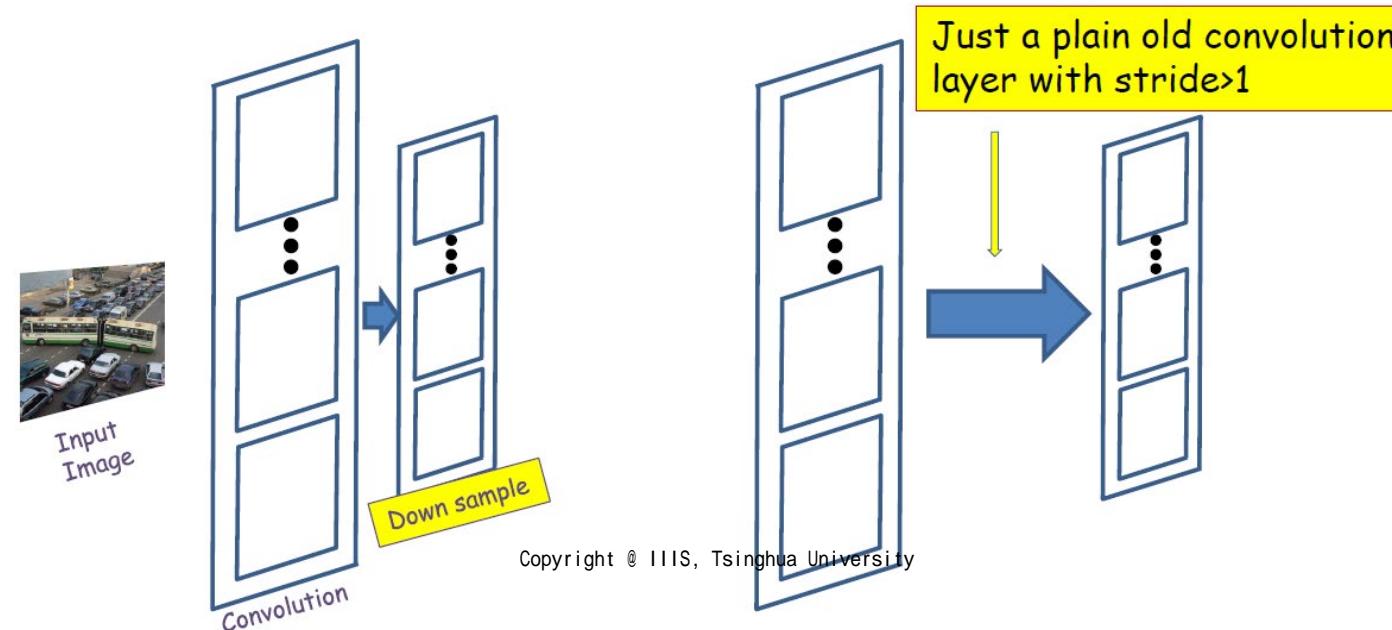
Additional Remarks

- Alternatives to Max-Pooling
 - Average-Pooling: use mean instead of max
 - A soft version of max operator → more informative gradients



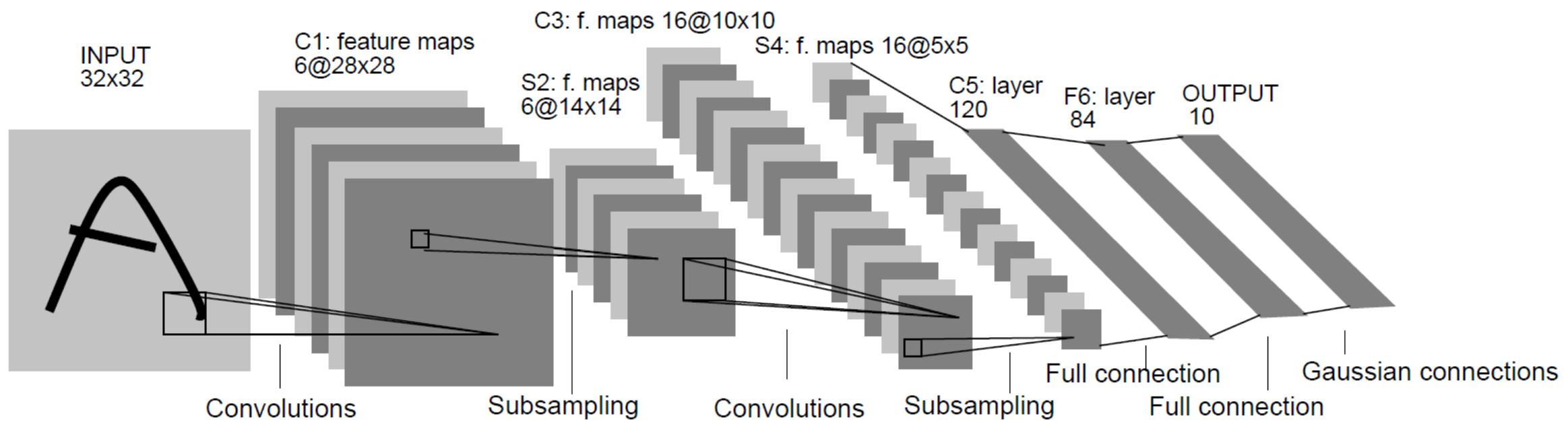
Additional Remarks

- Alternatives to Max-Pooling
 - Average-Pooling: use mean instead of max
 - Fully Convolutional Network: downsampling instead of pooling
 - Convolution with stride > 1 reduces the map size (downsampling)
 - Equivalent to “*learning a learned pooling operator*”



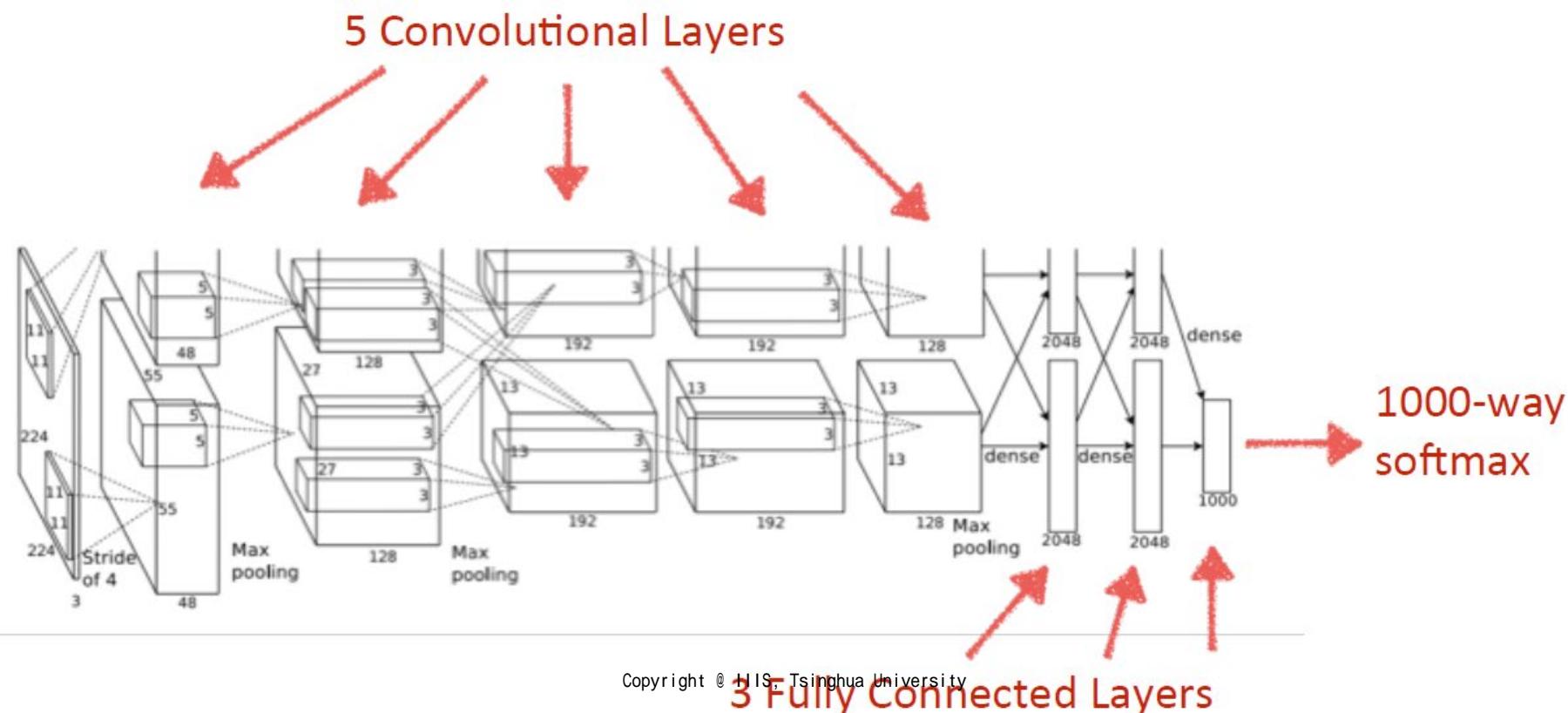
The LeNet-5 Example

- 1998 by Yan LeCun
 - First commercial CNN application for digit recognition



AlexNet

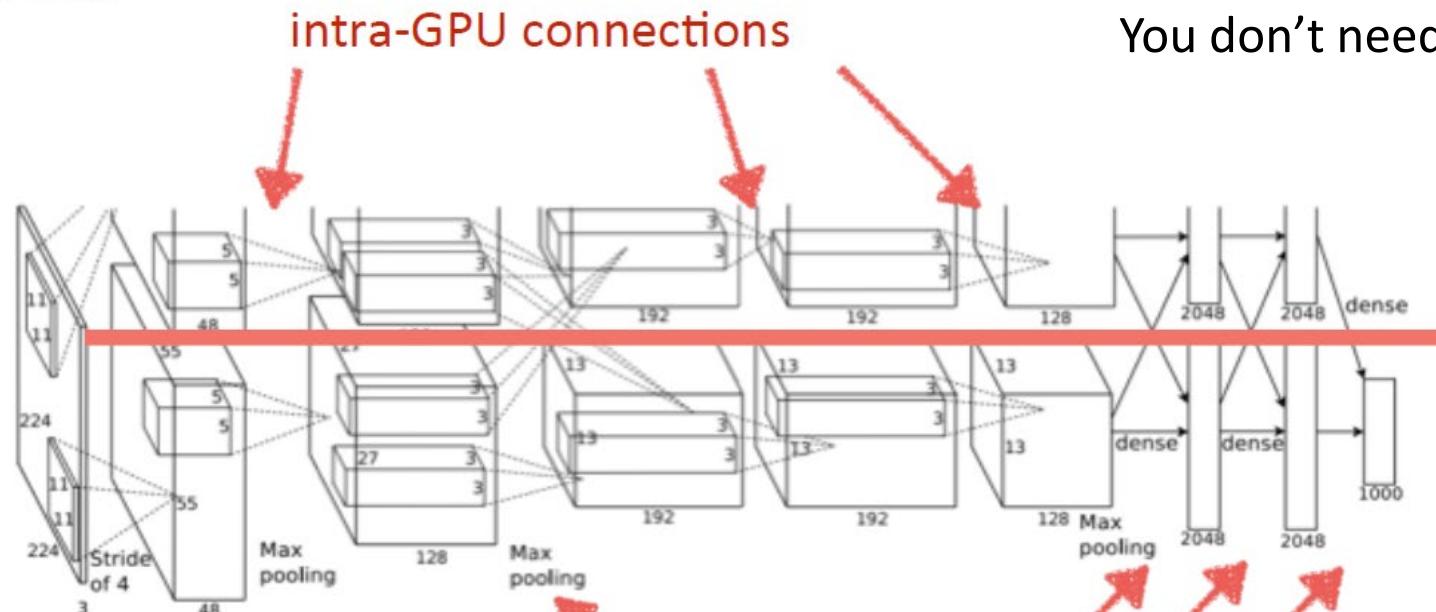
- 2012 by Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton
 - Breakthrough on ImageNet Challenge: the beginning of deep learning era



AlexNet

- 2012 by Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton
 - Breakthrough on ImageNet Challenge: the beginning of deep learning era

GPU #1



You don't need GPU for HW1 😊

GPU #2

Summary

- Part 1: learning an MLP for classification!
 - The basic components and learning algorithm
- Part 2: convolutional neural network
 - The intuition and basic architecture
- You are now ready for CP1!
 - Backpropagation and initial tuning attempt 😊
- Next lecture: more tricks are coming!
 - Get your hands extremely dirty!

Thanks