Machine Learning & Pattern Recognition

SONG Xuemeng

songxuemeng@sdu.edu.cn

http://xuemeng.bitcron.com/

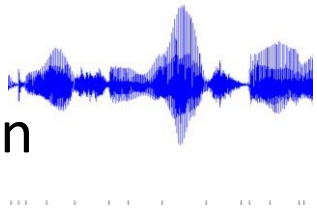
Neural Network

Data Representation

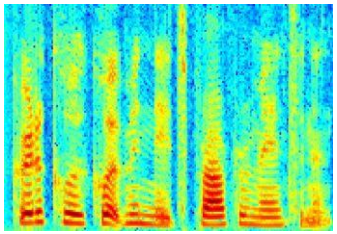
"hand-crafting"

Input data → Feature representation → Learning algorithm

Speech recognition



Audio



Features
E.g., MFCC.

Speaker identification

The mapping from representation to output

Data Representation

Car Detection

Data Representation

Car Detection

Feature Extraction

- The presence of a wheel

Data Representation

Car Detection

Feature Extraction

- The presence of a wheel

Difficult to describe what a wheel looks like!

Data Representation

Car Detection

Feature Extraction

- The presence of a wheel

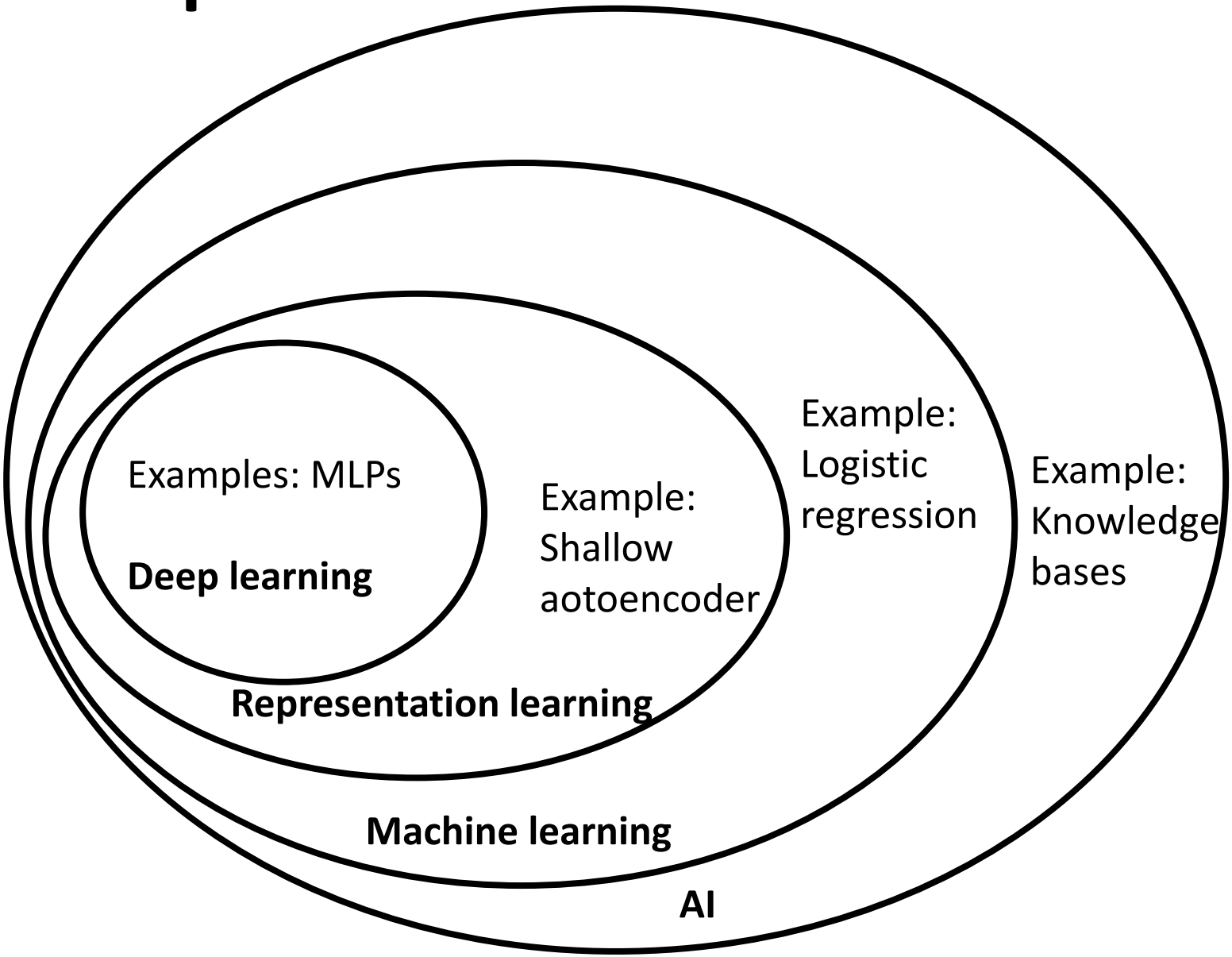
Difficult to describe what a wheel looks like!

Representation Learning

The mapping from representation to output

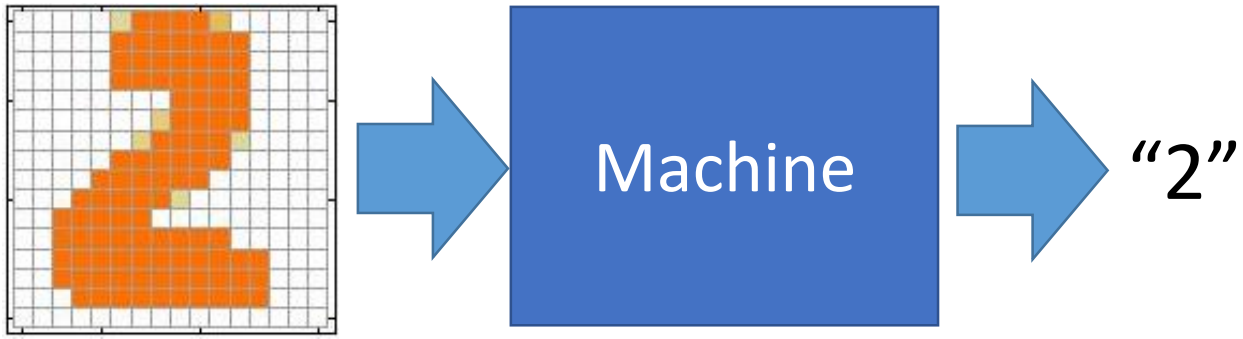
The representation itself

Data Representation



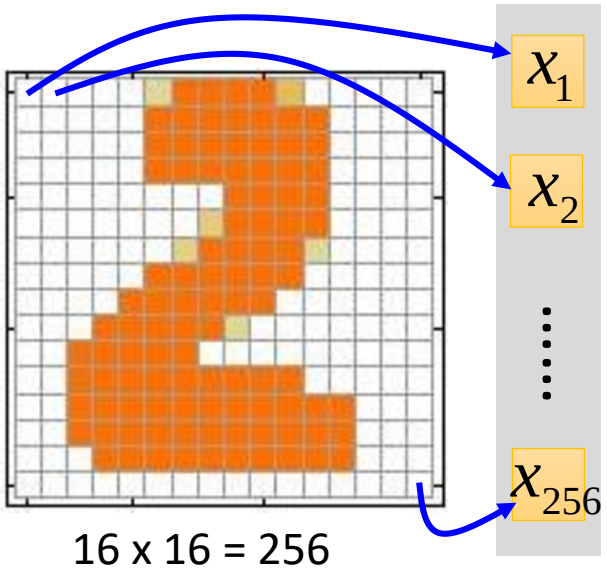
Example Application

- **Handwriting Digit Recognition**



Handwriting Digit Recognition

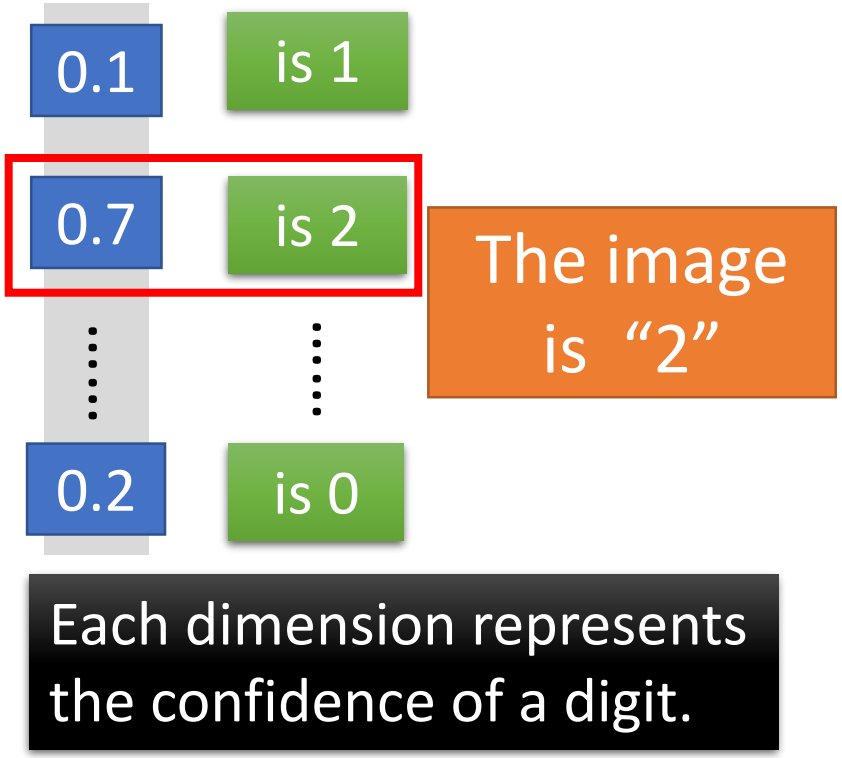
Input



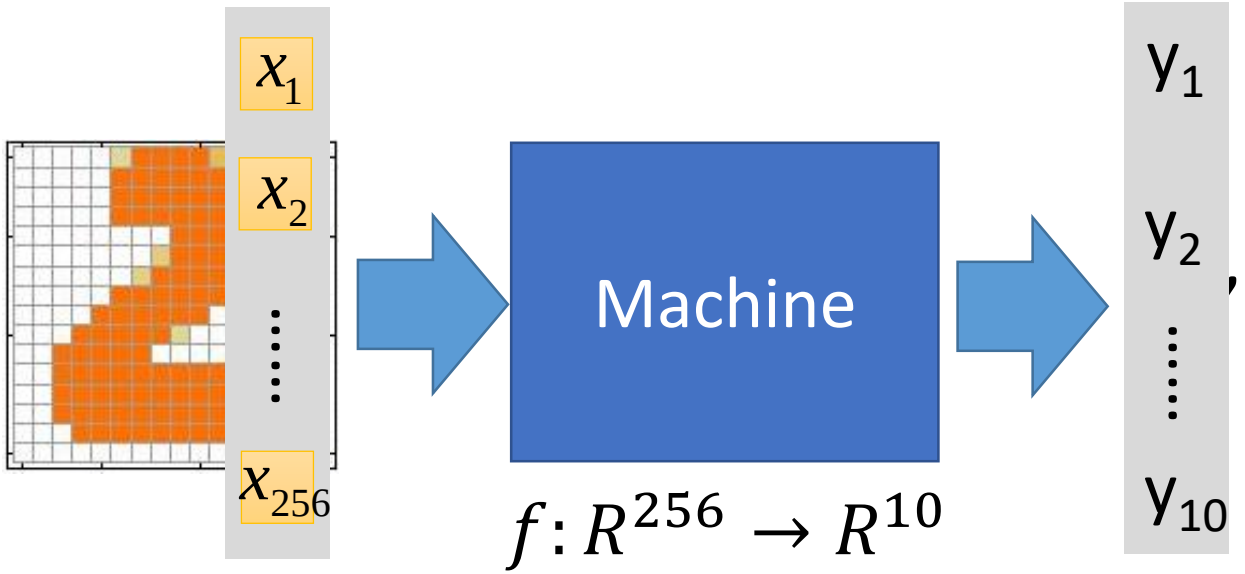
Ink → 1

No ink → 0

Output



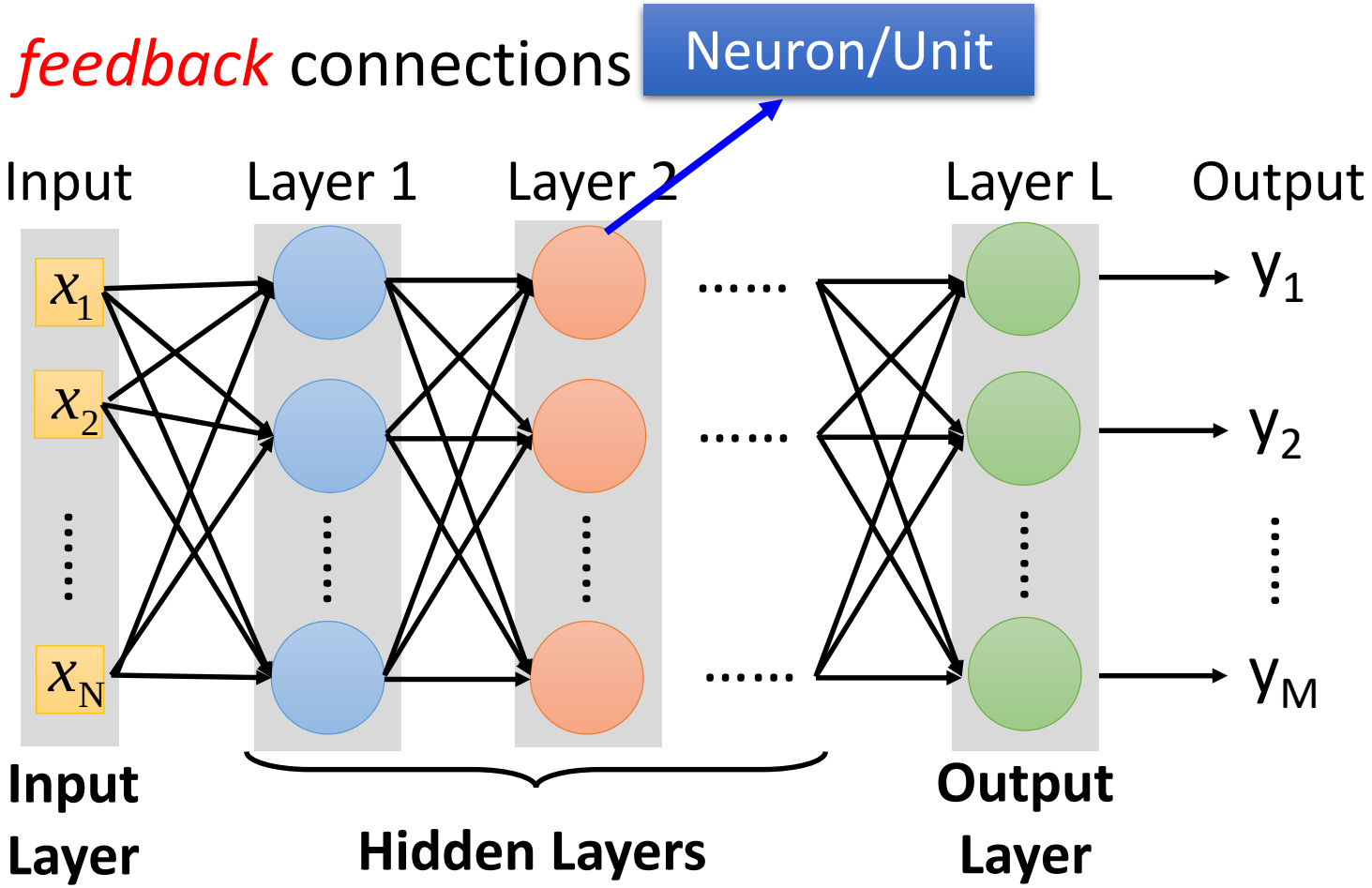
Handwriting Digit Recognition



In deep learning, the function f is represented by neural network

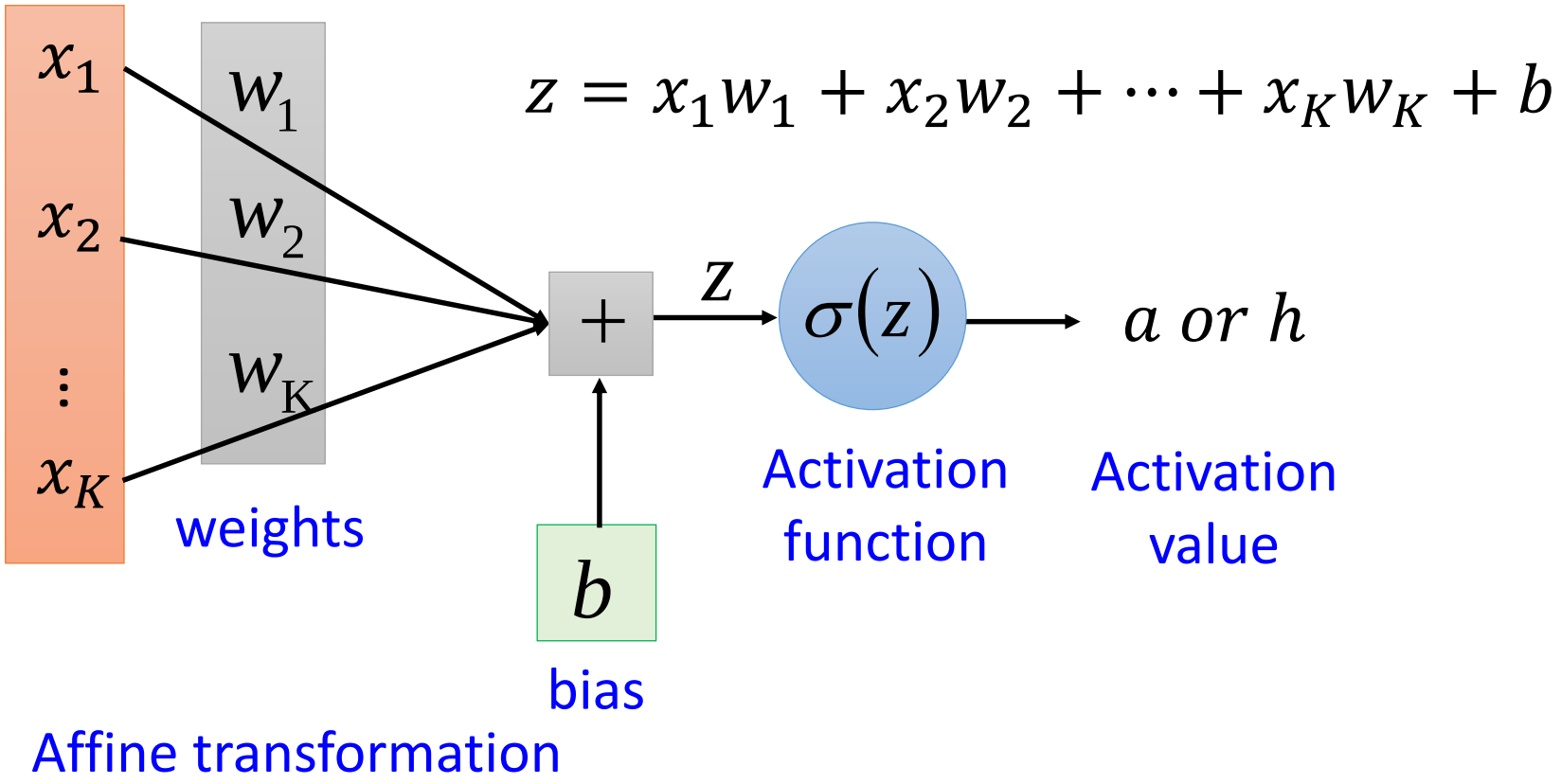
Neural Network

- Deep *feedforward* networks ⇔ Neural networks⇔ Multilayer perceptrons (MLPs)
- No *feedback* connections



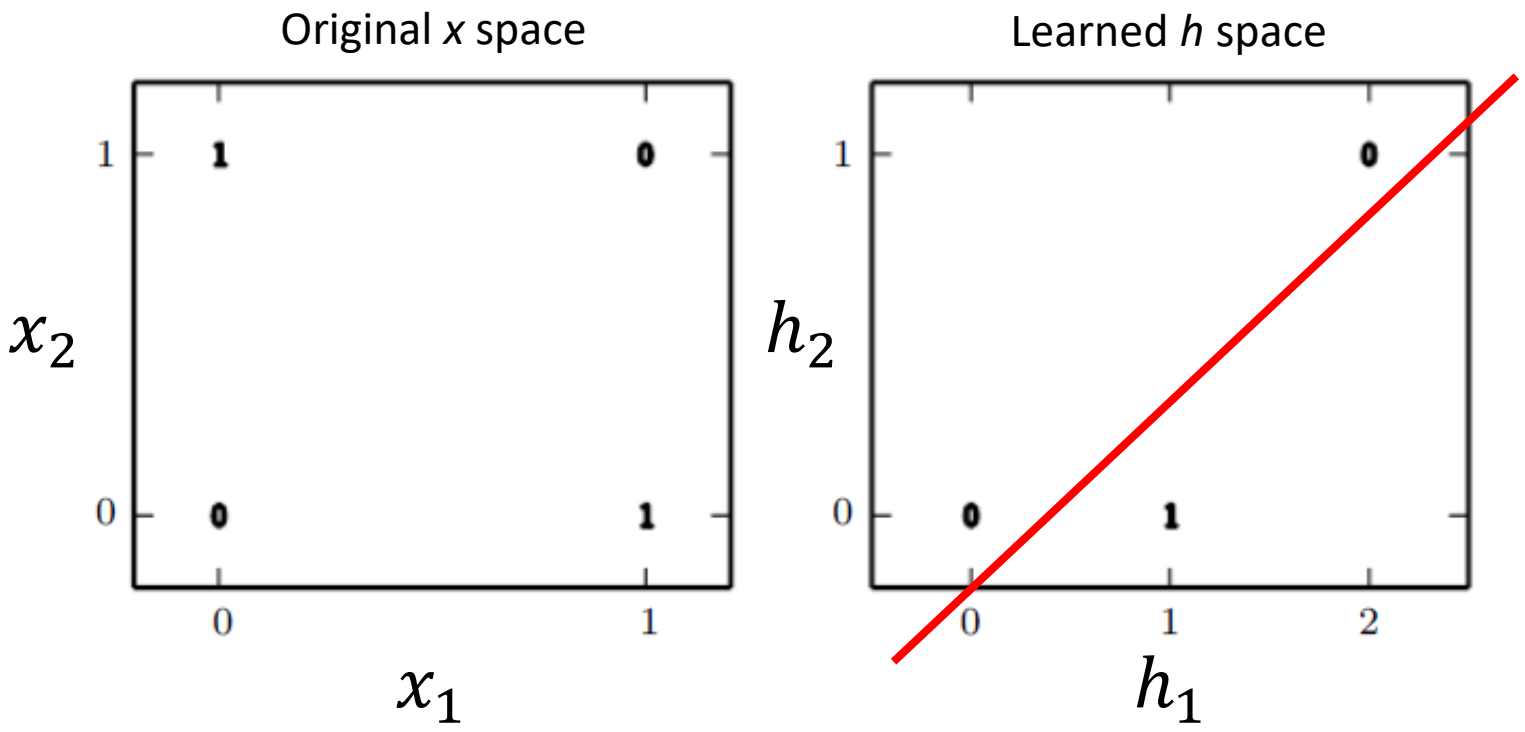
Element of Neural Network

Neuron $f: R^K \rightarrow R$

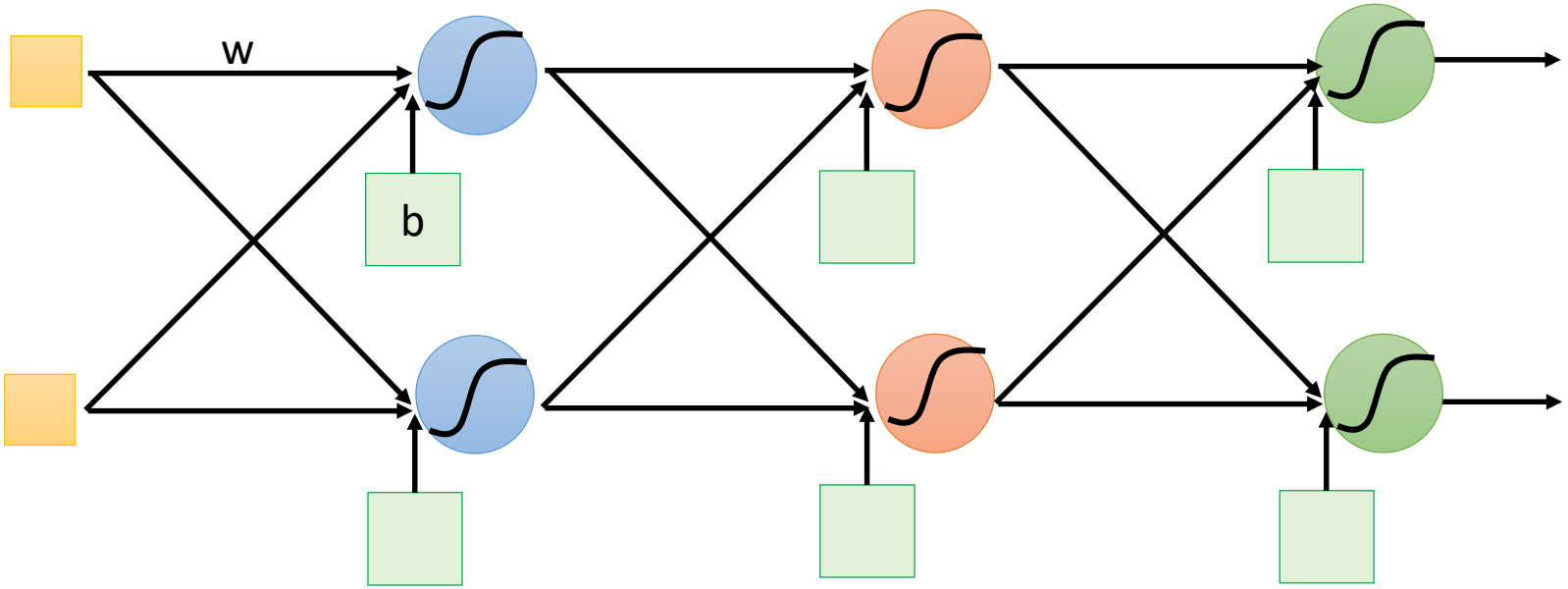


Why non-linear activation necessary…

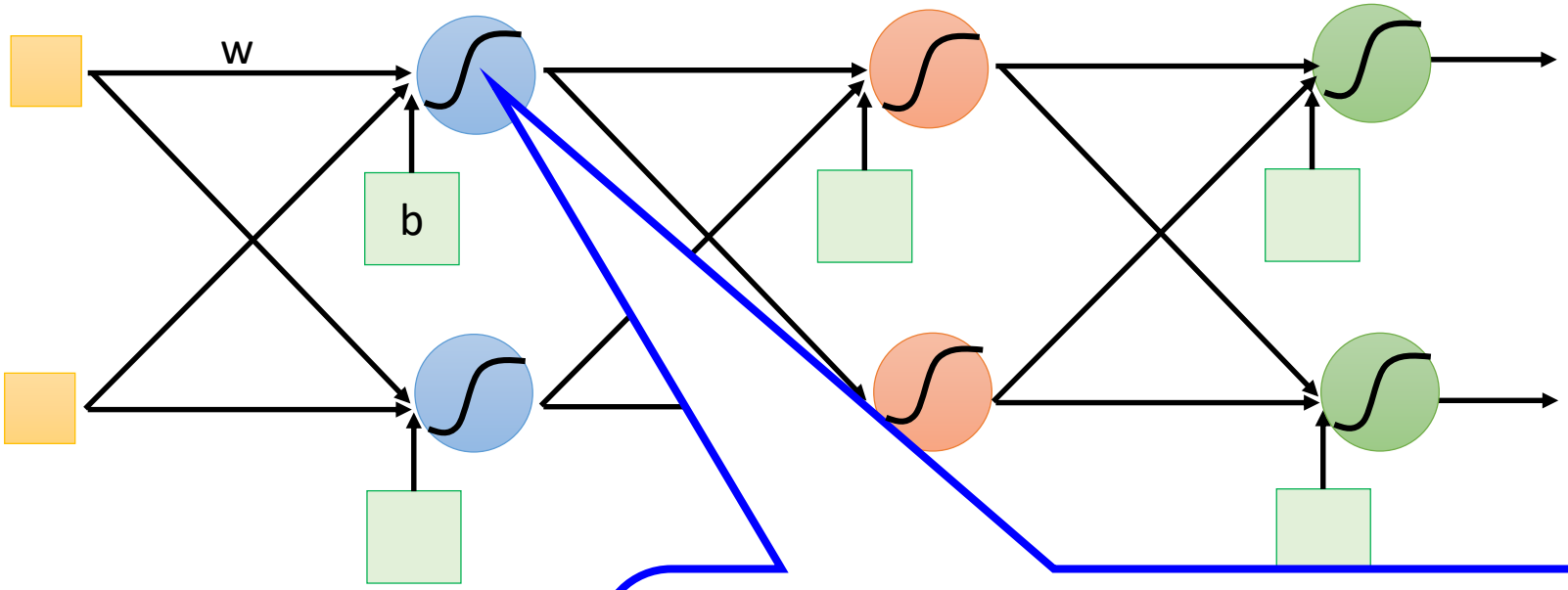
Linear models have many limitations..
Most famously, they cannot learn the **XOR** function..



Example of Neural Network

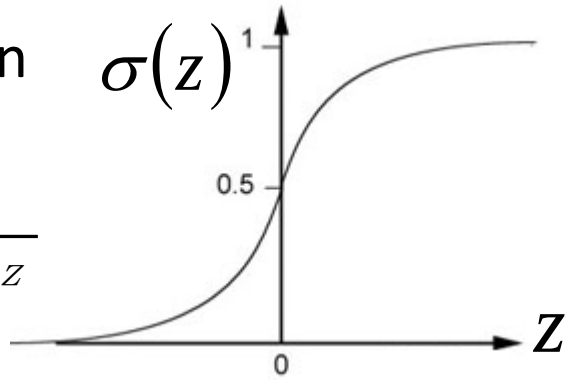


Example of Neural Network



Sigmoid Function

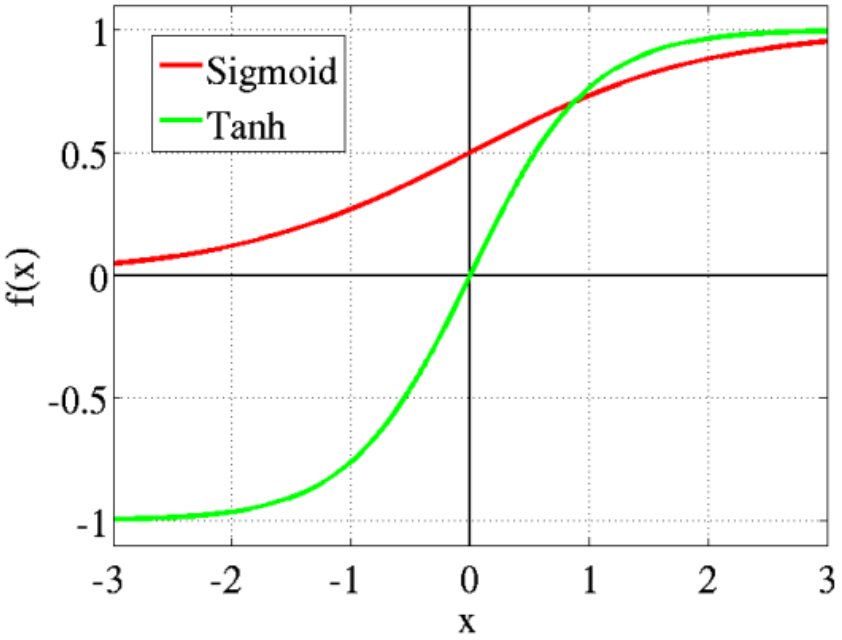
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Other Activation Functions

- **Tangent** $\tanh(z) = 2\sigma(2z) - 1$

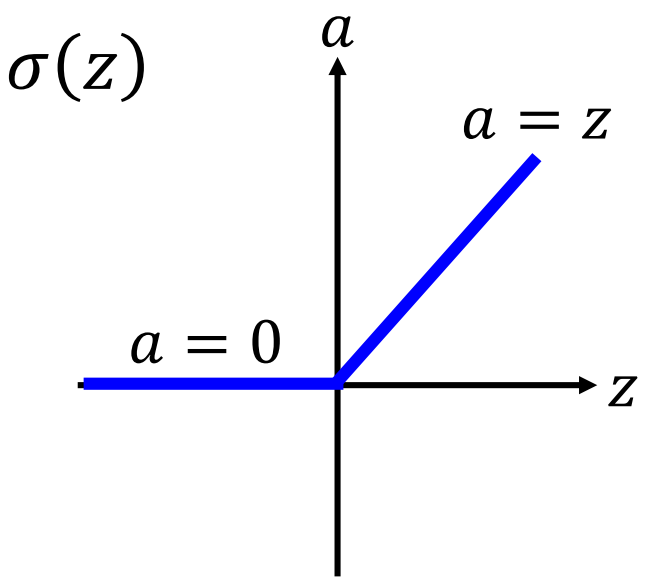
- Zero-centered.
- Squash to range [-1,1].
- Typically performs better than sigmoid.



Saturation. Gradient vanishing…

Other Activation Functions

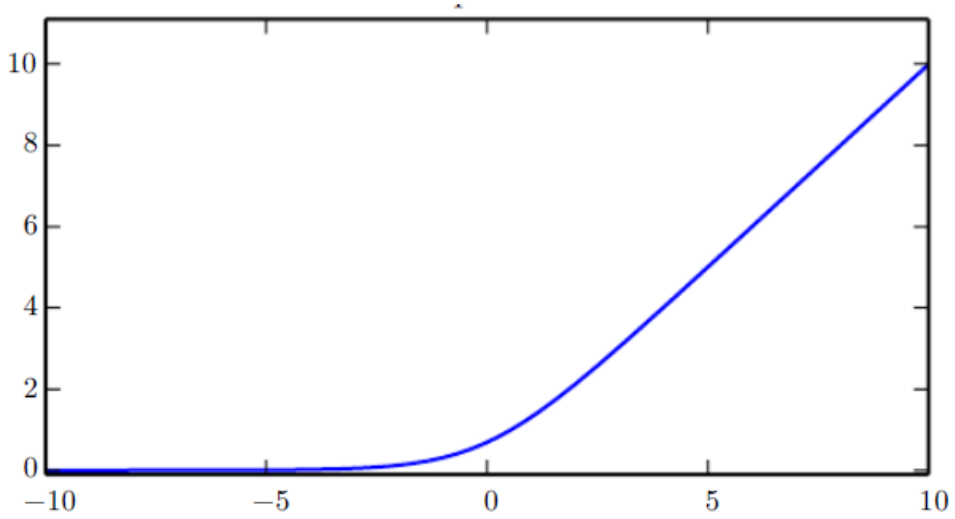
- **Rectified Linear Unit (ReLU)** $\max(0, z)$



- Fast to compute
- Excellent default choice of hidden unit

Other Activation Functions

- **Softplus** $\zeta(z) = \log(1 + \exp(z))$



$\zeta(z)$ Softplus: Soft version of ReLU

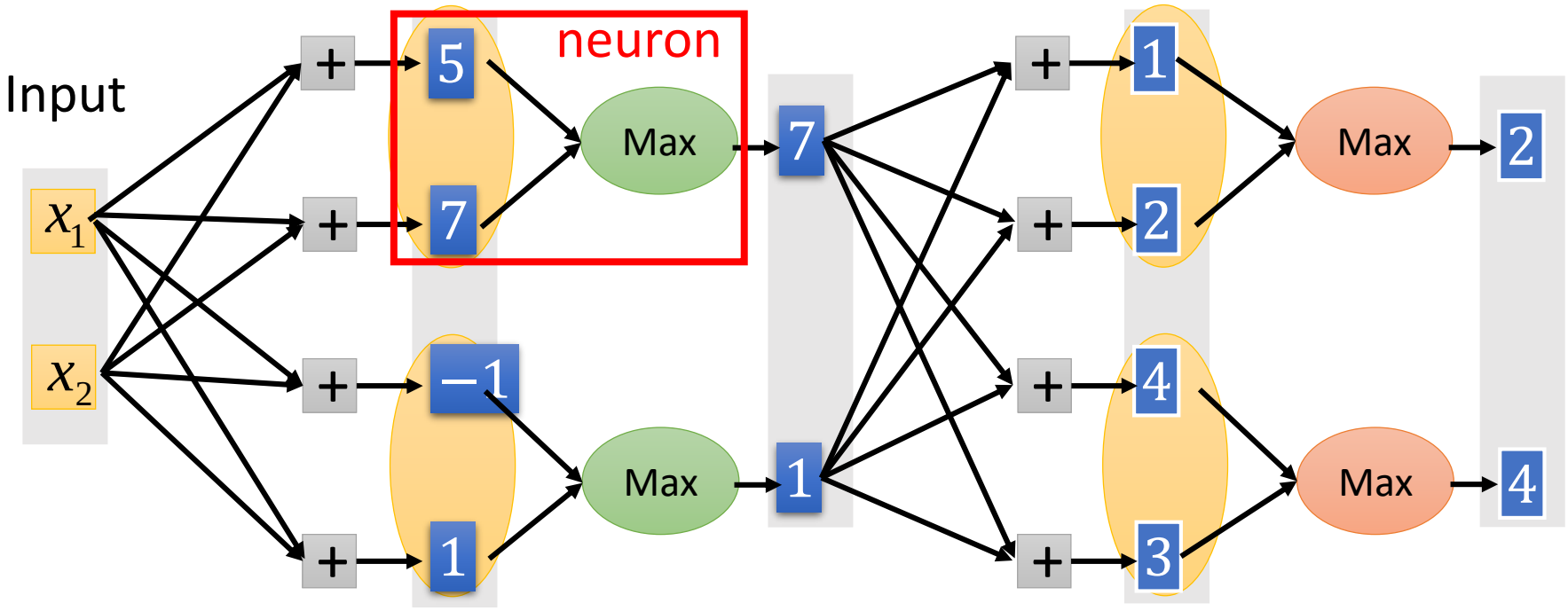
Generally discouraged.

ReLU achieves better results.

Other Activation Functions

- **Maxout**

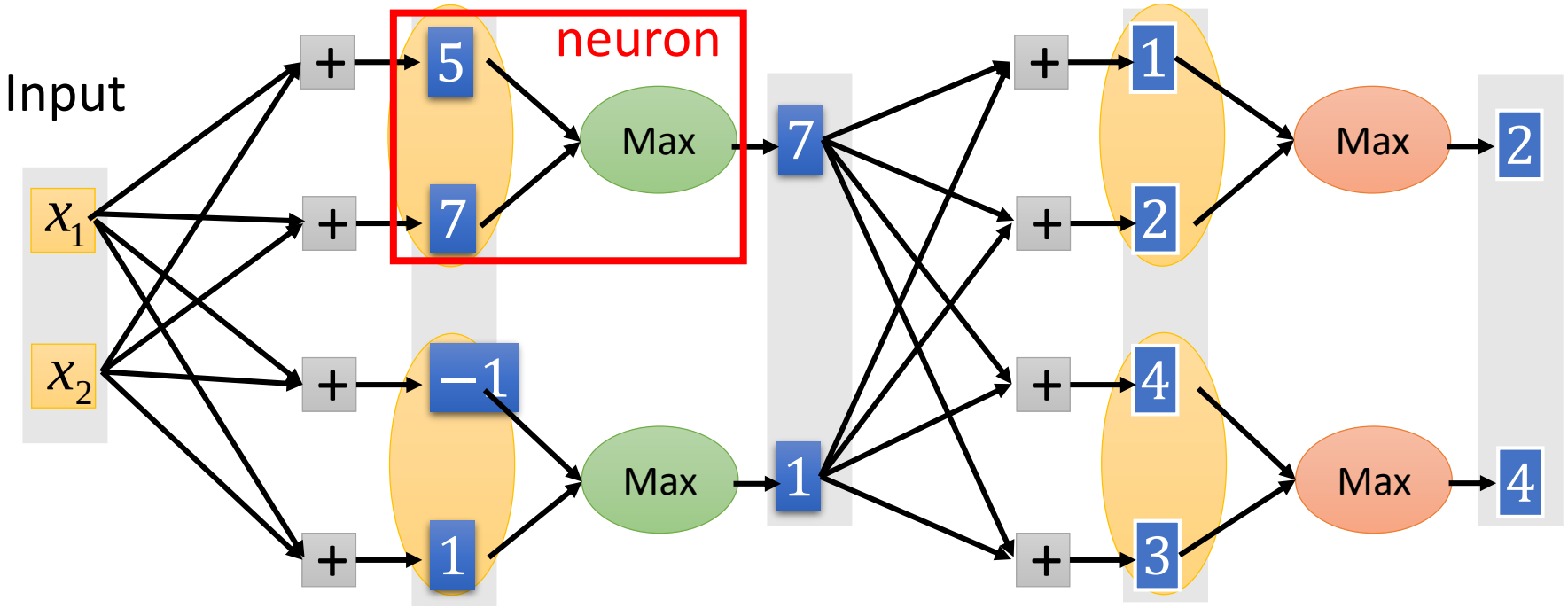
$$max(w_1^T x + b_1, w_2^T x + b_2, \ldots, w_n^T x + b_n)$$



Other Activation Functions

- **Maxout**

$$max(w_1^T x + b_1, w_2^T x + b_2, \ldots, w_n^T x + b_n)$$



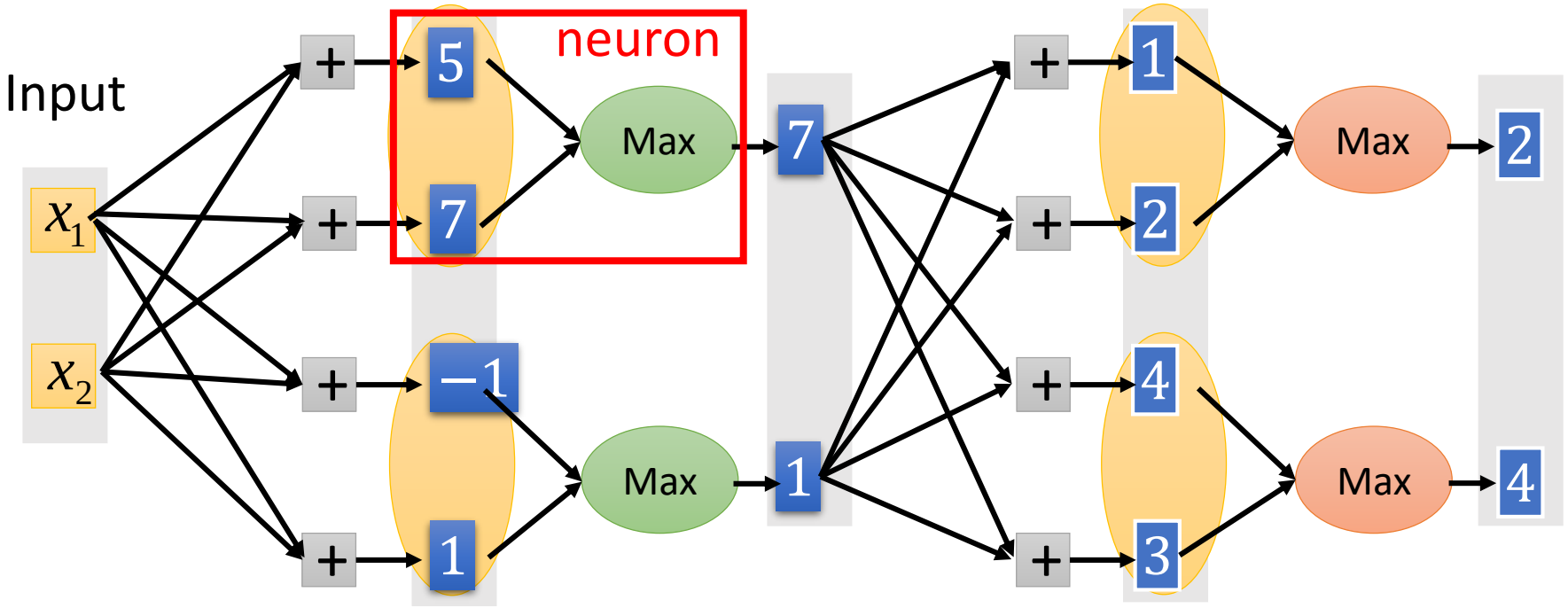
You can have more than 2 elements in a group.

Other Activation Functions

- **Maxout**

ReLU is a special cases of Maxout

$$max(w_1^T x + b_1, w_2^T x + b_2, \ldots, w_n^T x + b_n)$$



You can have more than 2 elements in a group.

Other Activation Functions

- **Maxout**

ReLU is a special cases of Maxout

$$max(w_1^T x + b_1, w_2^T x + b_2, \dots, w_n^T x + b_n)$$

  - Piecewise linear convex function
  - How many pieces depending on how many elements in a group

Other Activation Functions

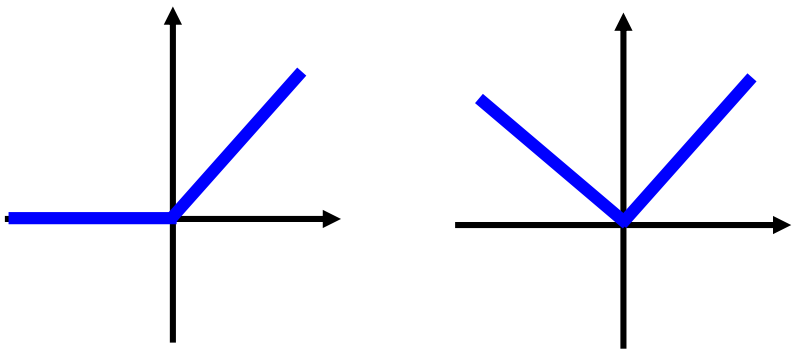
- **Maxout**

ReLU is a special cases of Maxout

$$max(w_1^T x + b_1, w_2^T x + b_2, \dots, w_n^T x + b_n)$$

  - Piecewise linear convex function
  - How many pieces depending on how many elements in a group

2 elements in a group



Other Activation Functions

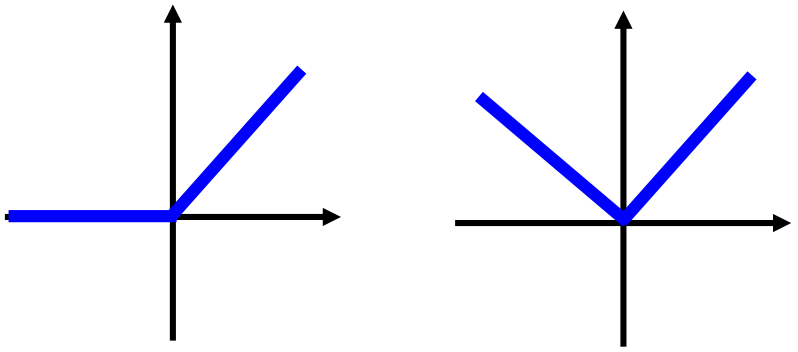
- **Maxout**

ReLU is a special cases of Maxout

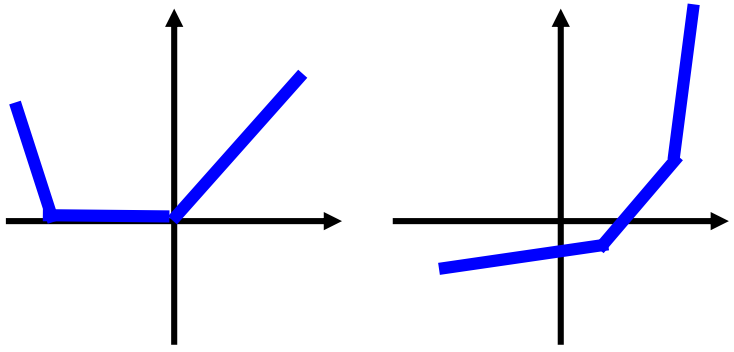
$$max(w_1^T x + b_1, w_2^T x + b_2, \dots, w_n^T x + b_n)$$

  - Piecewise linear convex function
  - How many pieces depending on how many elements in a group

2 elements in a group

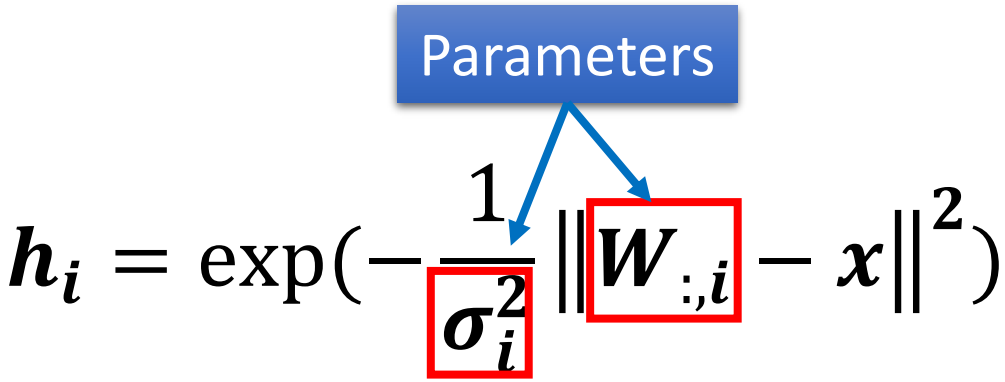


3 elements in a group

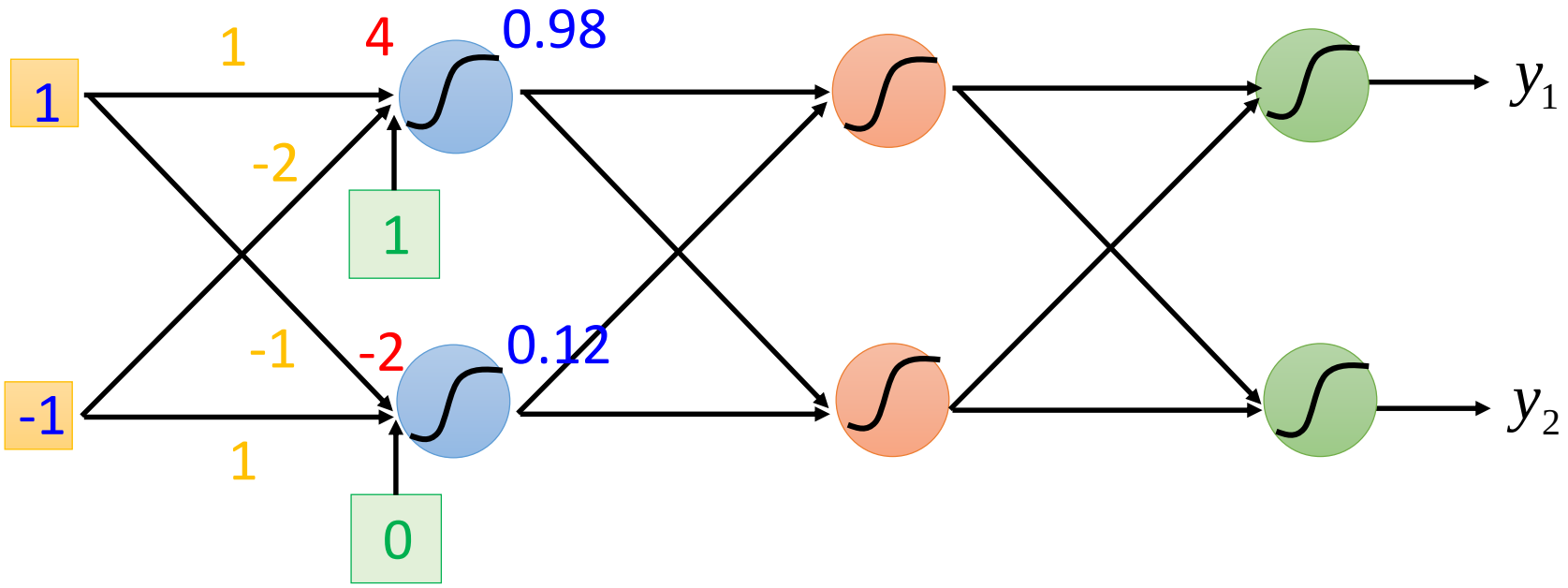


Other Activation Functions

- **Radial Basis Function (RBF)**



Matrix Operation

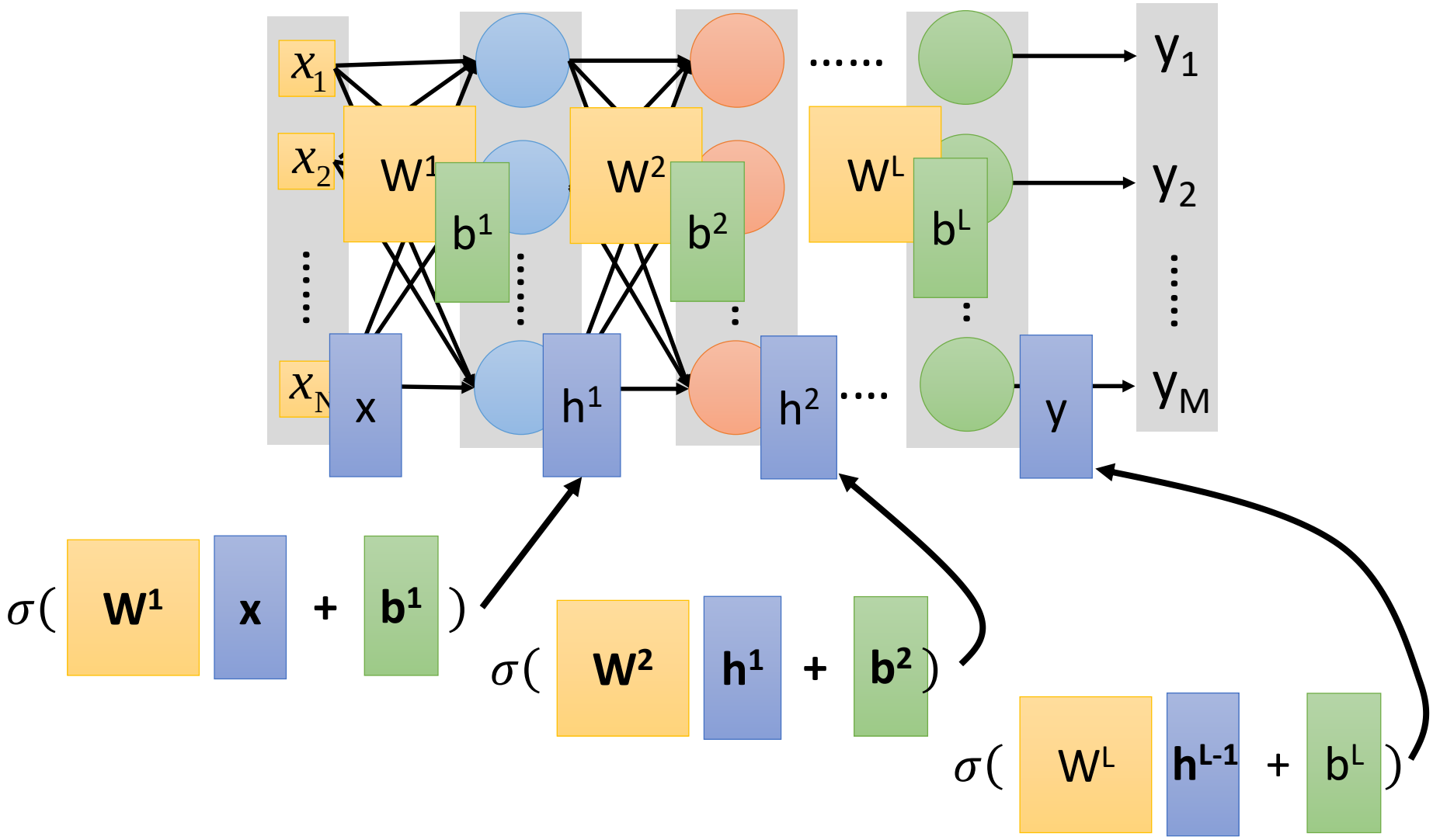


$$\sigma\left(\underbrace{\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}}_{\boldsymbol{z}\ \begin{bmatrix}4\\ -2\end{bmatrix}}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

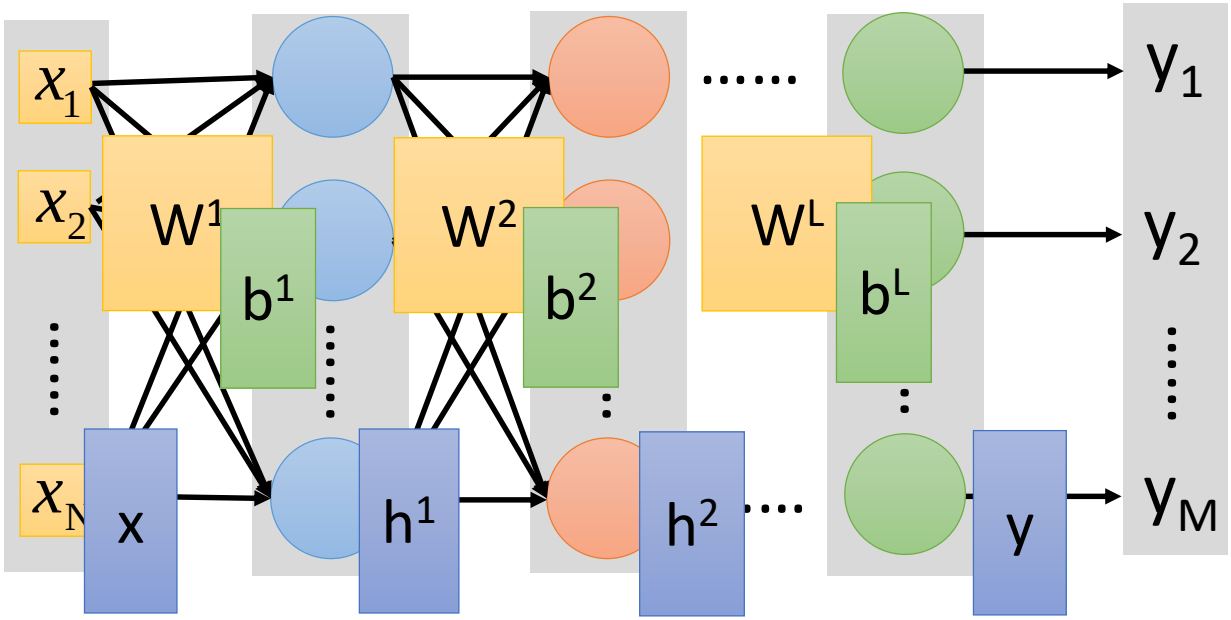
$\mathbf{w}$ $\boldsymbol{x}$ $\boldsymbol{b}$ $\mathbf{h}$

Element-wise!!!

Neural Network



Neural Network



$$\mathbf{y} = f(\mathbf{x})$$

$$= \sigma(\mathbf{W^L} \cdots \sigma(\mathbf{W^2}\,\sigma(\mathbf{W^1}\,\mathbf{x} + \mathbf{b^1}) + \mathbf{b^2}) \cdots + \mathbf{b^L})$$

Output Layer

Cost function is tightly coupled with the output unit.

- **Linear Units**
  - **Regression** — **MSE**

$$\hat{y} = w^T h + b$$

- **Sigmoid Units**
  - **Binary classification** — **Log-likelihood**

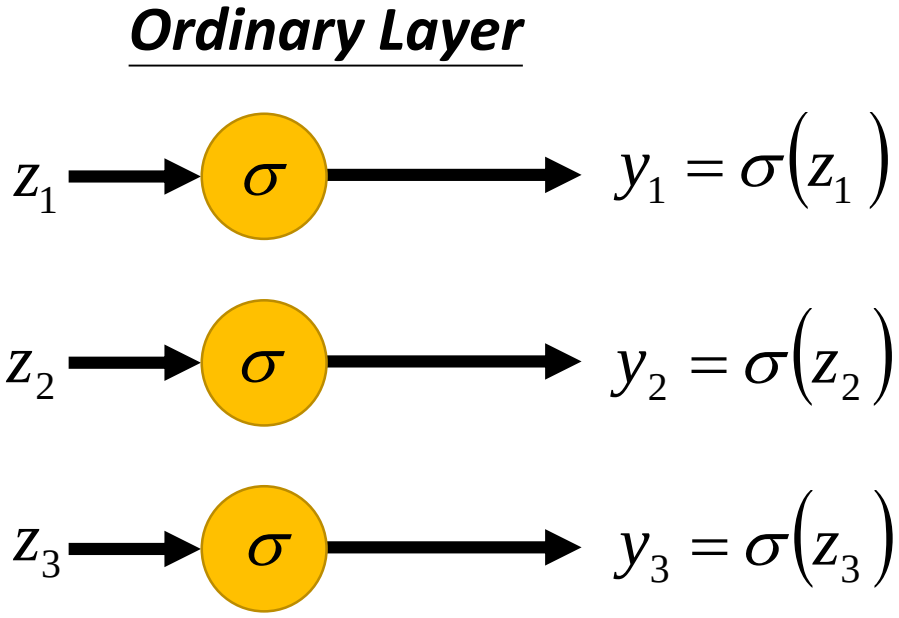
$$\hat{y} = \sigma(w^T h + b)$$

- **Softmax Units**
  - **Multiple mutual classification.** — **Log-likelihood**

$$softmax(\hat{y})_i = e^{z_i} / \sum_j e^{z_j}$$

Softmax

- Softmax layer as the output layer



In general, the output of network can be any value.

May not be easy to interpret .

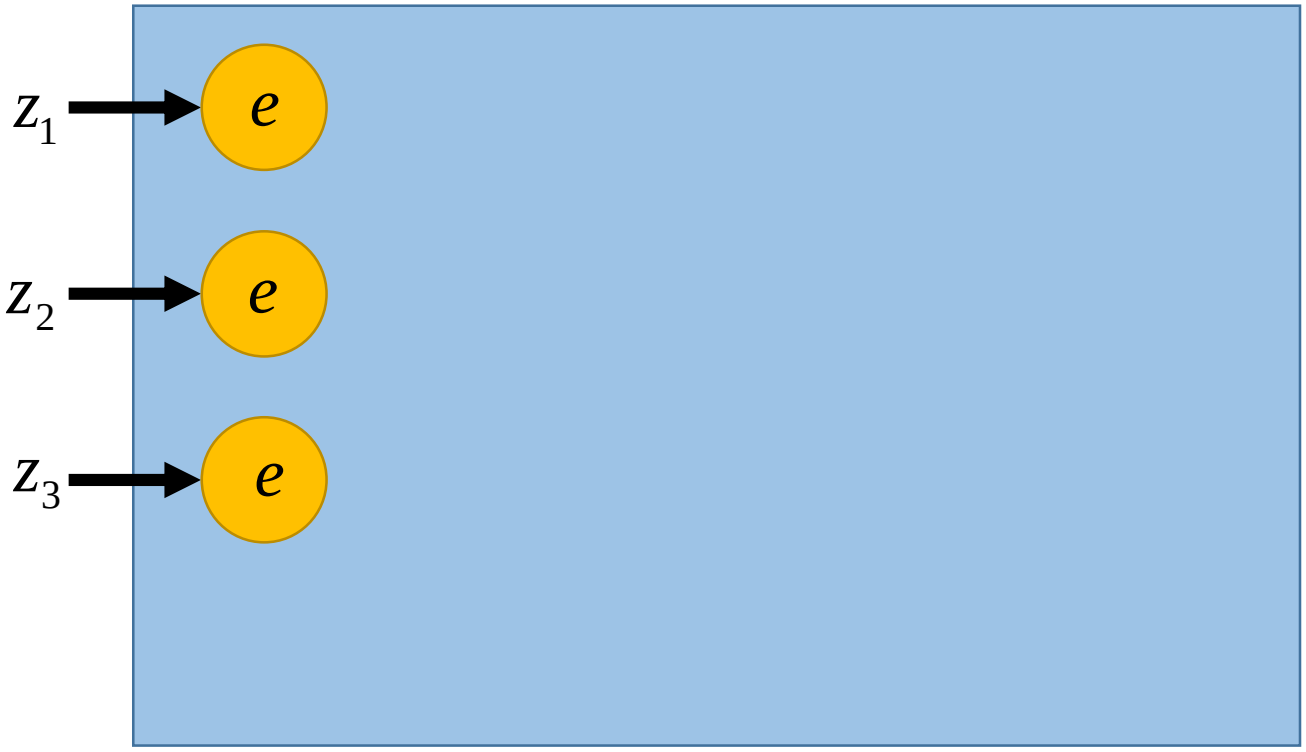
Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer

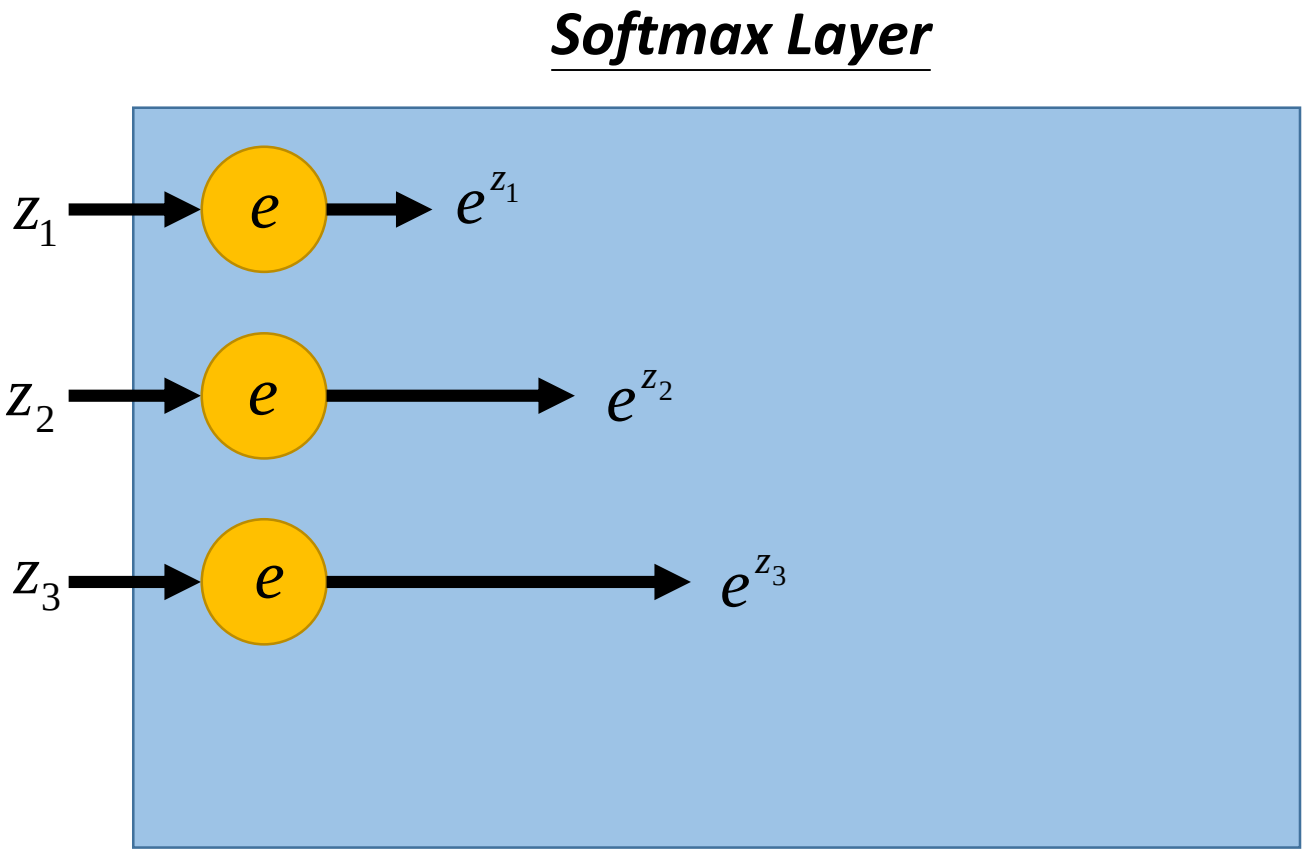


Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$



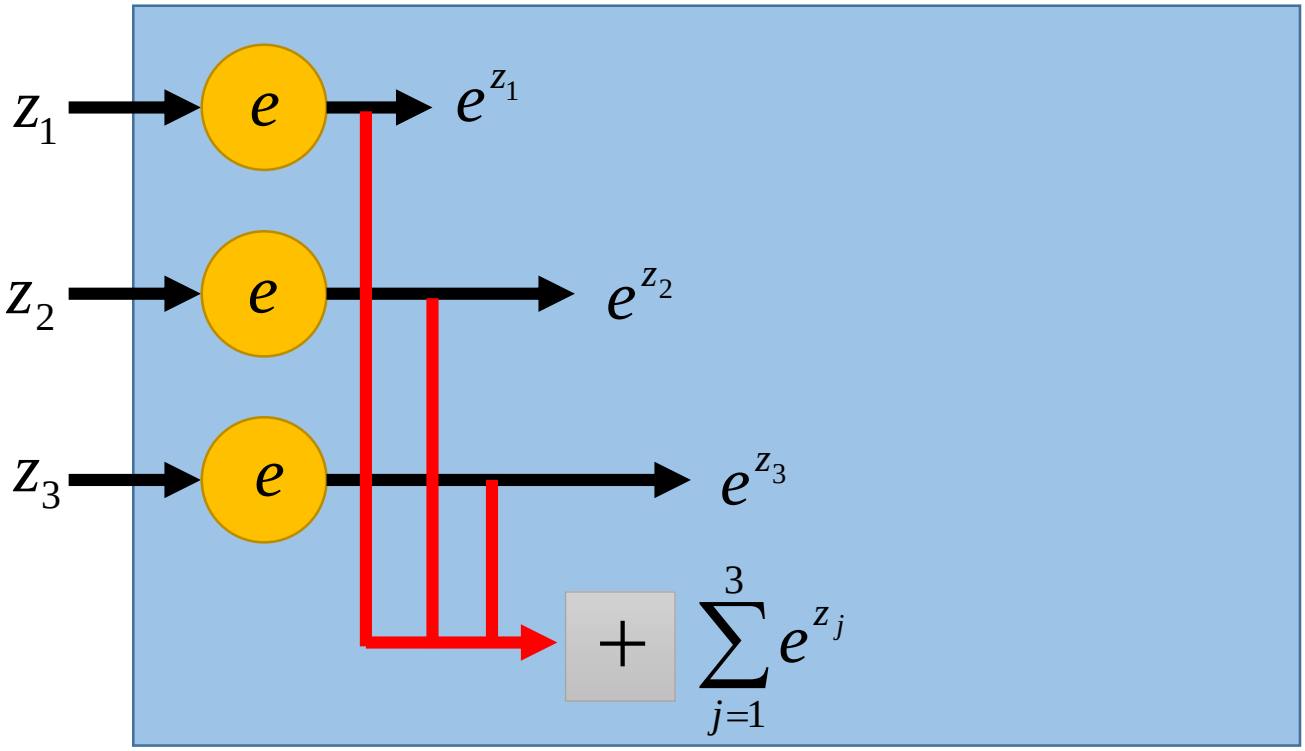
Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



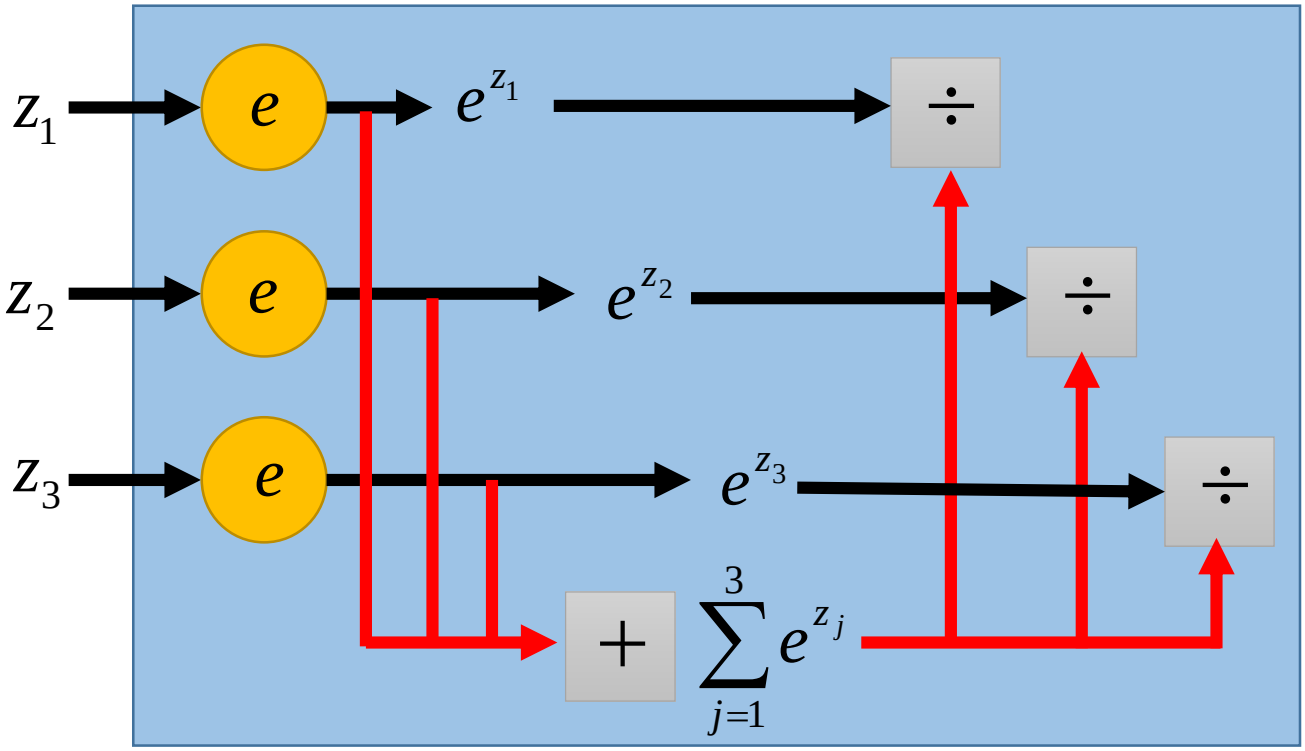
Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer

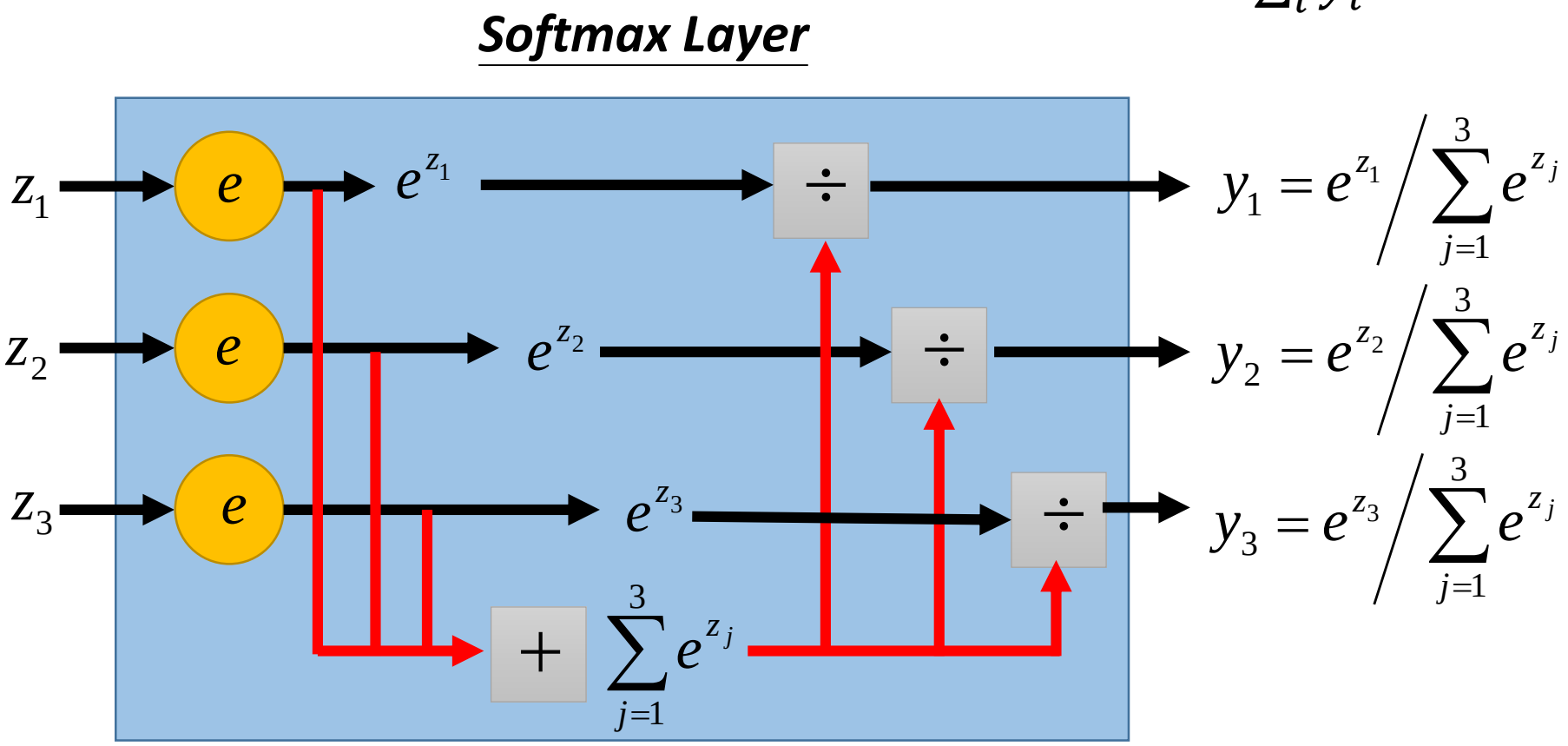


Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

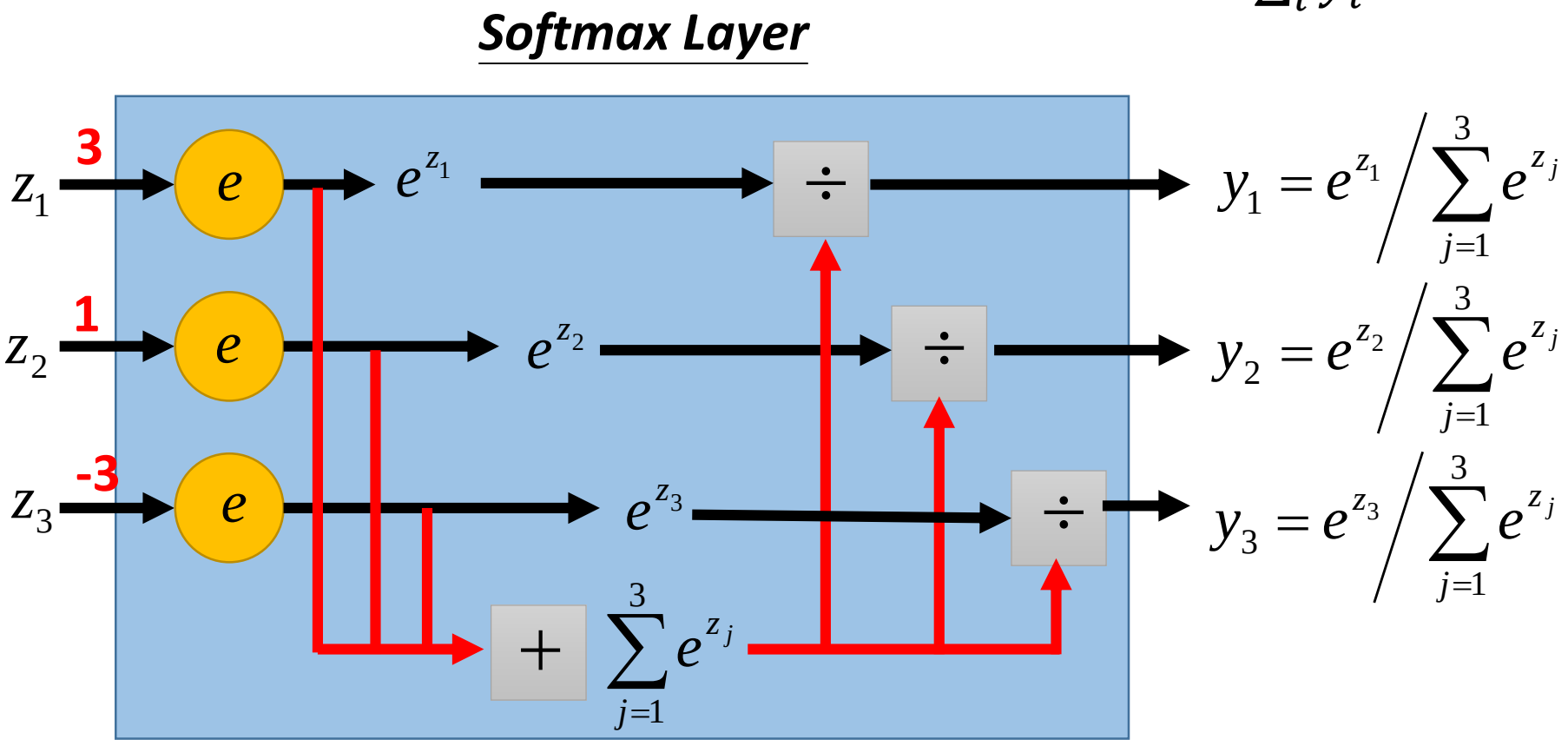


Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

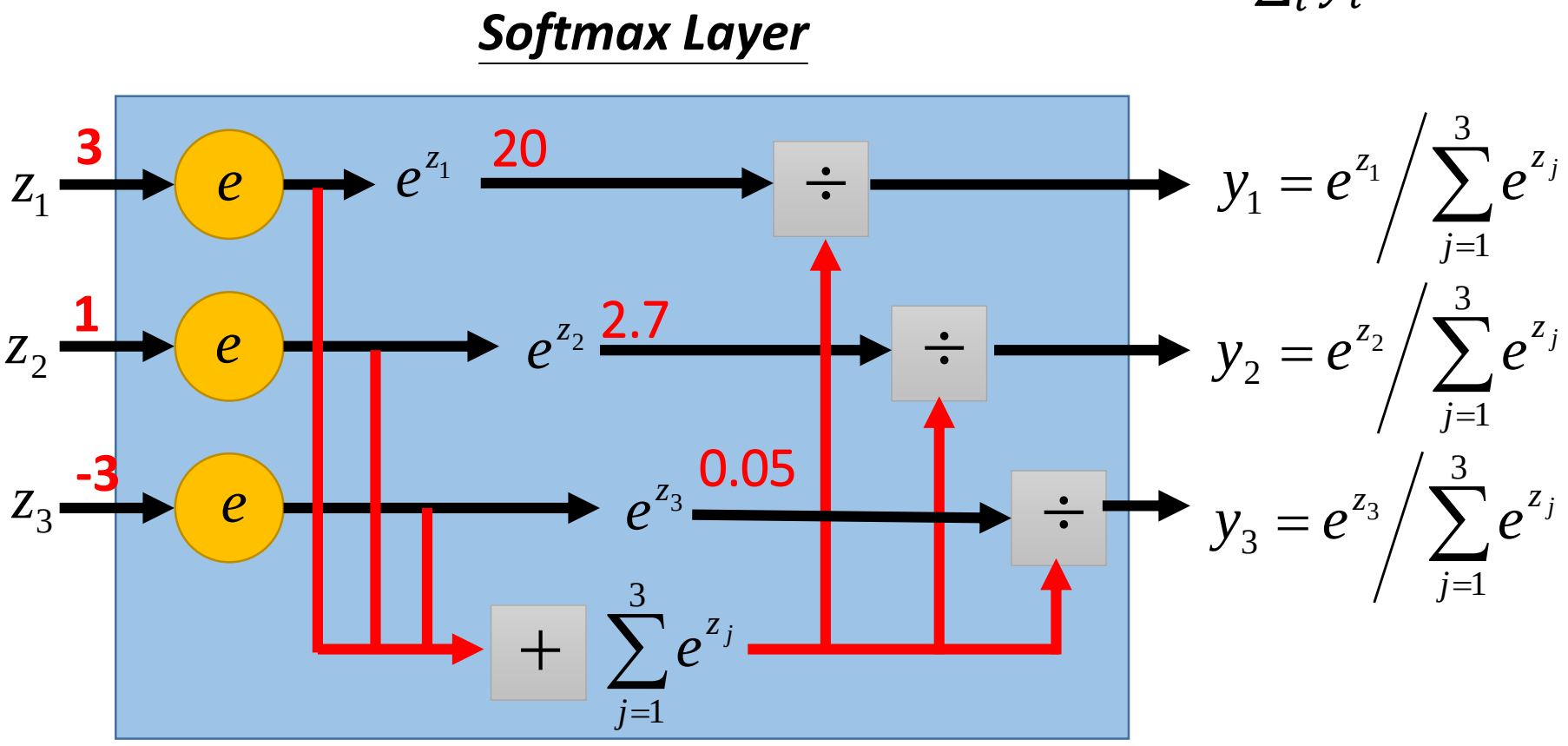


Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

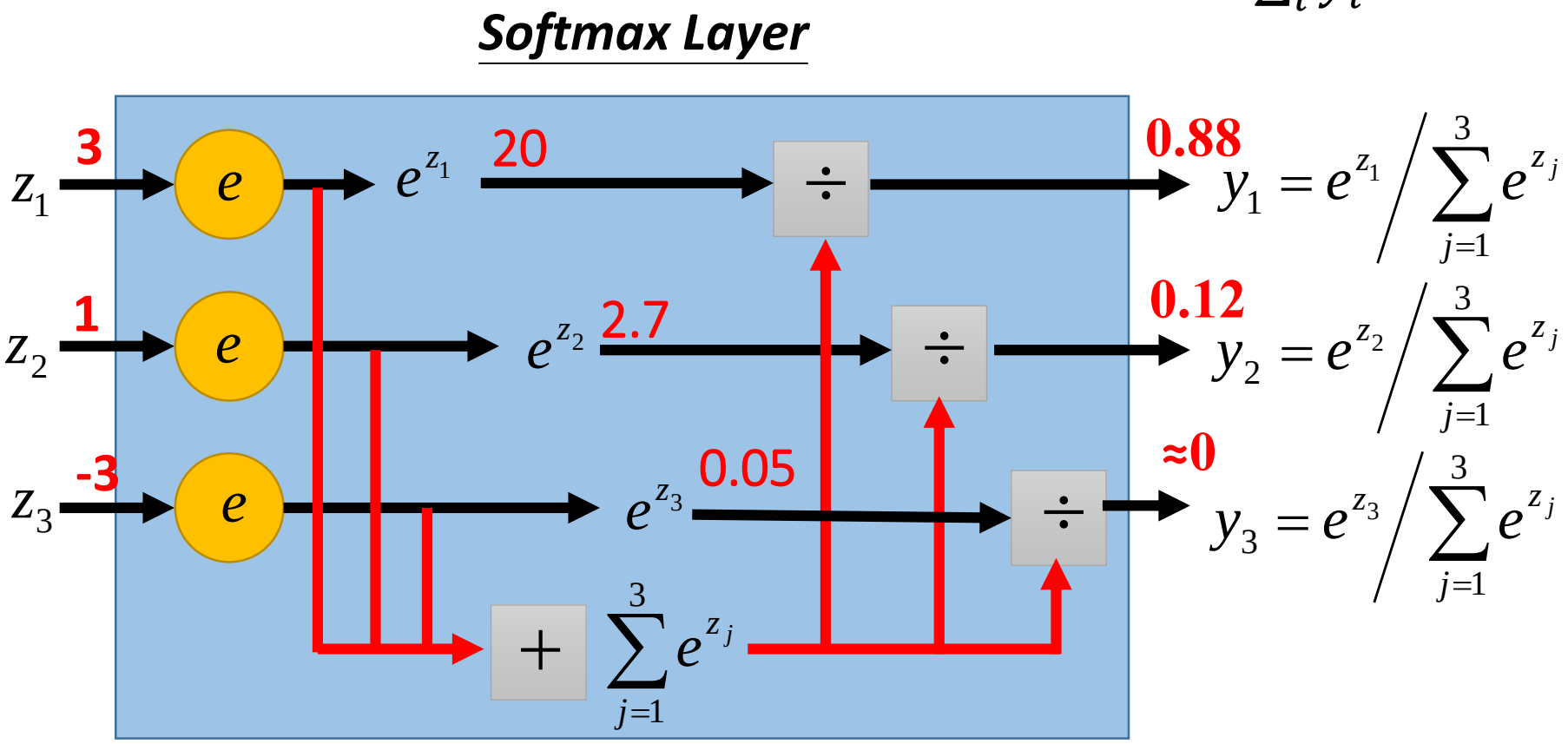


Softmax

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$



Short Summary

Neural Networks

- No need to do feature engineering.
- Input layer, hidden layers, output layer.

Neuron $f: R^K \rightarrow R$

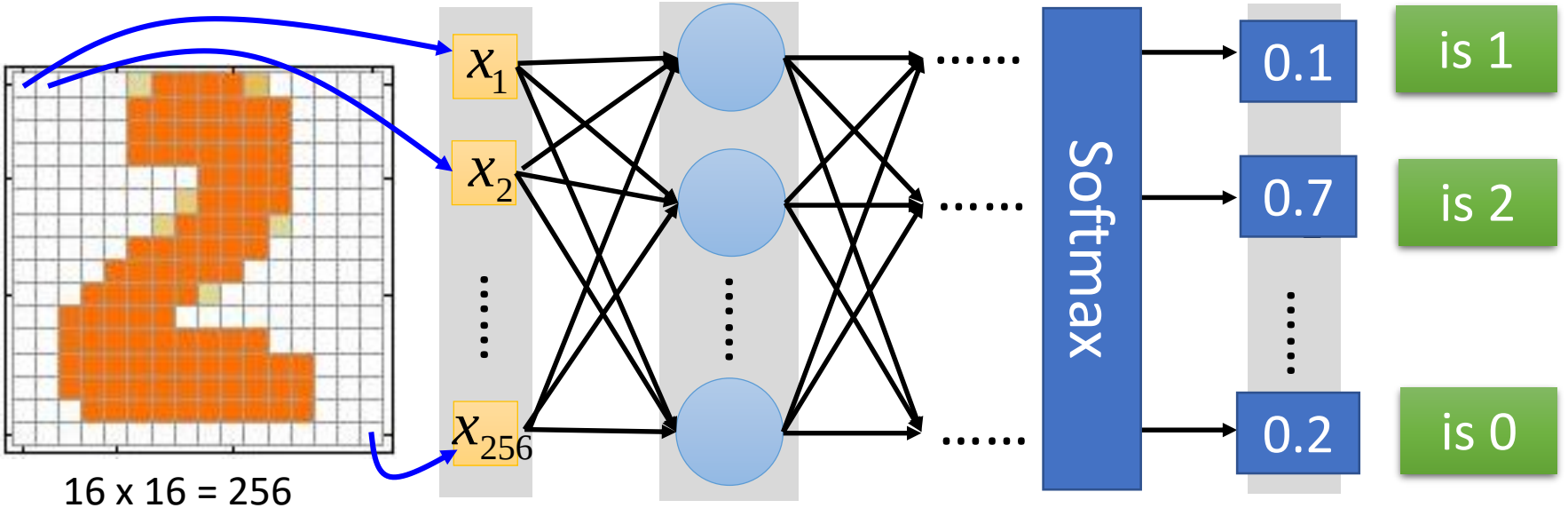
- Affine transformation (linear)+ activation function (non-linear).
- Activation function (element-wise):
  - Sigmoid, Tangent, ReLU, Softplus, Maxout
  - Radial Basis Function (RBF)

Output layer

- Linear unit
  - Regression
- Sigmoid
  - Binary classification
- Softmax (argmax).
  - Multiple mutual classification.

How to Set Network Parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1

No ink → 0

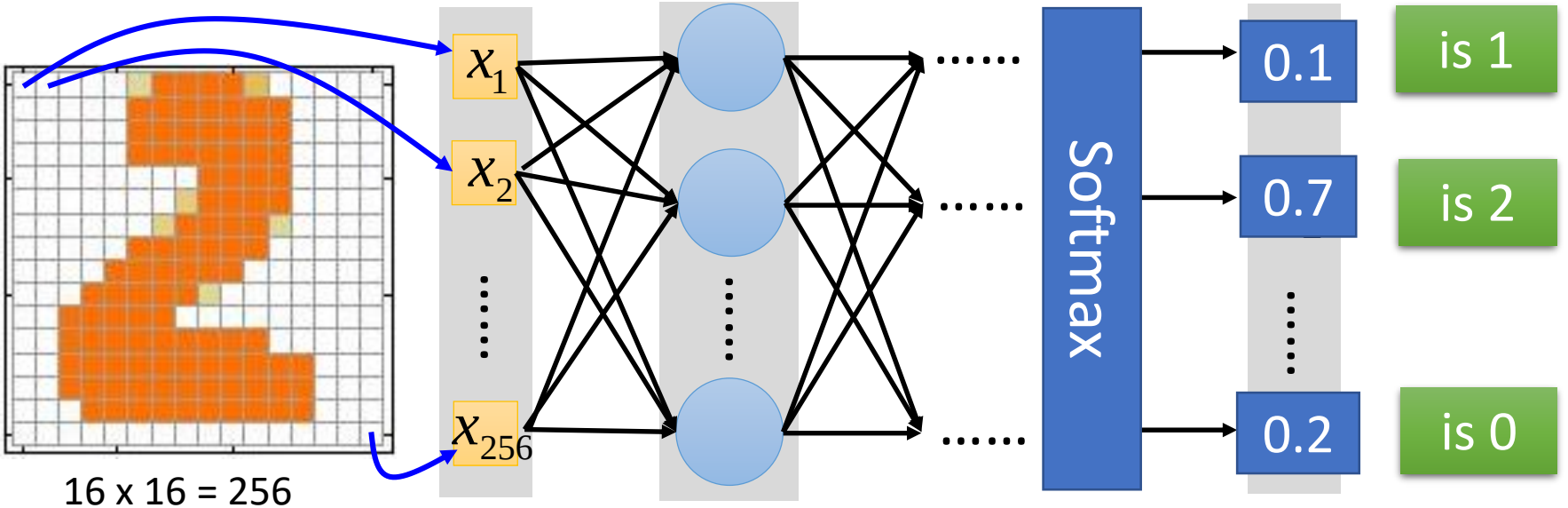
Set the network parameters θ such that ……

Input:  → y_1 has the maximum value

Input:  → y_2 has the maximum value

How to Set Network Parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1

No ink → 0

Set the network parameters θ such that ……

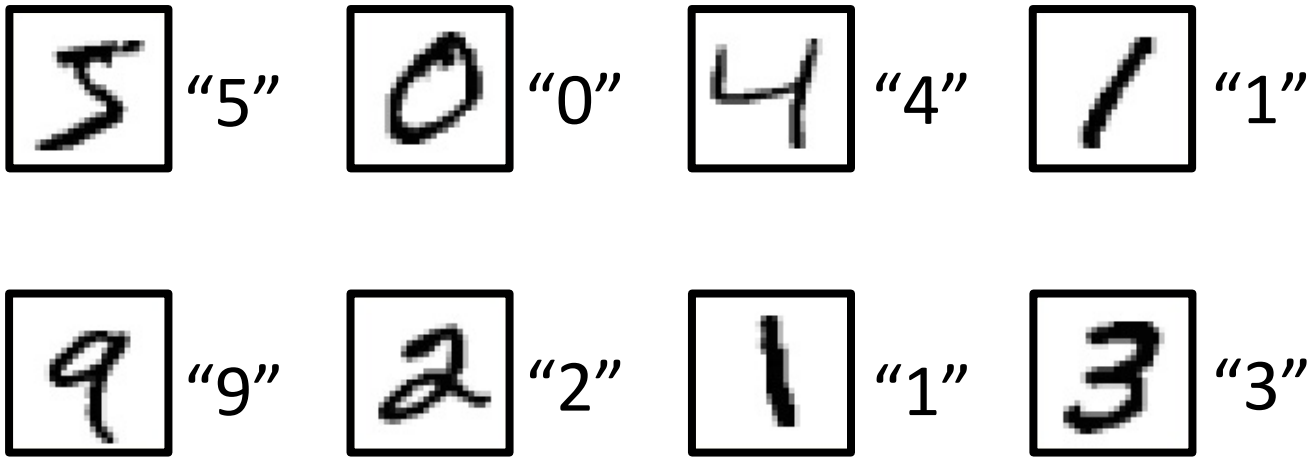
Input: m value

Input: y_2 has the maximum value

How to let the neural network achieve this

Training Data

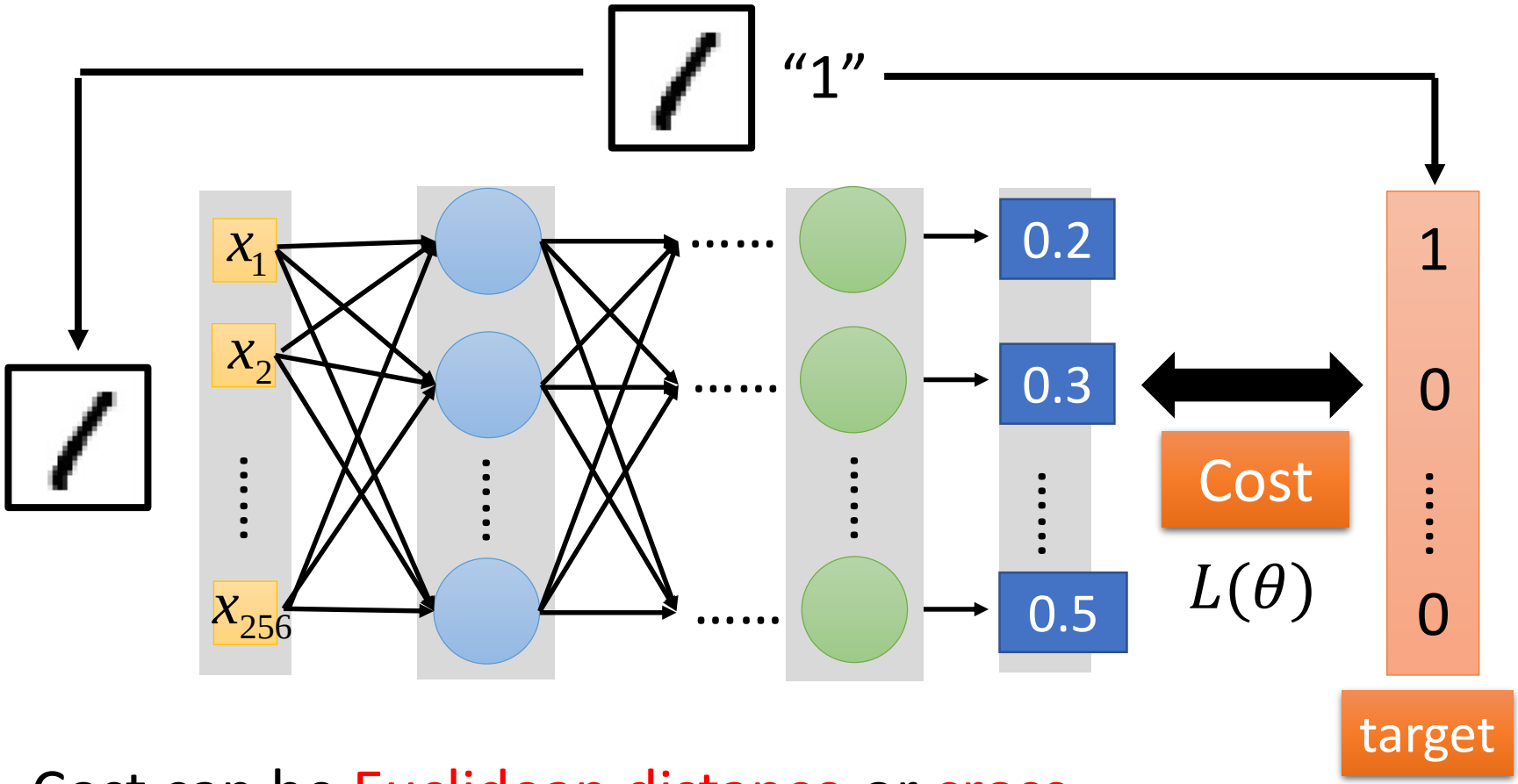
- Preparing training data: images and their labels



Using the training data to find the network parameters.

Cost

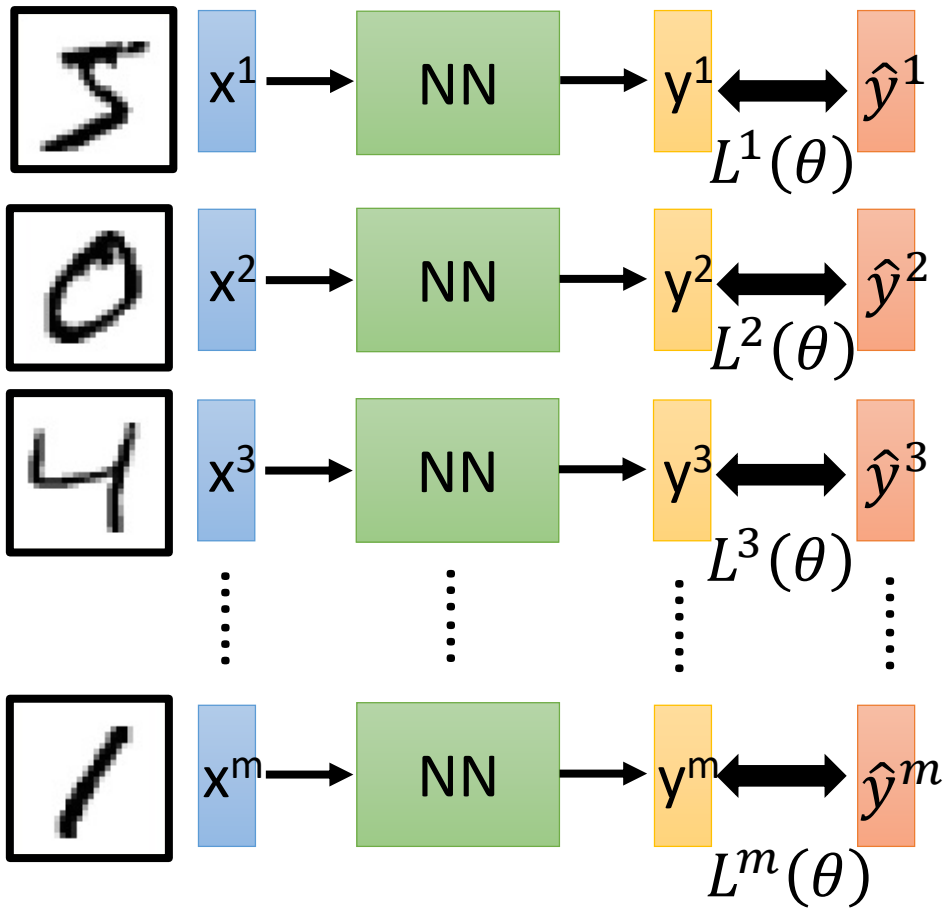
Given a set of network parameters θ, each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data …

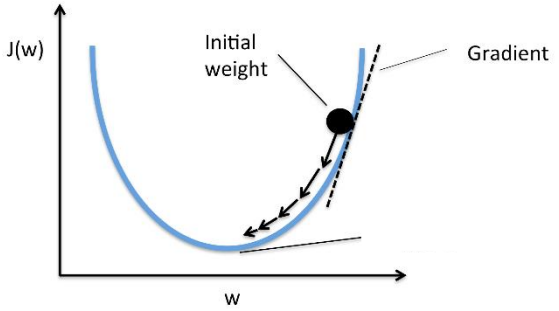


Total Cost:

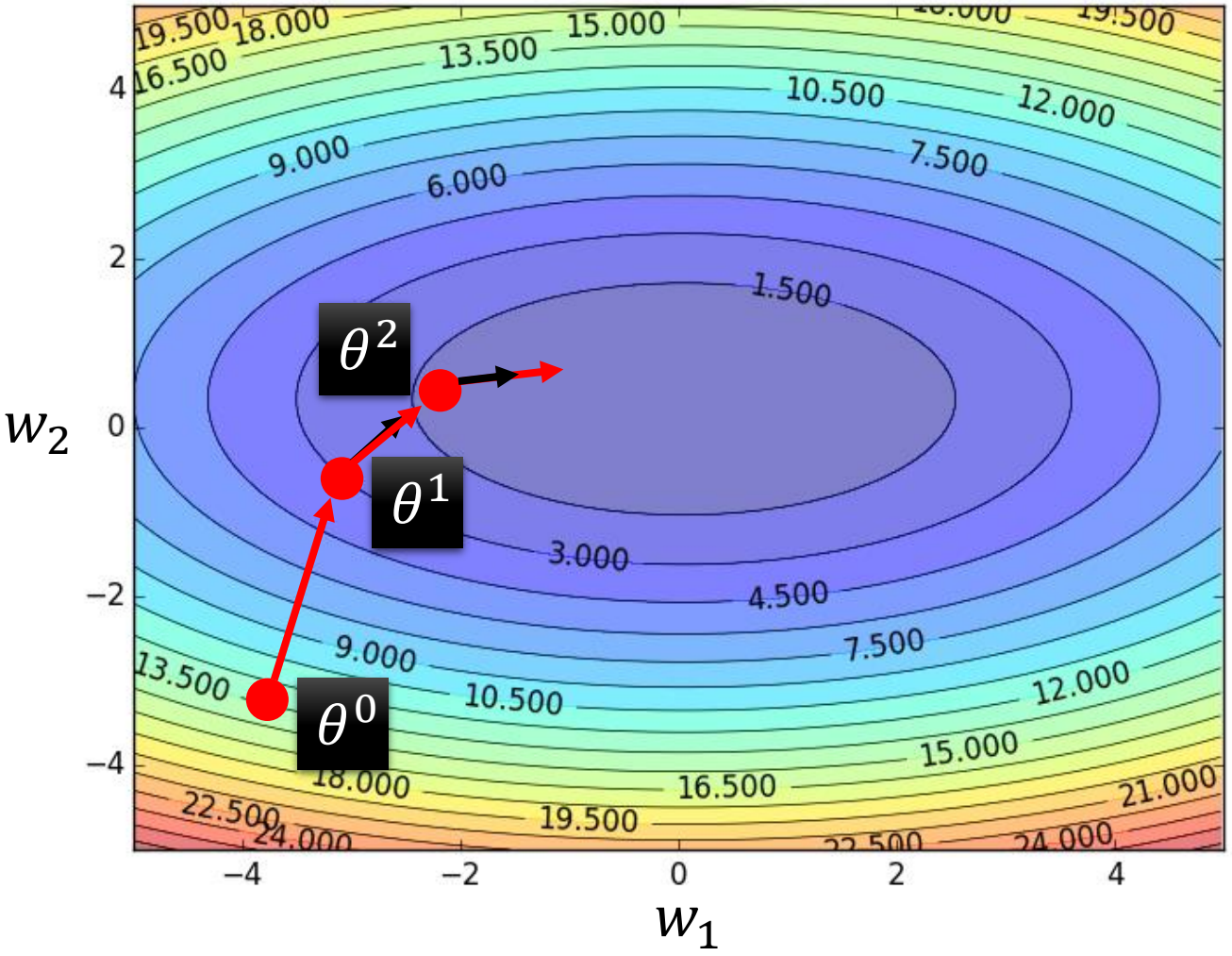
$$L(\theta) = \sum_{r=1}^{m} L^r(\theta)$$

Find the network parameters θ^* that minimize this value

Gradient Descent



Error Surface



Randomly pick a starting point θ^0

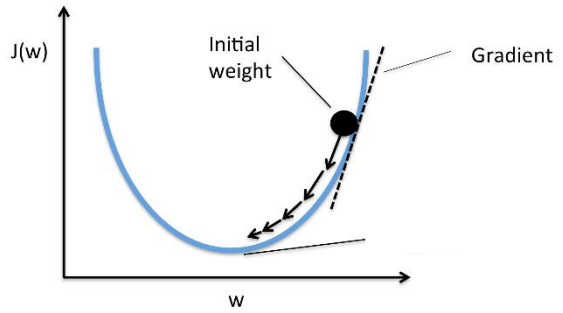
Compute the negative gradient at θ^0

$\rightarrow -\nabla L(\theta^0)$

Times the learning rate η

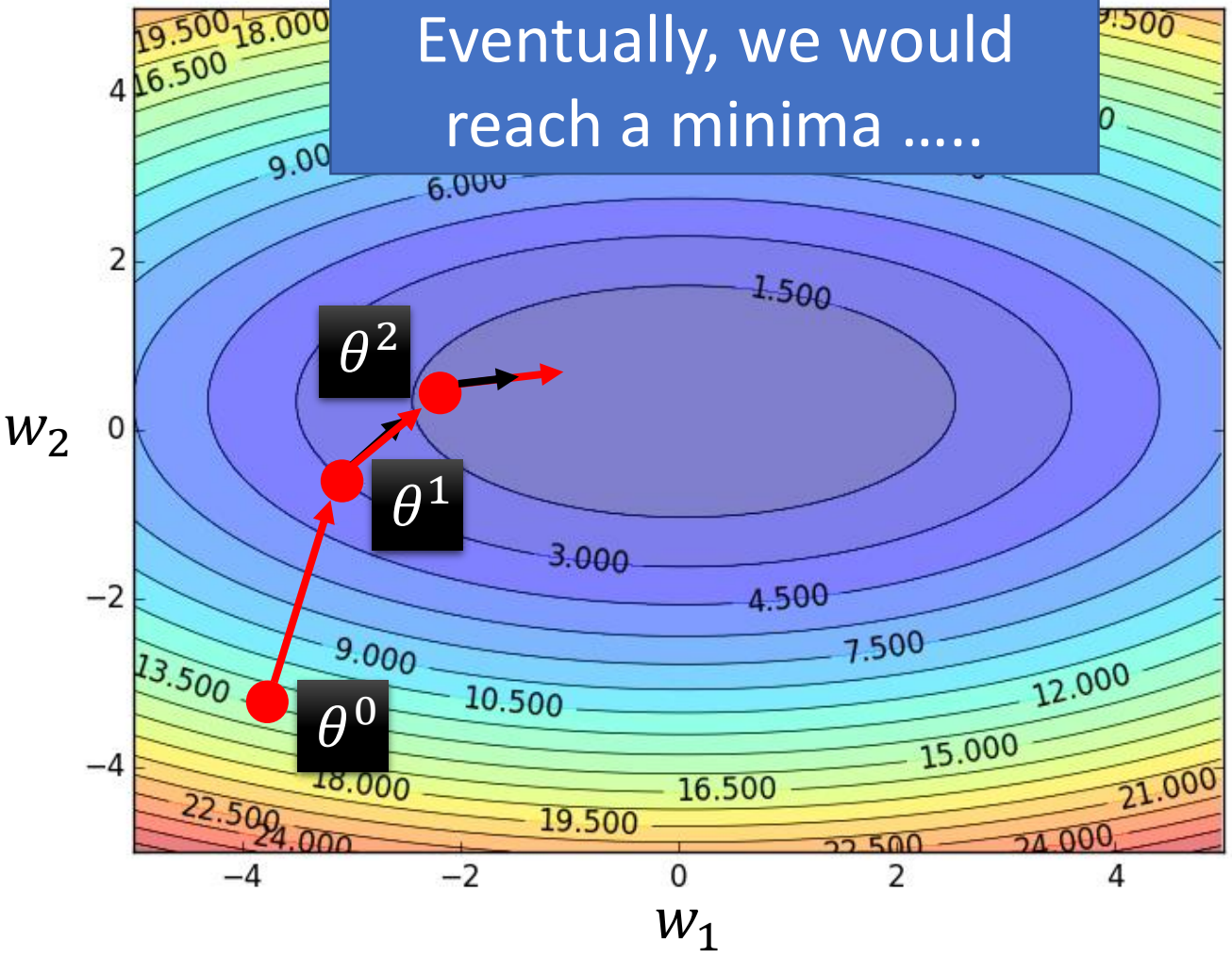
$\rightarrow -\eta \nabla L(\theta^0)$

Gradient Descent



Error Surface

Eventually, we would reach a minima .....



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

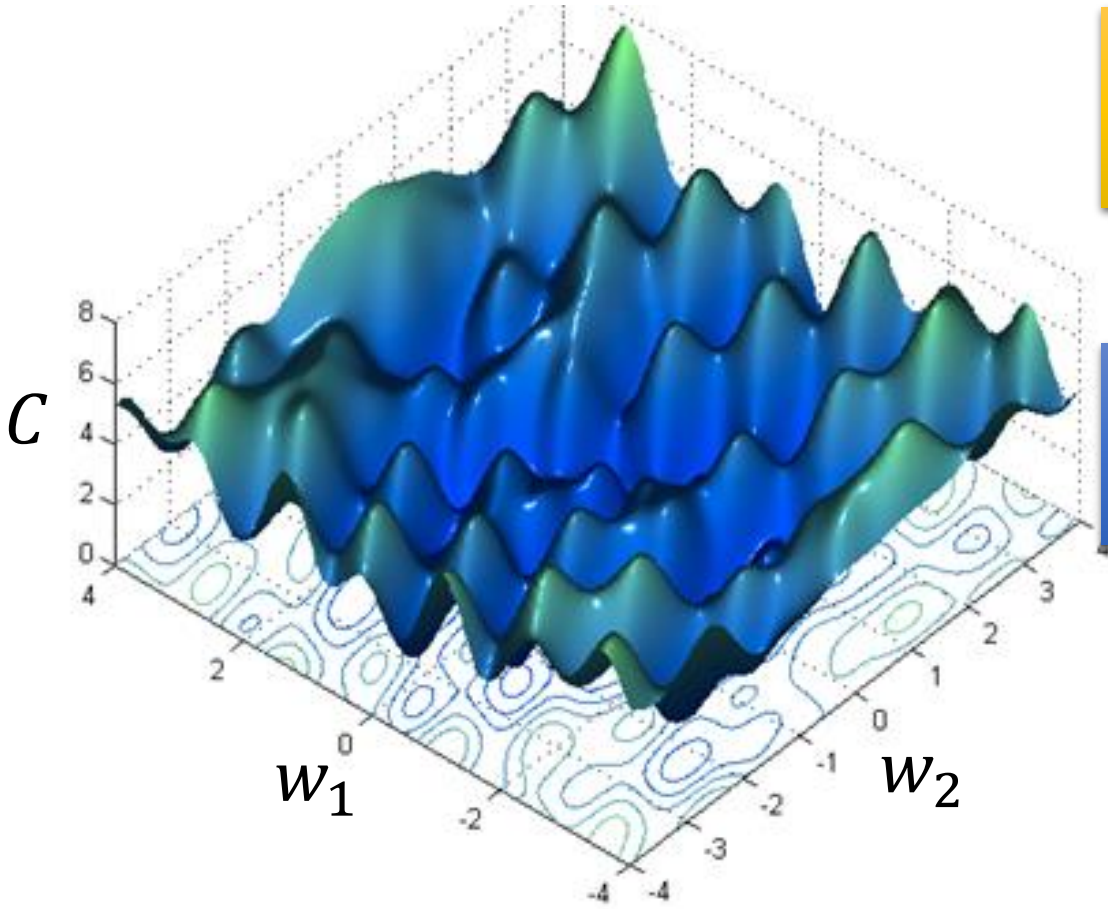
$\rightarrow -\nabla L(\theta^0)$

Times the learning rate η

$\rightarrow -\eta \nabla L(\theta^0)$

Local Minima

- Gradient descent never guarantee global minima



Different initial point θ^0

Reach different minima, so different results

Optimization

Stochastic Gradient Descent (on mini-batches):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in (0, 1)$$

Stochastic Gradient Descent with Momentum:

$$\theta \leftarrow \theta - \eta \Delta$$

$$\Delta \leftarrow 0.9\Delta + \frac{\partial L}{\partial \theta}$$

Note: there are many other variants...

Example

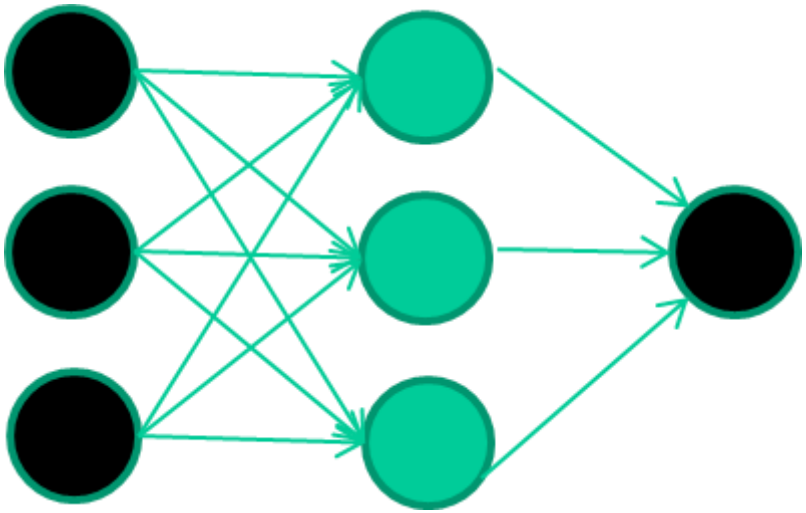
Training the neural network

Initialise with random weights

A dataset

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



Example

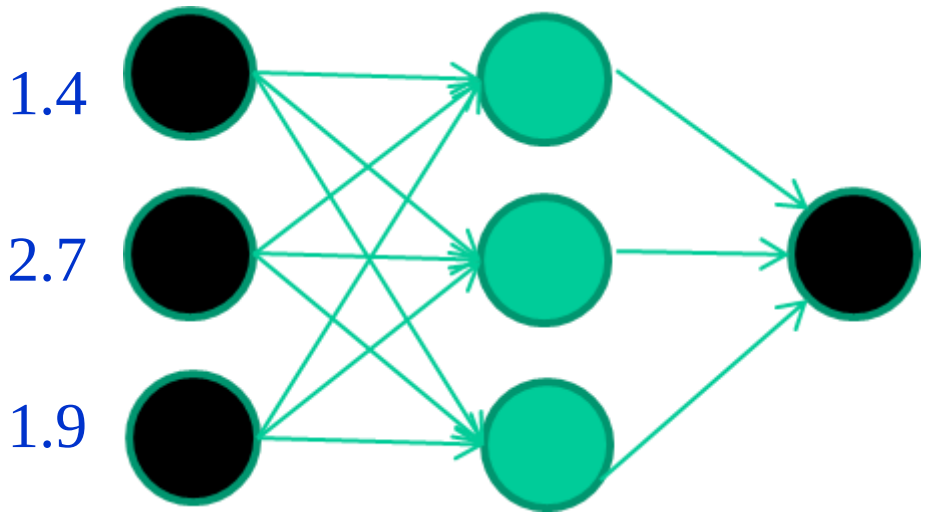
Training the neural network

Initialise with random weights

- **Present a training pattern**

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



Example

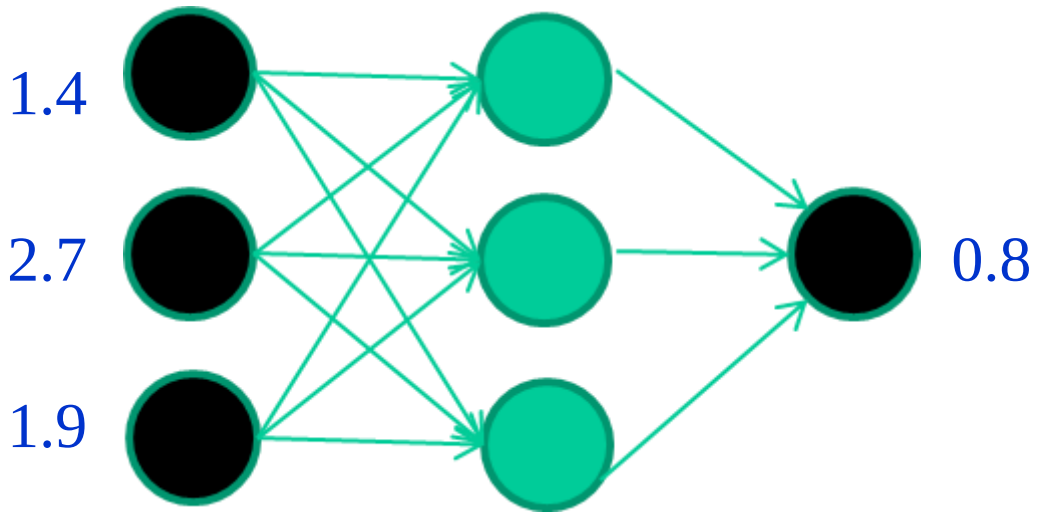
Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**



Example

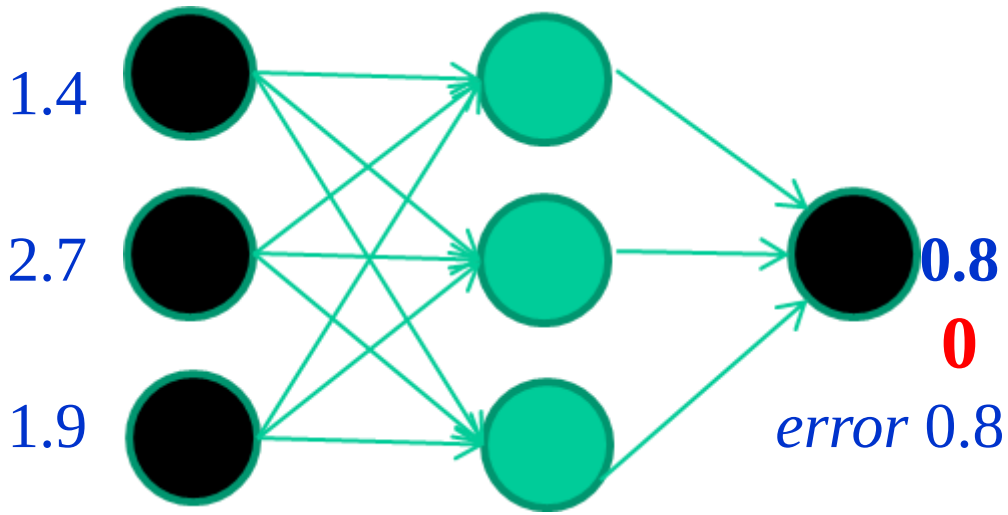
Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**



Example

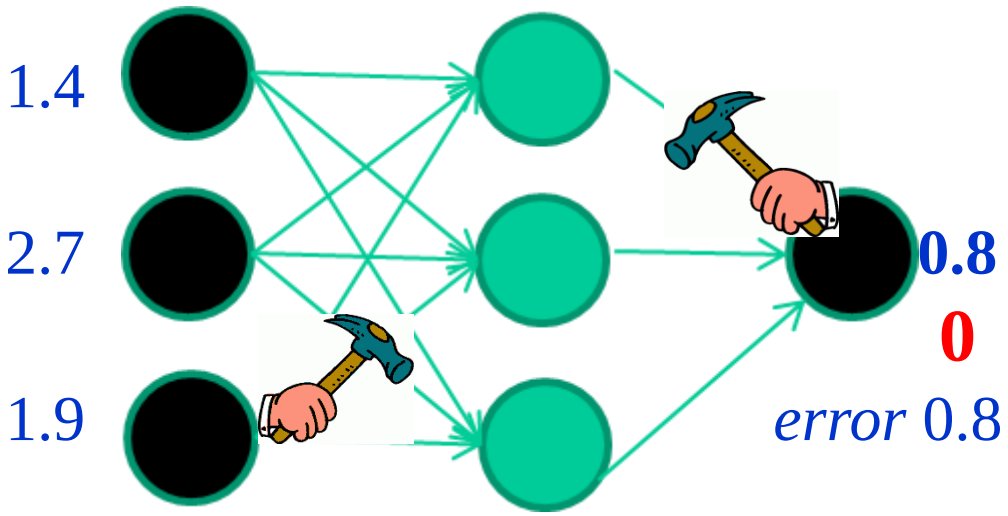
Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**
- **Adjust weights based on error**



Example

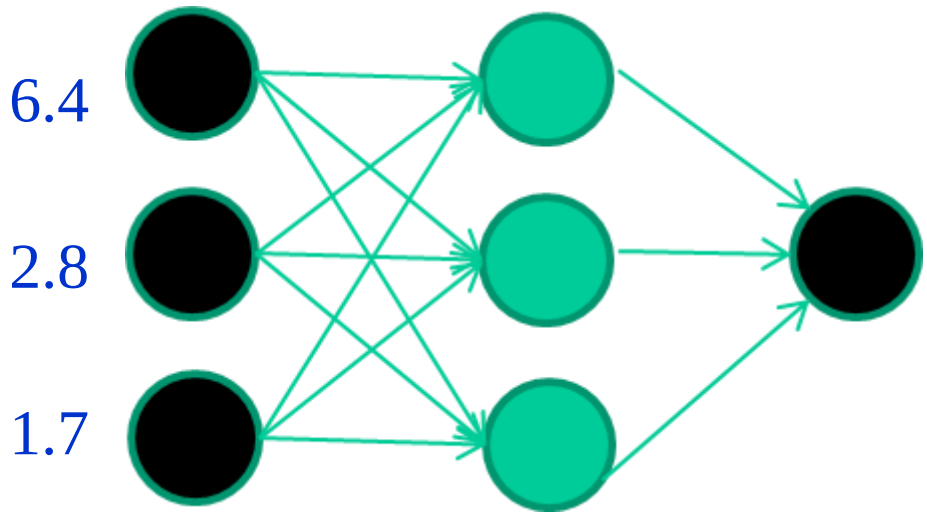
Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**
- **Adjust weights based on error**



Example

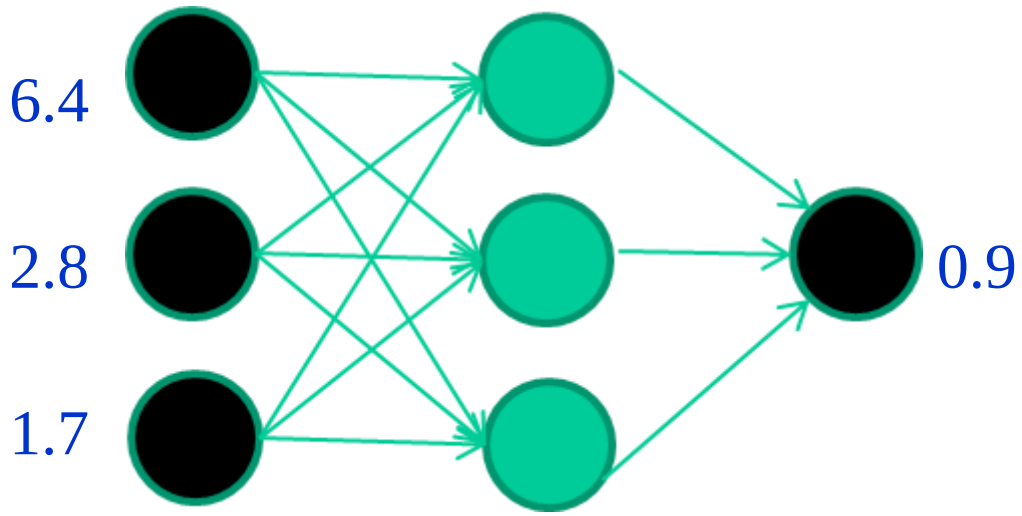
Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**
- **Adjust weights based on error**



Example

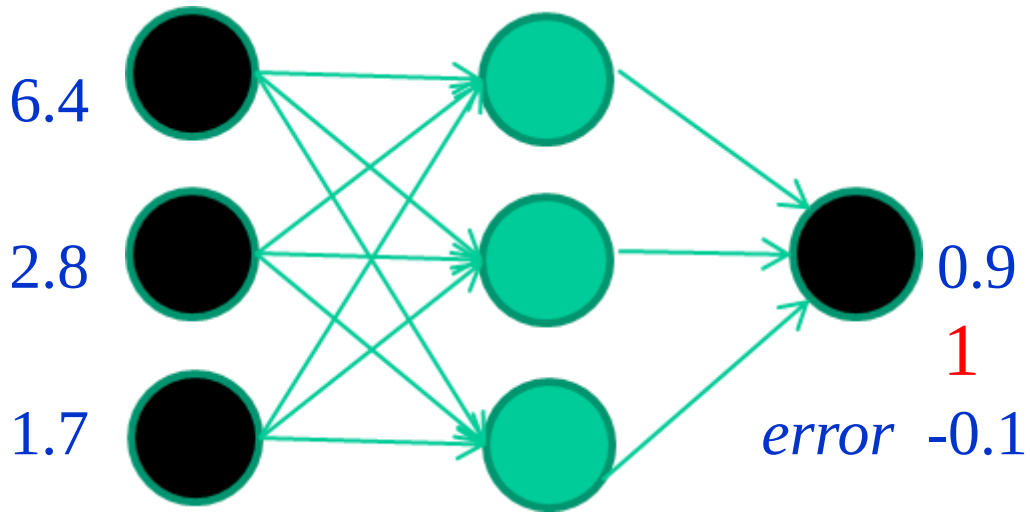
Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**
- **Adjust weights based on error**



Example

Training data

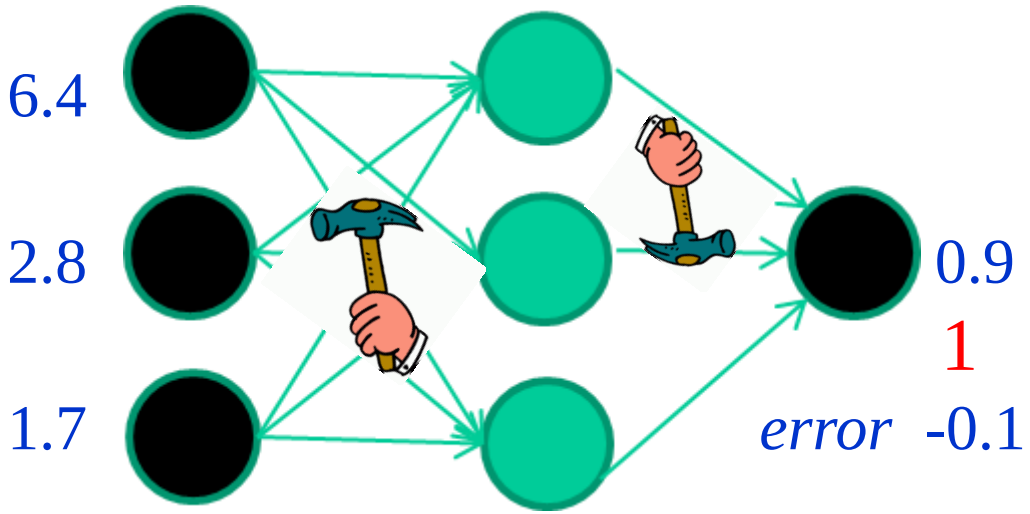
Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Training the neural network

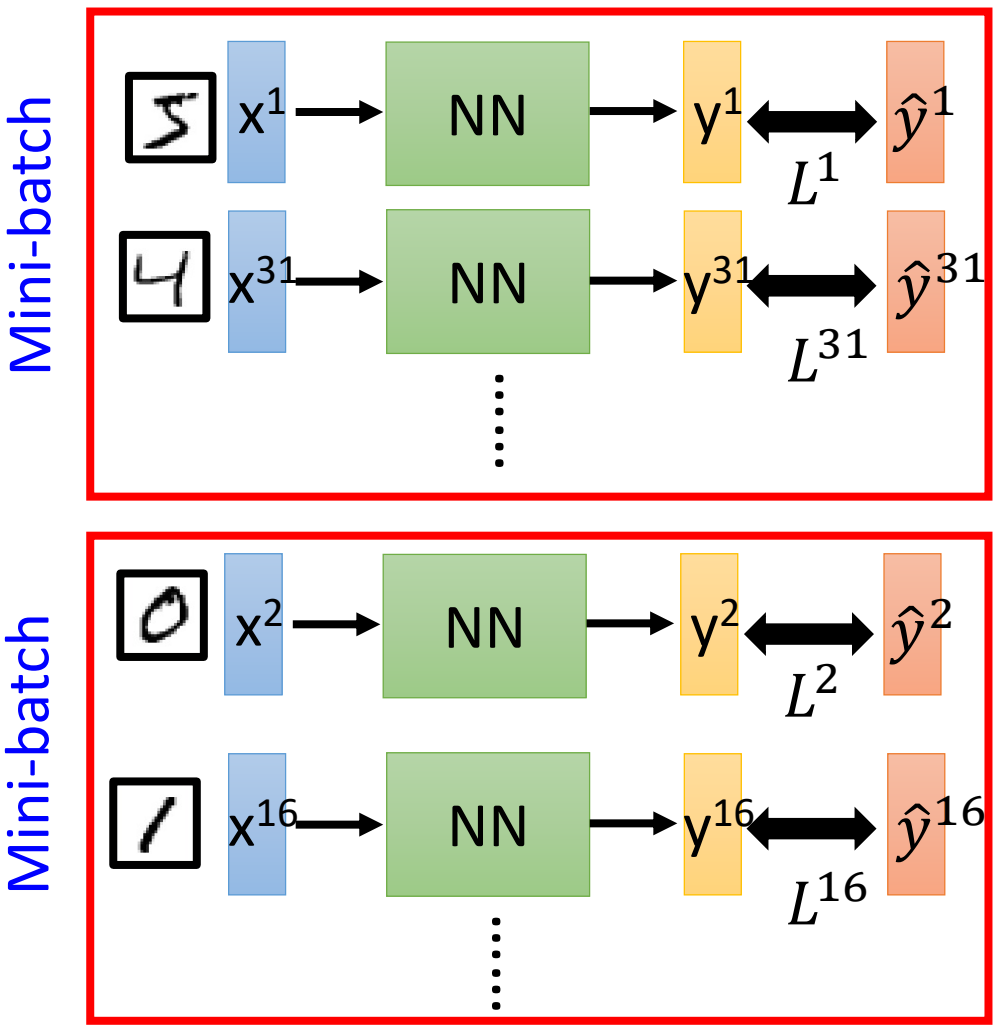
Initialise with random weights

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**
- **Adjust weights based on error**

Repeat…



Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

$$L = L^1 + L^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla L(\theta^0)$$

- Pick the 2nd batch

$$L = L^2 + L^{16} + \cdots$$

$$\theta^2 \leftarrow \theta^1 - \eta \nabla L(\theta^1)$$

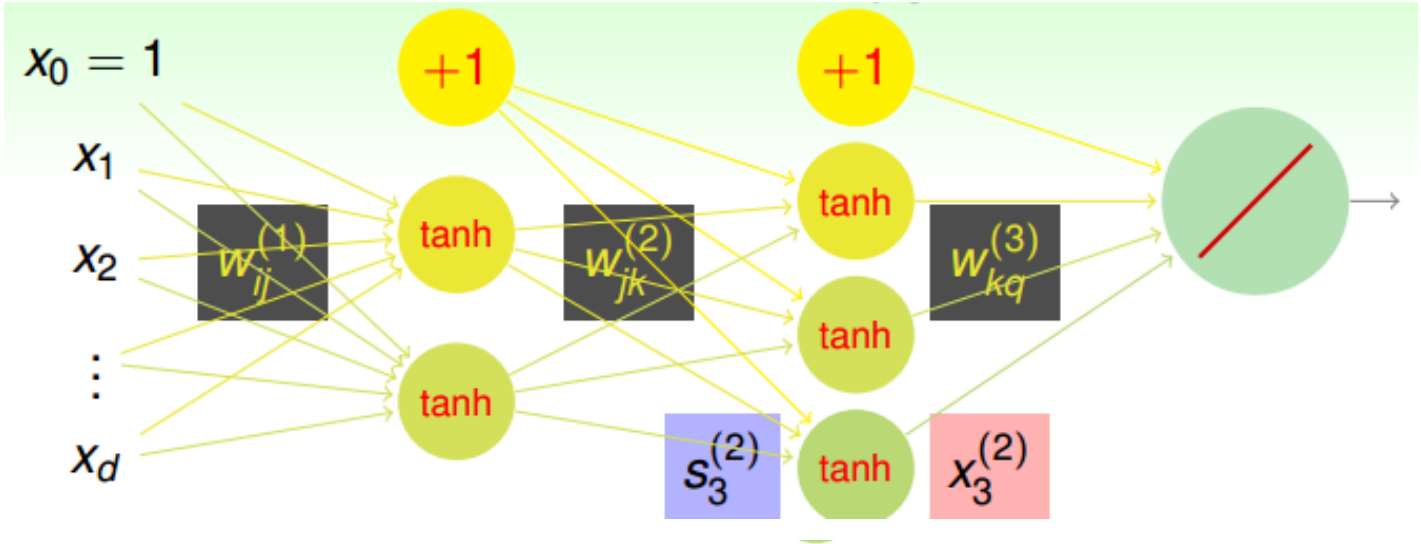
$\vdots$

- Until all mini-batches have been picked

one epoch

Repeat the above process

Neural Network (Formal) Hypothesis



$d^{(0)}$-$d^{(1)}$-$d^{(2)}$-$\cdots$-$d^{(L)}$ Neural Network (NNet)

$$w_{ij}^{(\ell)}: \begin{cases} 1 \le \ell \le L & \text{layers} \\ 0 \le i \le d^{(\ell-1)} & \text{inputs} \\ 1 \le j \le d^{(\ell)} & \text{outputs} \end{cases}, \quad \text{score } s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_i^{(\ell-1)},$$

$$\text{transformed } x_j^{(\ell)} = \begin{cases} \tanh\left(s_j^{(\ell)}\right) & \text{if } \ell < L \\ s_j^{(\ell)} & \text{if } \ell = L \end{cases}$$

apply $\mathbf{x}$ as input layer $\mathbf{x}^{(0)}$, go through hidden layers to get $\mathbf{x}^{(\ell)}$, predict at output layer $x_1^{(L)}$

Exercise

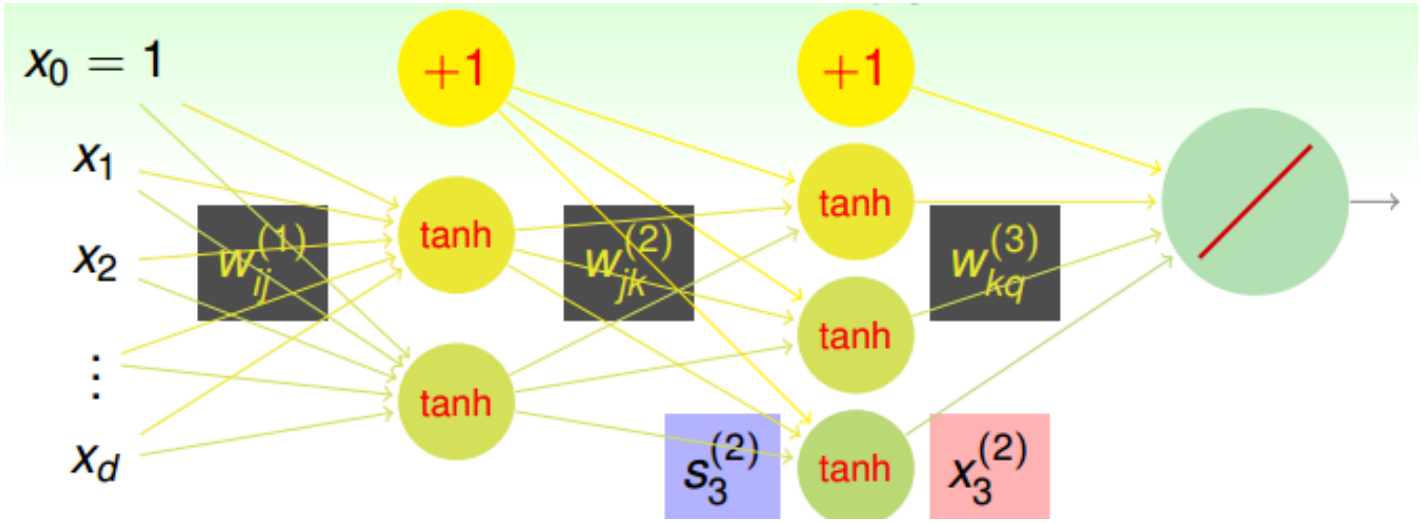
How many weights $\{w_{ij}^{(\ell)}\}$ are there in a 3-5-1 NNet?

1. 9
2. 15
3. 20
4. 26

Reference Answer: (4)

There are $(3+1) \times 5$ weights in $w_{ij}^{(1)}$, and $(5+1) \times 1$ weights in $w_{jk}^{(2)}$.

How to Learn the Weights?



- goal: learning all $\{w_{ij}^{(\ell)}\}$ to **minimize** $J\left(\{w_{ij}^{(\ell)}\}\right)$
- let $e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2$:
  can apply **(stochastic) GD** after computing $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$!

next: efficient computation of $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$

Backpropagation

- Backpropagation
  - Information flows backwards
  - Computes the gradient efficiently.
  - The chain rule of calculus.

- Many toolkits can compute the gradients automatically

theano

TensorFlow

Computing $\frac{\partial e_n}{\partial w_{i1}^{(L)}}$ (Output Layer)

$$e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2 = \left(y_n - s_1^{(L)}\right)^2 = \left(y_n - \sum_{i=0}^{d^{(L-1)}} w_{i1}^{(L)} x_i^{(L-1)}\right)^2$$

specially (output layer) $(0 \le i \le d^{(L-1)})$

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{i1}^{(L)}} \\ = & \frac{\partial e_n}{\partial s_1^{(L)}} \cdot \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} \\ = & -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right) \end{aligned}$$

generally $(1 \le \ell < L)$ $(0 \le i \le d^{(\ell-1)}; 1 \le j \le d^{(\ell)})$

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{ij}^{(\ell)}} \\ = & \frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} \\ = & \delta_j^{(\ell)} \cdot \left(x_i^{(\ell-1)}\right) \end{aligned}$$

$\delta_1^{(L)} = -2\left(y_n - s_1^{(L)}\right)$, how about **others**?

Computing $\delta_j^{(l)} = \frac{\partial e_n}{\partial s_j^{(l)}}$

$$s_j^{(\ell)} \xrightarrow{\tanh} x_j^{(\ell)} \xrightarrow{w_{jk}^{(\ell+1)}} \begin{bmatrix} s_1^{(\ell+1)} \\ \vdots \\ s_k^{(\ell+1)} \\ \vdots \end{bmatrix} \Longrightarrow \cdots \Longrightarrow e_n$$

$$\delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} = \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}}$$

$$= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\tanh'\left(s_j^{(\ell)}\right)\right)$$

$\delta_j^{(\ell)}$ can be computed **backwards** from $\delta_k^{(\ell+1)}$

Backpropagation

Backprop on NNet

initialize all weights $w_{ij}^{(\ell)}$

for $t = 0, 1, \ldots, T$

1. stochastic: randomly pick $n \in \{1, 2, \cdots, N\}$
2. forward: compute all $x_i^{(\ell)}$ with $\mathbf{x}^{(0)} = \mathbf{x}_n$
3. backward: compute all $\delta_j^{(\ell)}$ subject to $\mathbf{x}^{(0)} = \mathbf{x}_n$
4. gradient descent: $w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta x_i^{(\ell-1)} \delta_j^{(\ell)}$

return $g_{\text{NNET}}(\mathbf{x}) = \left(\cdots \tanh \left(\sum_j w_{jk}^{(2)} \cdot \tanh \left(\sum_i w_{ij}^{(1)} x_i \right) \right) \right)$

sometimes (1) to (3) is (parallelly) done many times and average($x_i^{(\ell-1)} \delta_j^{(\ell)}$) taken for update in (4), called **mini-batch**

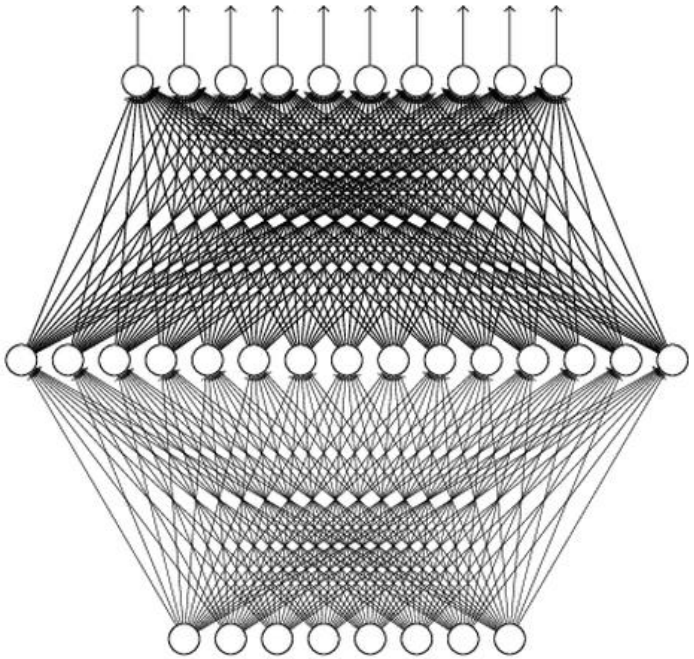
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^{\mathrm{M}}$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



How Many Layers Are Deep?

- **No much clear definition**
  - "several", "many"…

- **Neural nets with 1 hidden layer are not deep.**
  - No feature hierarchy

Ref: LeCun from NYU, Ranzato from Google, Deep Learning Tutorial@ ICML 2013

Short Summary

- **Cost function**
  - Euclidean distance.
  - Cross-entropy (negative log-likelyhood).
    - (Maximum likelihood principle.) $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$
- **Back-propagation**
  - The chain rule of calculus.
  - Many toolkits.
  - Mini batch.

Radial Basis Function Network

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right)$:

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

—linear aggregation of selected radial hypotheses

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, **remember? :-)**

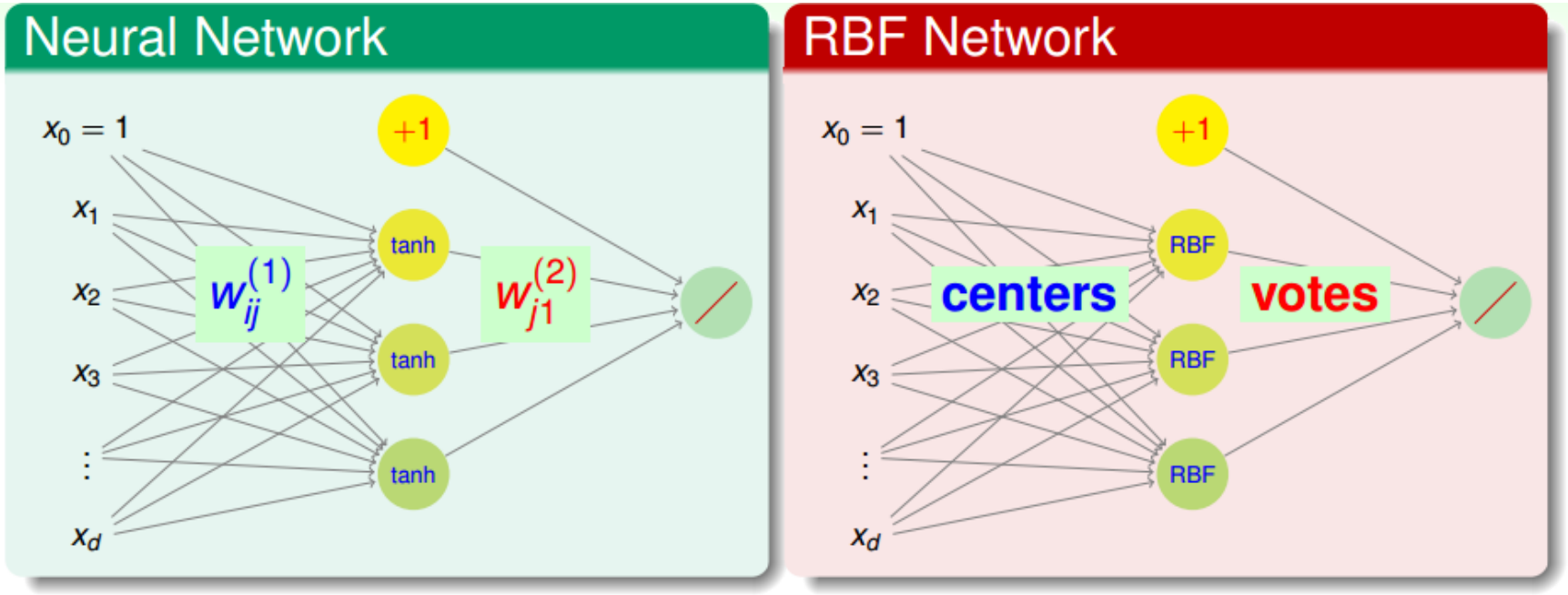
- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right)$:

  $$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\textstyle\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

  —linear aggregation of selected radial hypotheses

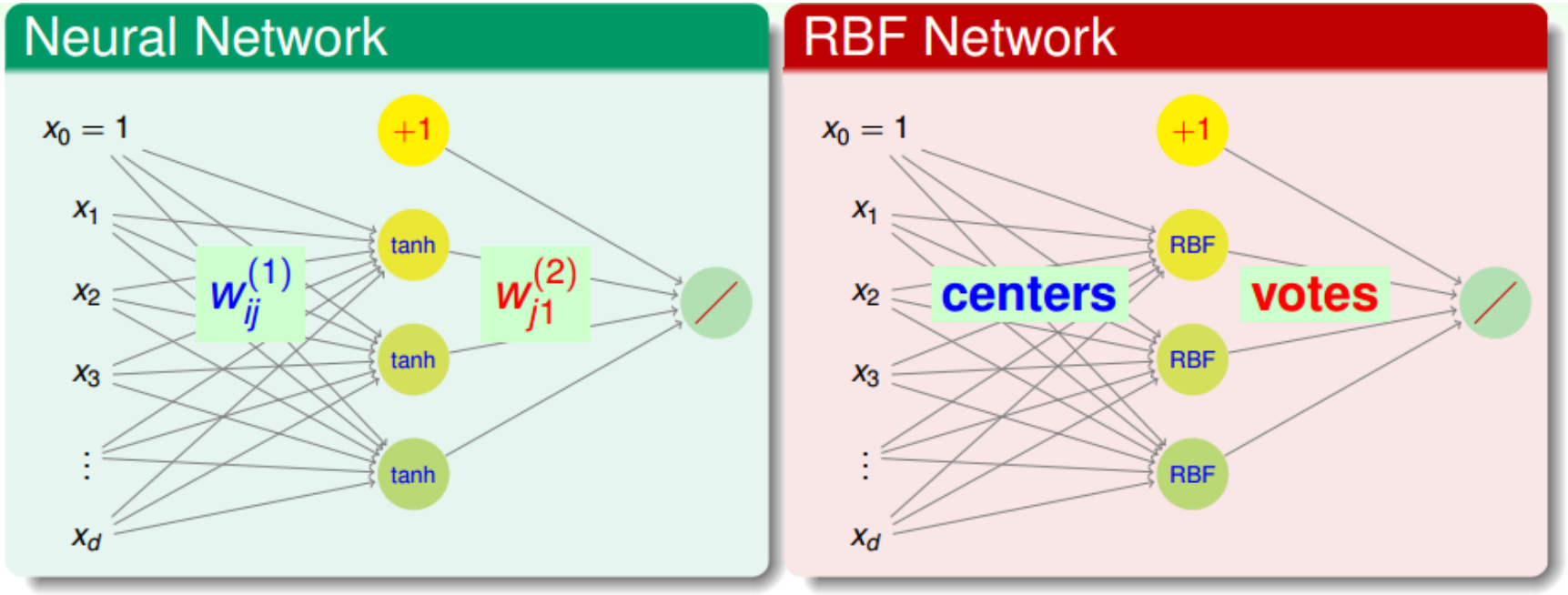
Radial Basis Function (RBF) **Network**:
linear aggregation of radial hypotheses

From Neural Network to RBF Network



- hidden layer different:
  (inner-product + tanh) versus (distance + Gaussian)
- output layer same: **just linear aggregation**

From Neural Network to RBF Network



- hidden layer different:
  (inner-product + tanh) versus (distance + Gaussian)
- output layer same: **just linear aggregation**

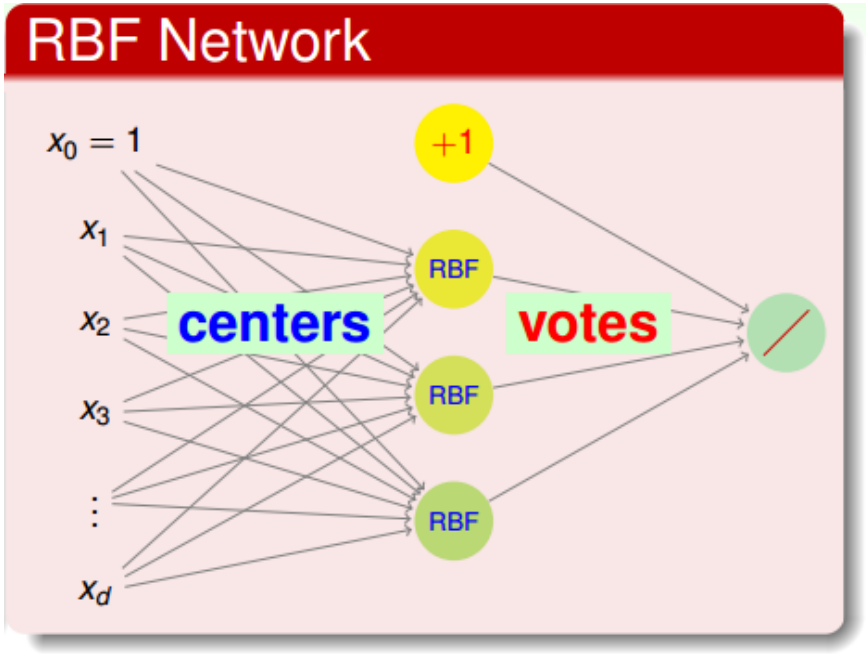
RBF Network: historically **a type of NNet**

RBF Network Hypothesis

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

key variables:

centers $\boldsymbol{\mu}_m$; (signed) votes β_m



g_{SVM} for Gaussian-SVM

- RBF: Gaussian; Output: sign (binary classification)
- $M = \#\text{SV}$; $\boldsymbol{\mu}_m$: SVM SVs $\mathbf{x}_m$; β_m: $\alpha_m y_m$ from SVM Dual

Exercise

Which of the following is not a radial basis function?

1. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \exp(-\gamma \|\mathbf{x} - \boldsymbol{\mu}\|^2)$
2. $\phi(\mathbf{x}, \boldsymbol{\mu}) = -\sqrt{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\mu}}$
3. $\phi(\mathbf{x}, \boldsymbol{\mu}) = [\![\mathbf{x} = \boldsymbol{\mu}]\!]$
4. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{x}^T\mathbf{x} + \boldsymbol{\mu}^T\boldsymbol{\mu}$

Exercise

Which of the following is not a radial basis function?

1. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \exp(-\gamma \|\mathbf{x} - \boldsymbol{\mu}\|^2)$
2. $\phi(\mathbf{x}, \boldsymbol{\mu}) = -\sqrt{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\mu}}$
3. $\phi(\mathbf{x}, \boldsymbol{\mu}) = [\![\mathbf{x} = \boldsymbol{\mu}]\!]$
4. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{x}^T\mathbf{x} + \boldsymbol{\mu}^T\boldsymbol{\mu}$

Reference Answer: (4)

Note that (3) is an extreme case of (1) (Gaussian) with $\gamma \to \infty$, and (2) contains an $\|\mathbf{x} - \boldsymbol{\mu}\|^2$ somewhere :-).

Full RBF Network

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m)\right)$$

- full RBF Network: $M = N$ and each $\boldsymbol{\mu}_m = \mathbf{x}_m$
- physical meaning: each $\mathbf{x}_m$ **influences** similar $\mathbf{x}$ by β_m
- e.g. uniform influence with $\beta_m = 1 \cdot y_m$ for binary classification

$$g_{\text{uniform}}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{N} y_m \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_m\|^2\right)\right)$$

—**aggregate** each example's opinion subject to similarity

Nearest Neighbor

$$g_{\text{uniform}}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{N} y_m \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_m\|^2\right)\right)$$

- $\exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_m\|^2\right)$: maximum when $\mathbf{x}$ **closest to** $\mathbf{x}_m$
  —maximum one often dominates the $\sum_{m=1}^{N}$ term
- take y_m of **maximum** $\exp(\ldots)$ instead of voting of all y_m
  —**selection** instead of **aggregation**
- physical meaning:

  $$g_{\text{nbor}}(\mathbf{x}) = y_m \text{ such that } \mathbf{x} \textbf{ closest to } \mathbf{x}_m$$

  —called **nearest neighbor** model
- can **uniformly aggregate** k neighbors also: k **nearest neighbor**

Full RBF Network for Regression

full RBF Network for squared error regression:

$$h(\mathbf{x}) = \cancel{\text{Output}}\left(\sum_{m=1}^{N} \beta_m \text{RBF}(\mathbf{x}, \mathbf{x}_m)\right)$$

- just linear regression on RBF-transformed data

$$\mathbf{z}_n = [\text{RBF}(\mathbf{x}_n, \mathbf{x}_1), \text{RBF}(\mathbf{x}_n, \mathbf{x}_2), \ldots, \text{RBF}(\mathbf{x}_n, \mathbf{x}_N)]$$

Fewer Centers as Regularization

recall:

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_m y_m \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_m\|^2\right) + b\right)$$

—only '$\ll N$' SVs needed in 'network'

- next: $M \ll N$ instead of $M = N$
- effect: **regularization**
  by constraining **number of centers and voting weights**
- physical meaning of centers μ_m: **prototypes**

remaining question:
how to extract **prototypes**?

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,
$\Longrightarrow$ **no need** both RBF$(\mathbf{x}, \mathbf{x}_1)$ & RBF$(\mathbf{x}, \mathbf{x}_2)$ in RBFNet,
$\Longrightarrow$ **cluster** $\mathbf{x}_1$ and $\mathbf{x}_2$ by **one prototype** $\mu \approx \mathbf{x}_1 \approx \mathbf{x}_2$

- **clustering** with **prototype**:
  - **partition** $\{\mathbf{x}_n\}$ to disjoint sets $S_1, S_2, \cdots, S_M$
  - **choose** μ_m for each S_m

  —hope: $\mathbf{x}_1, \mathbf{x}_2$ both $\in S_m \Leftrightarrow \mu_m \approx \mathbf{x}_1 \approx x_2$
- cluster error with squared error measure:

$$E_{\text{in}}(S_1, \cdots, S_M; \mu_1, \cdots, \mu_M) = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \mu_m\|^2$$

goal: with $S_1, \cdots, S_M$ being a partition of $\{\mathbf{x}_n\}$,

$$\min_{\{S_1, \cdots, S_M; \mu_1, \cdots, \mu_M\}} E_{\text{in}}(S_1, \cdots, S_M; \mu_1, \cdots, \mu_M)$$

Partition Optimization

with $S_1, \cdots, S_M$ being a partition of $\{\mathbf{x}_n\}$,

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N}\sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

- **hard to optimize**: joint combinatorial-numerical optimization
- **two sets** of variables: will optimize alternatingly

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

- $[\![\mathbf{x}_n \in S_m]\!]$: choose one and only one subset
- $\|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$: distance to each prototype

optimal chosen subset $S_m =$ the one with **minimum** $\|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$

for given $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$, each $\mathbf{x}_n$
'optimally partitioned' using its **closest** $\boldsymbol{\mu}_m$

Prototype Optimization

with $S_1, \cdots, S_M$ being a partition of $\{\mathbf{x}_n\}$,

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N}\sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

- **hard to optimize**: joint combinatorial-numerical optimization
- **two sets** of variables: will optimize alternatingly

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

$$\nabla_{\boldsymbol{\mu}_m} E_{\text{in}} = -2\sum_{n=1}^{N} [\![\mathbf{x}_n \in S_m]\!](\mathbf{x}_n - \boldsymbol{\mu}_m) = -2\left(\left(\sum_{\mathbf{x}_n \in S_m} \mathbf{x}_n\right) - |S_m|\boldsymbol{\mu}_m\right)$$

optimal prototype $\boldsymbol{\mu}_m =$ **average** of $\mathbf{x}_n$ within S_m

for given $S_1, \cdots, S_M$, each $\boldsymbol{\mu}_m$
'optimally computed' as **consensus** within S_m

k-Means Algorithm

use k **prototypes** instead of M historically
(different from k nearest neighbor, though)

k-Means Algorithm

1. initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$: say, as k randomly chosen $\mathbf{x}_n$
2. **alternating optimization** of E_{in}: repeatedly
   1. optimize $S_1, S_2, \ldots, S_k$:
      each $\mathbf{x}_n$ 'optimally partitioned' using its closest $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ 'optimally computed' as consensus within S_m

   until **converge**

converge: no change of $S_1, S_2, \ldots, S_k$ anymore
—guaranteed as E_{in} **decreases** during alternating minimization

k-Means: the most popular **clustering**
algorithm through **alternating minimization**

RBF Network Using k-Means

RBF Network Using k-Means

1. run k-Means with $k = M$ to get $\{\boldsymbol{\mu}_m\}$
2. construct transform $\boldsymbol{\Phi}(\mathbf{x})$ from RBF (say, Gaussian) at $\boldsymbol{\mu}_m$

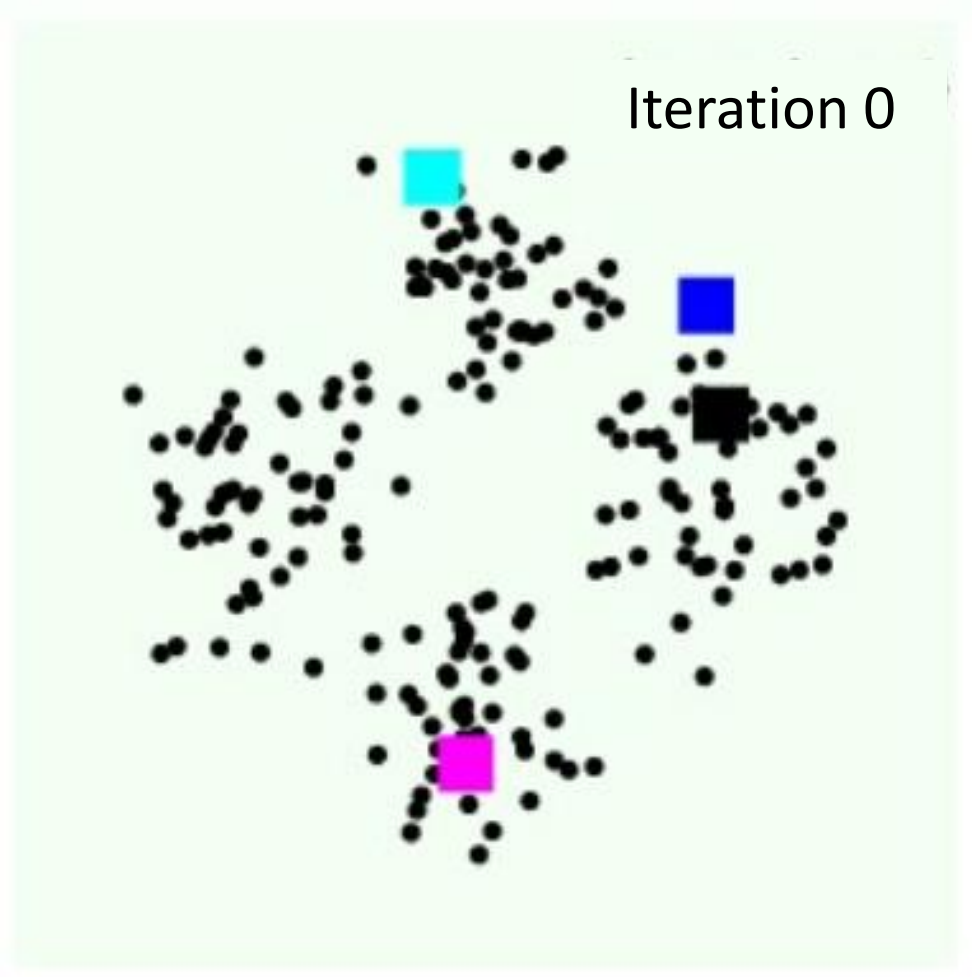
$$\boldsymbol{\Phi}(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

3. run linear model on $\{(\boldsymbol{\Phi}(\mathbf{x}_n), y_n)\}$ to get $\boldsymbol{\beta}$
4. return $g_{\text{RBFNET}}(\mathbf{x}) = \text{LinearHypothesis}\,(\boldsymbol{\beta}, \boldsymbol{\Phi}(\mathbf{x}))$

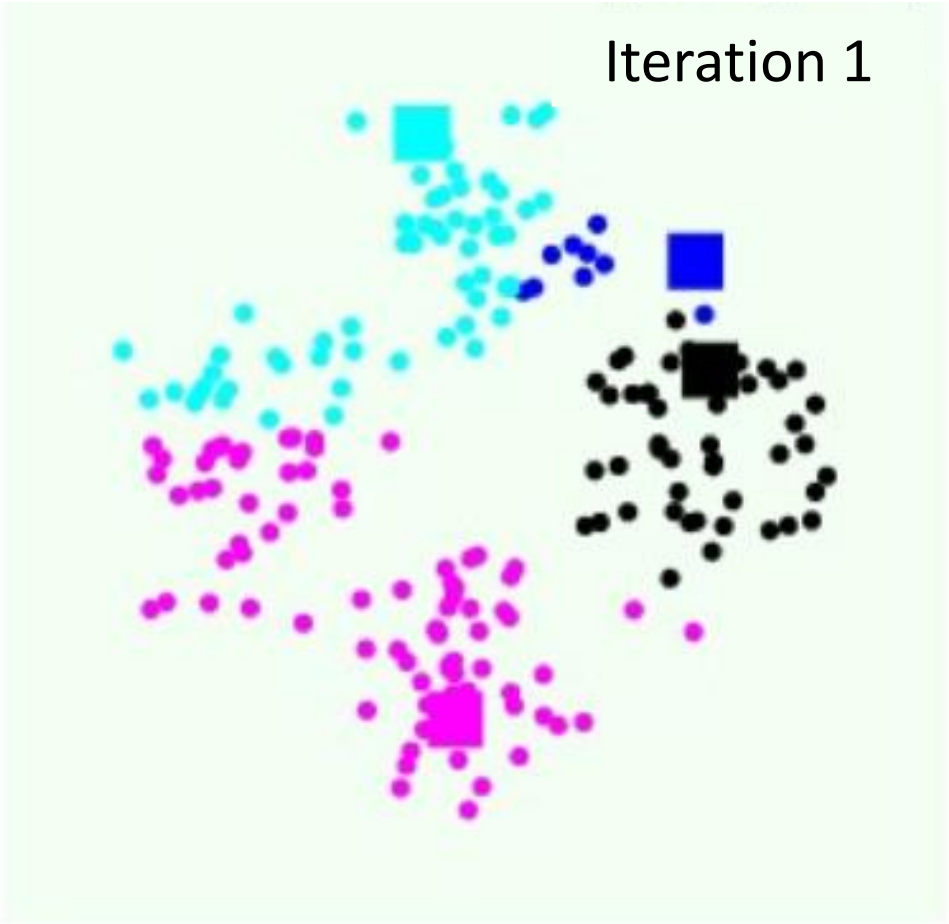
- using unsupervised learning (k-Means) to assist **feature transform**—like **autoencoder**
- parameters: M (prototypes), RBF (such as γ of Gaussian)

RBF Network: a simple (**old-fashioned**) model

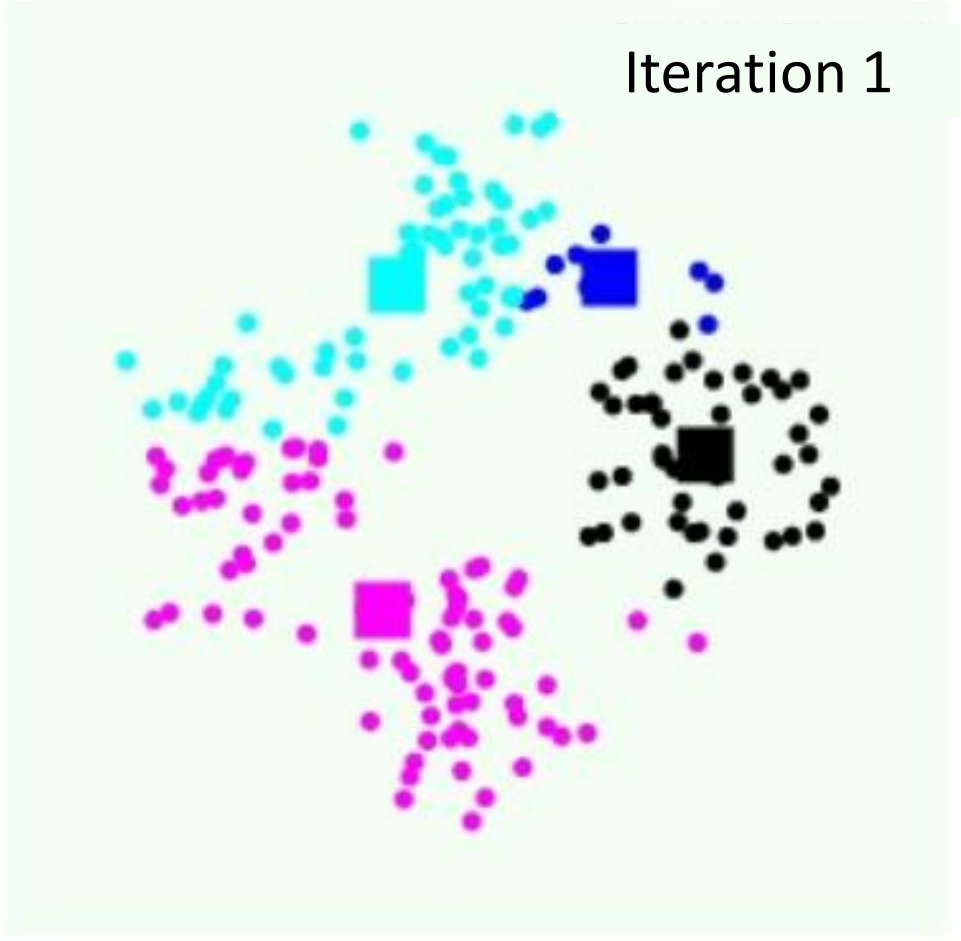
k-Means Algorithm



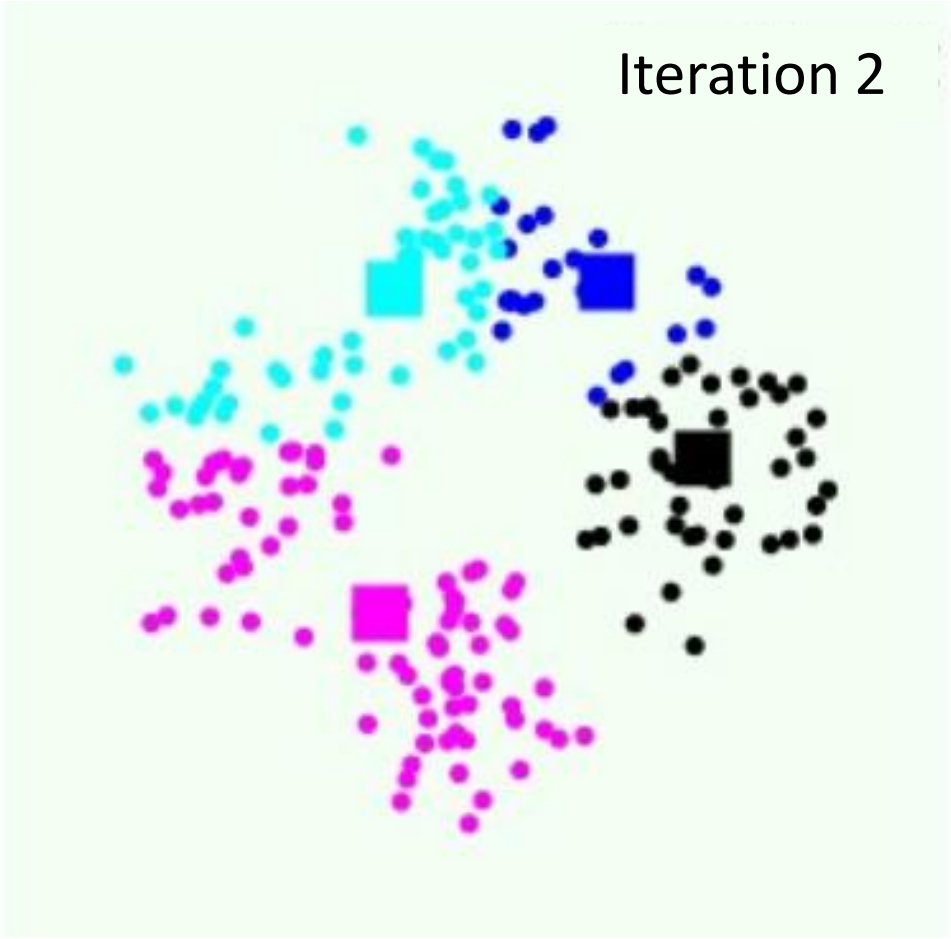
k-Means Algorithm



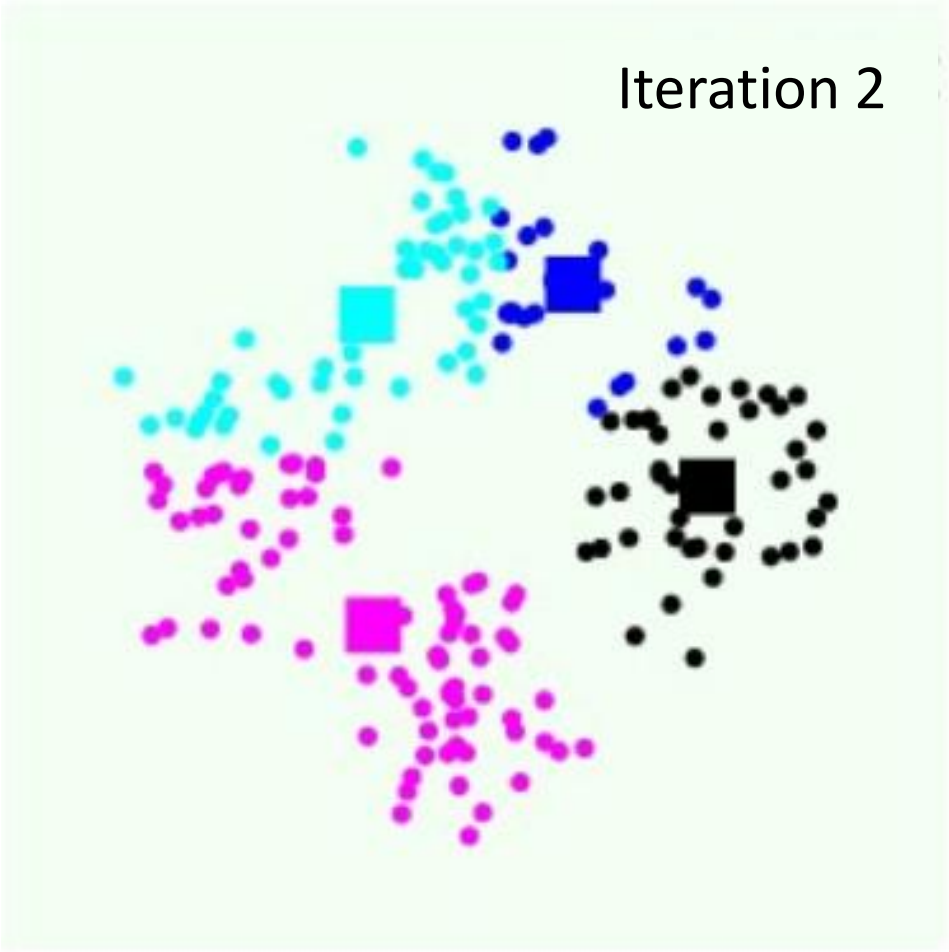
k-Means Algorithm



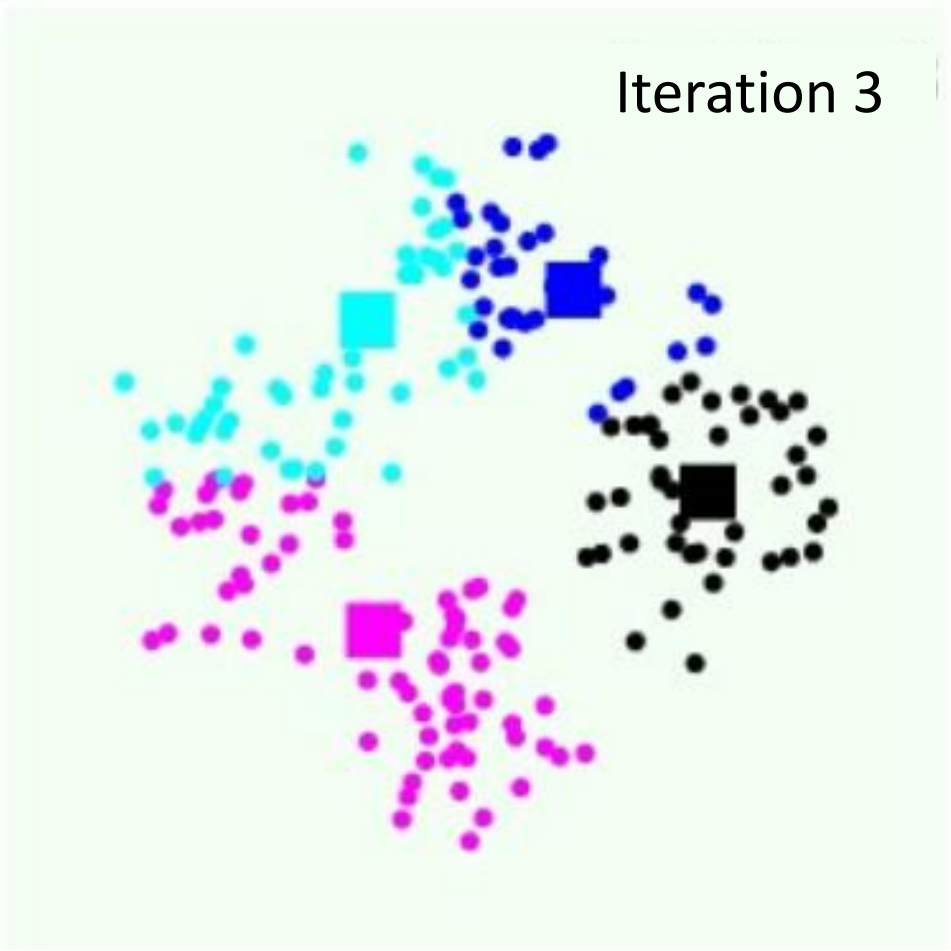
k-Means Algorithm



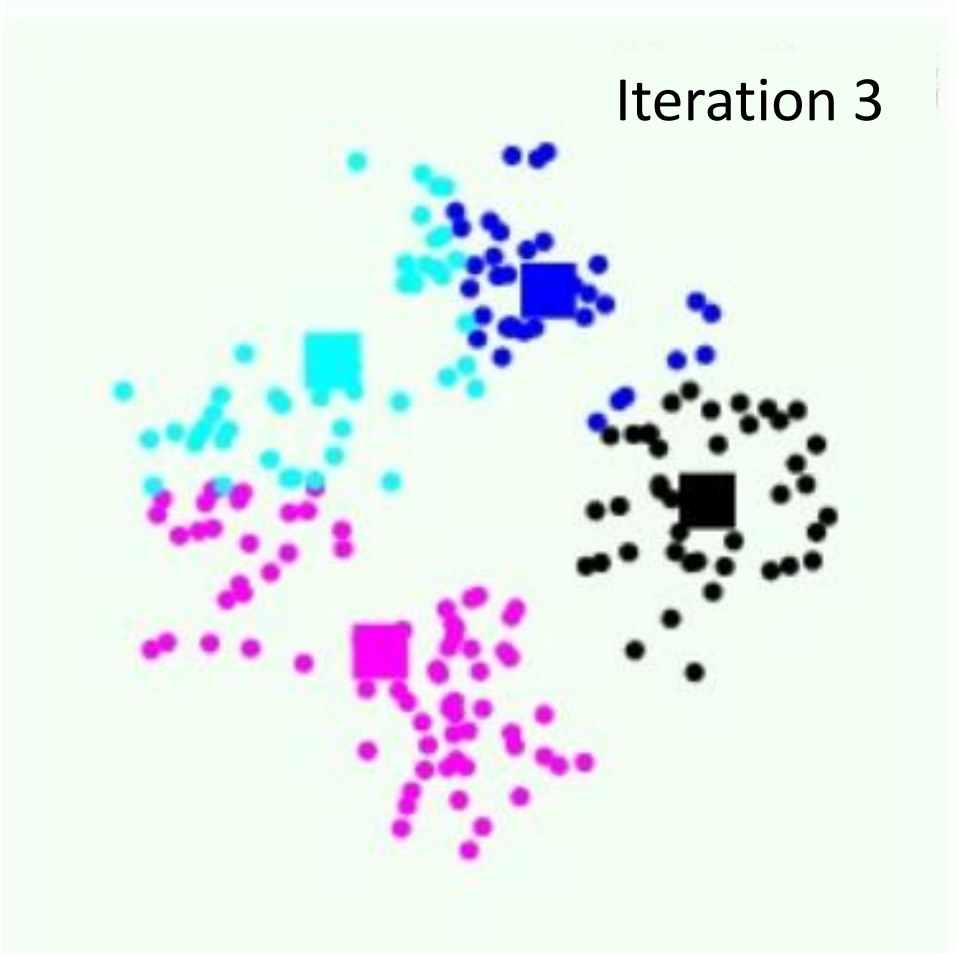
k-Means Algorithm



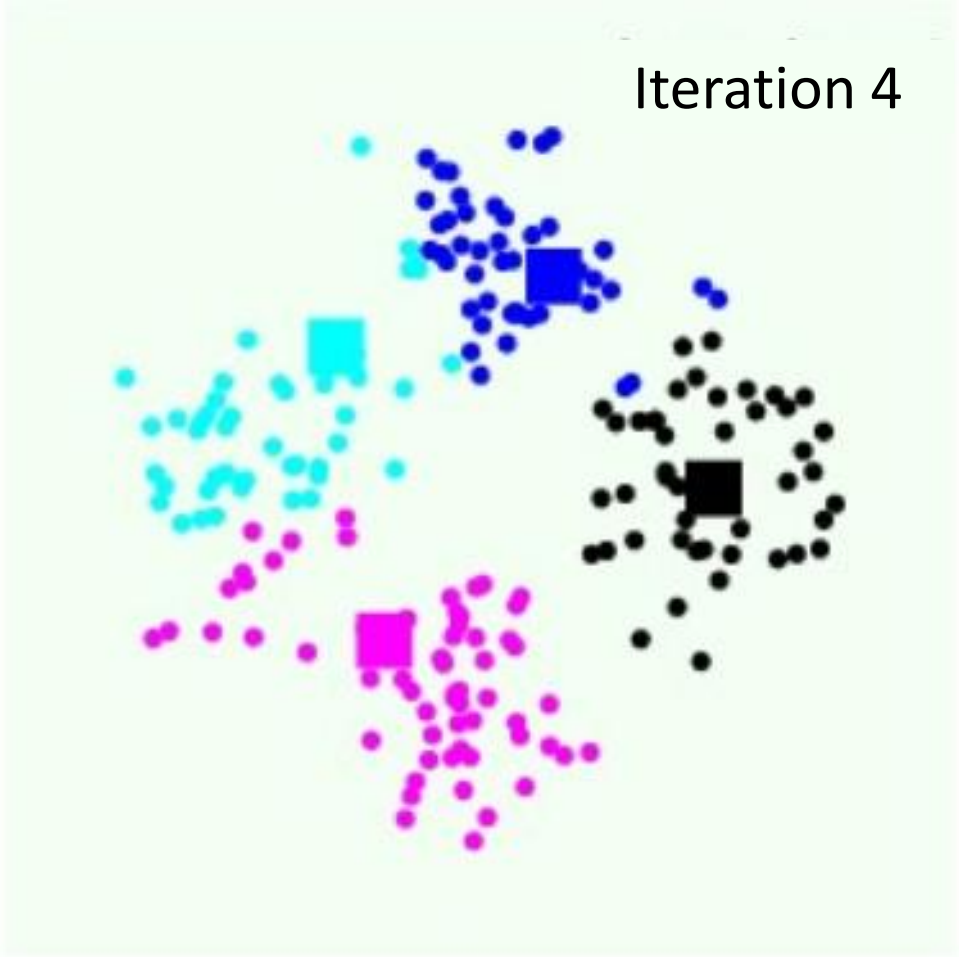
k-Means Algorithm



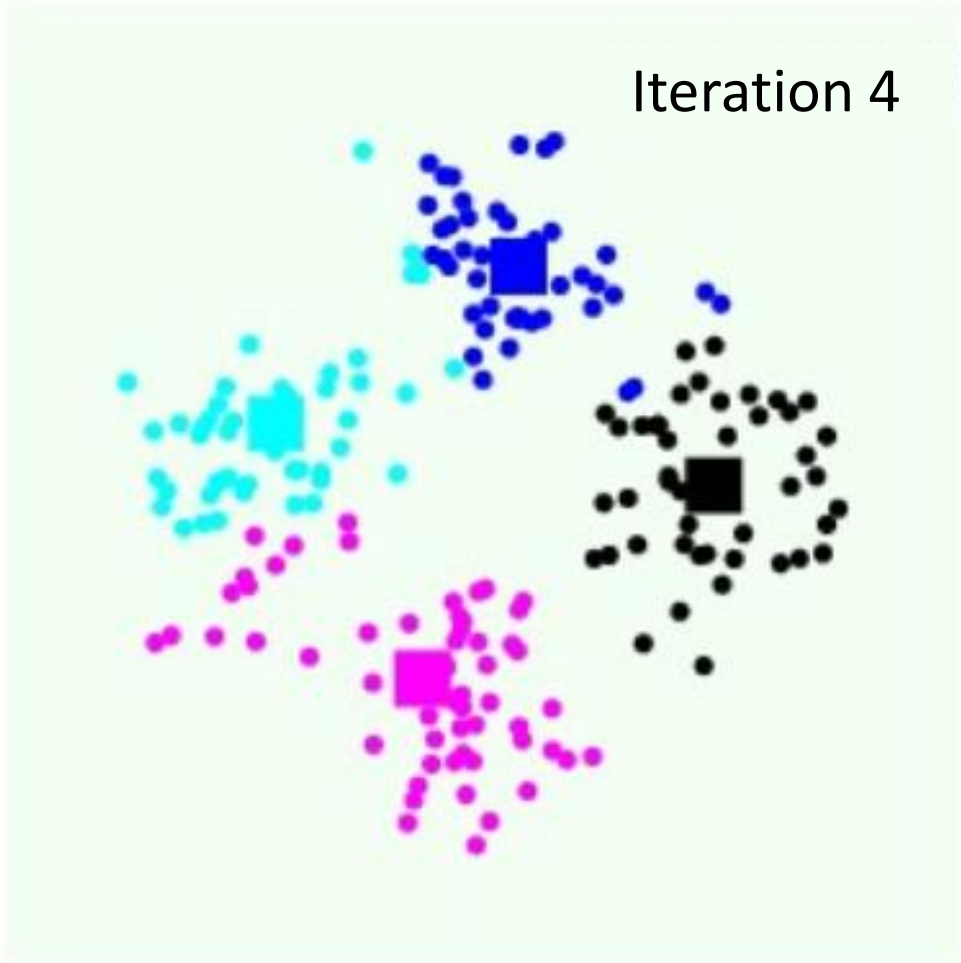
k-Means Algorithm



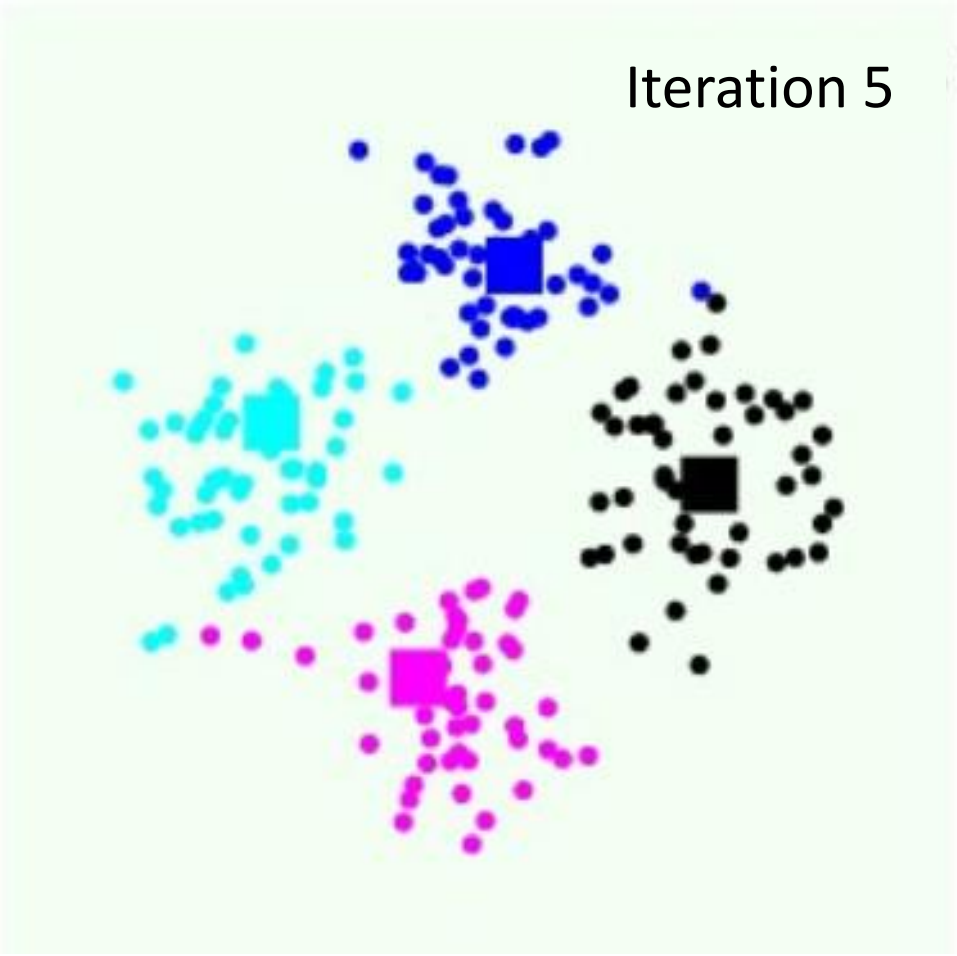
k-Means Algorithm



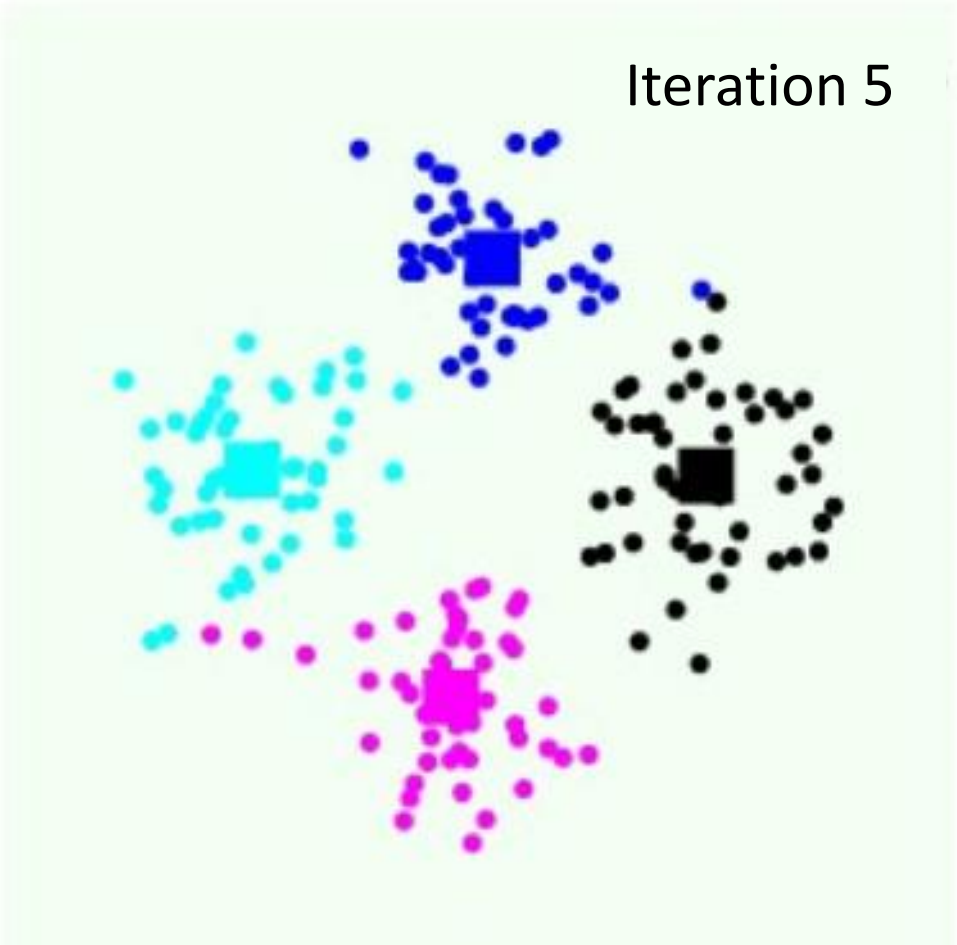
k-Means Algorithm



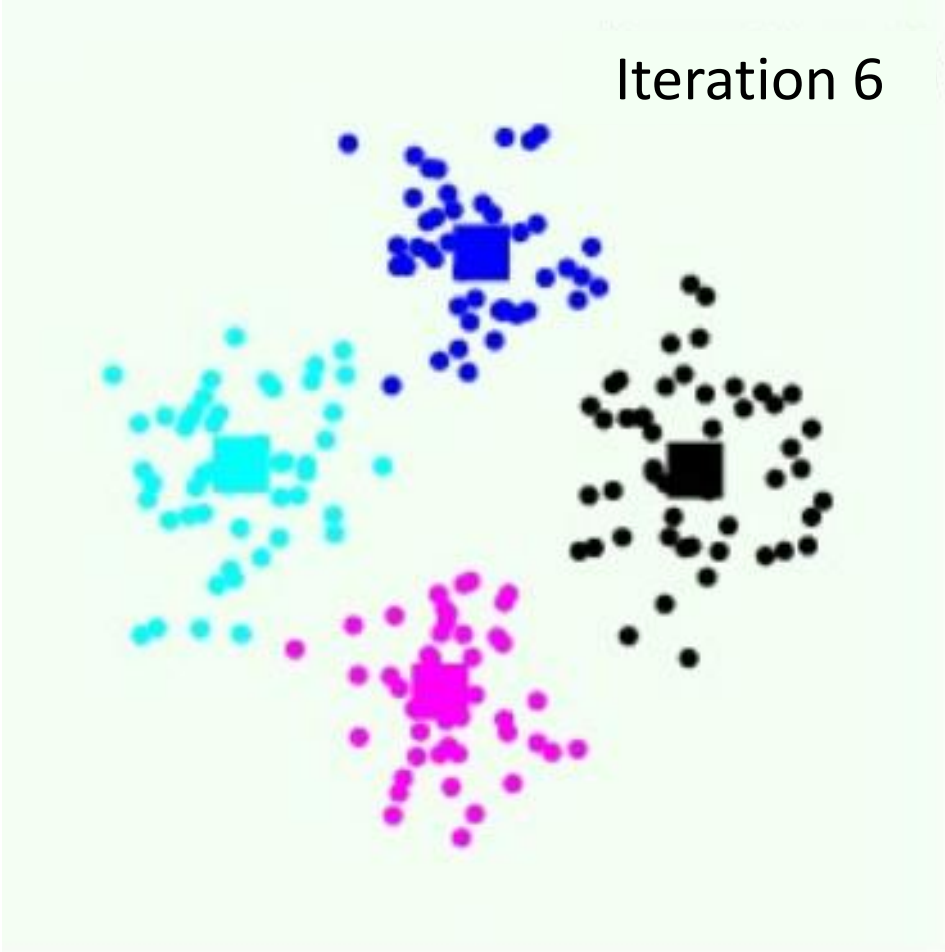
k-Means Algorithm



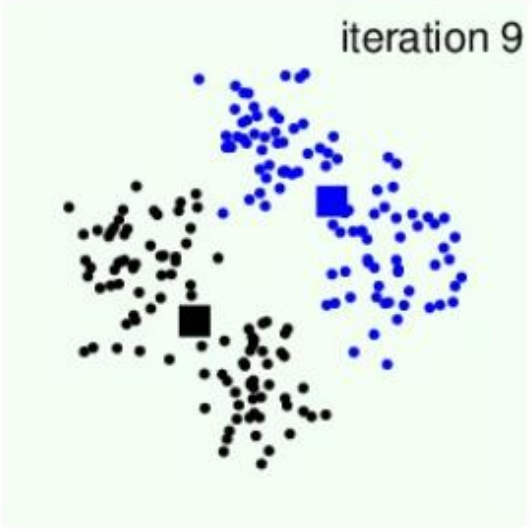
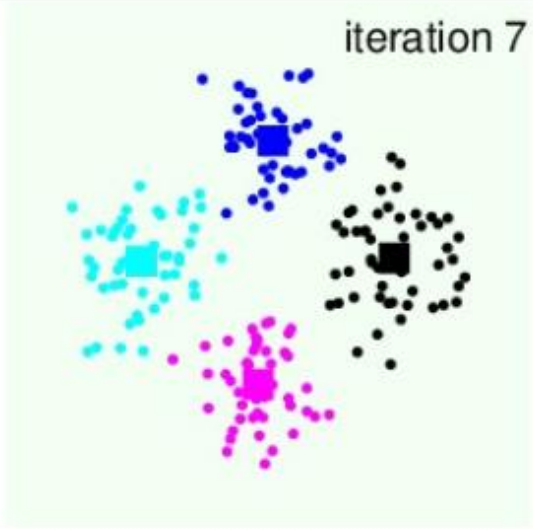
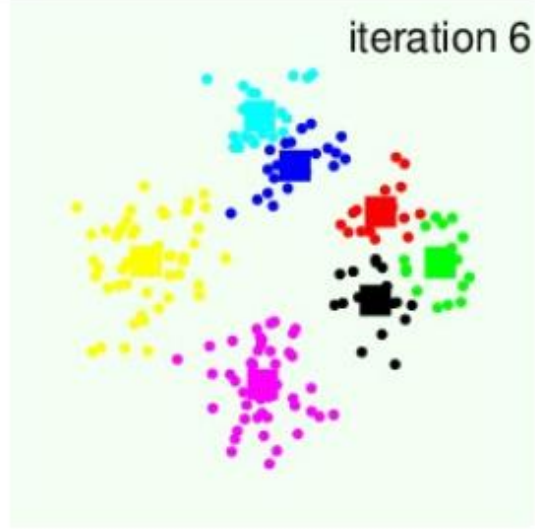
k-Means Algorithm



k-Means Algorithm

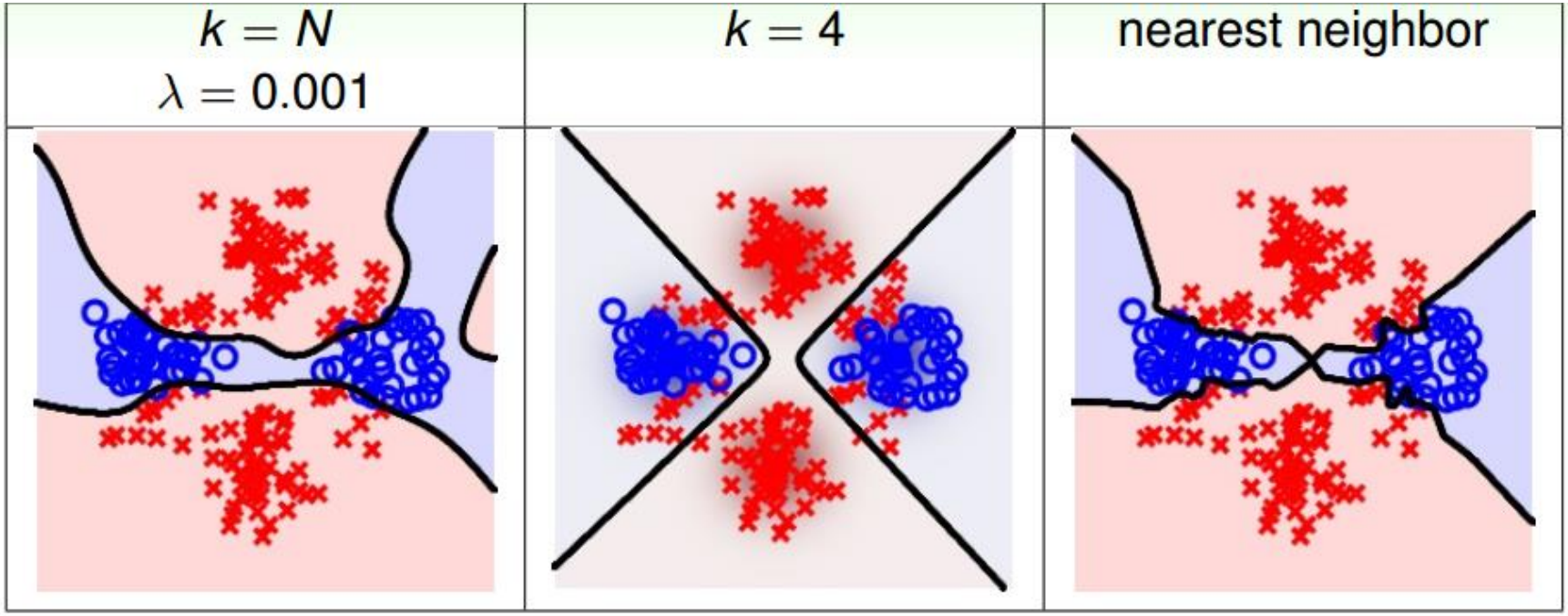


Difficulty of k-Means

$k = 2$	$k = 4$	$k = 7$
		

'sensitive' to k and initialization

Difficulty of k-Means



full RBF Network: generally less useful

Convolutional Neural Network

Convolutional Networks

Use convolution in place of general matrix multiplication in at least one of their layers. $\boldsymbol{w}$

Convolution is a specialized kind of linear operation.

- Average estimate.

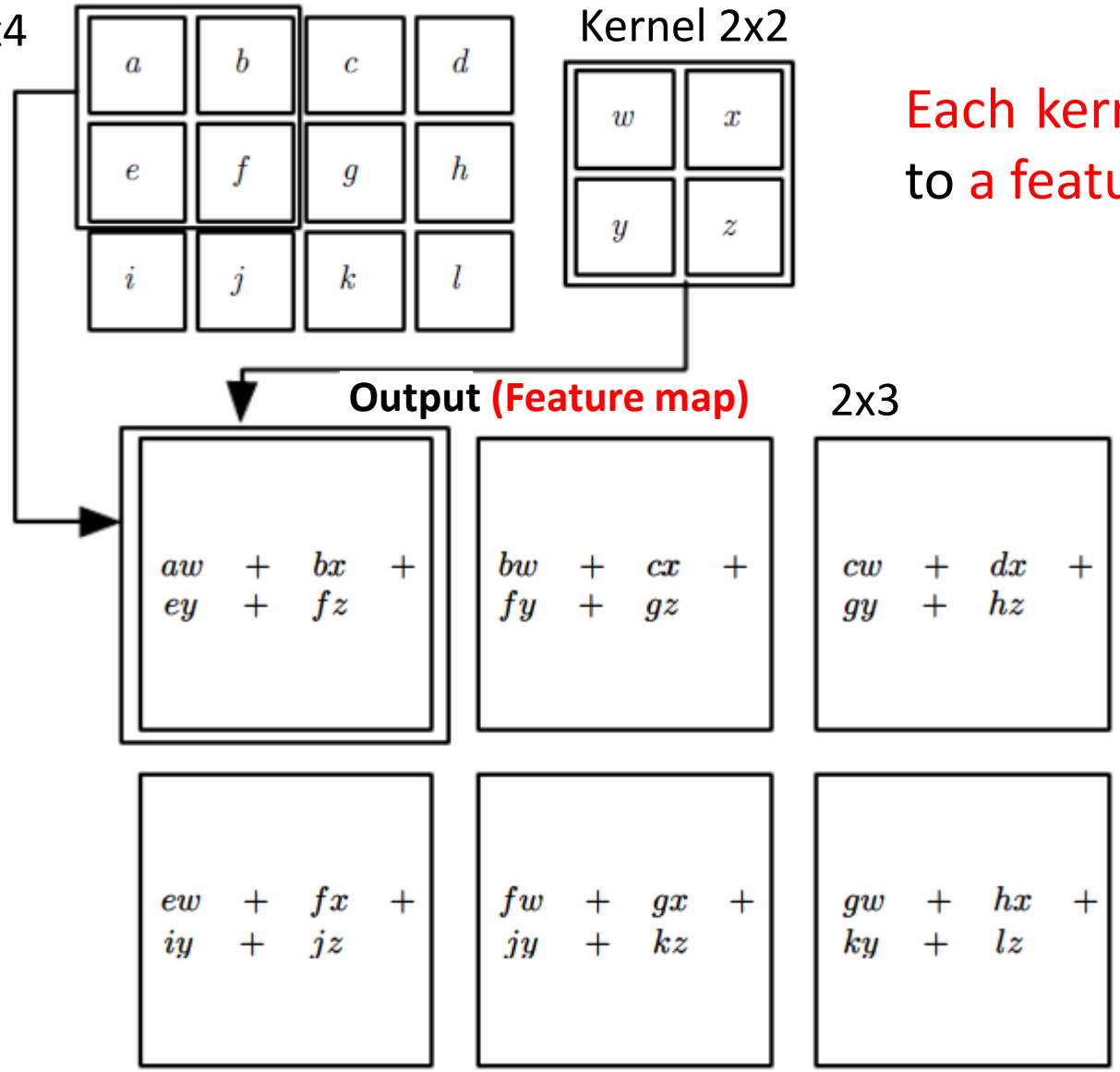
Discrete convolution

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

2D convolution

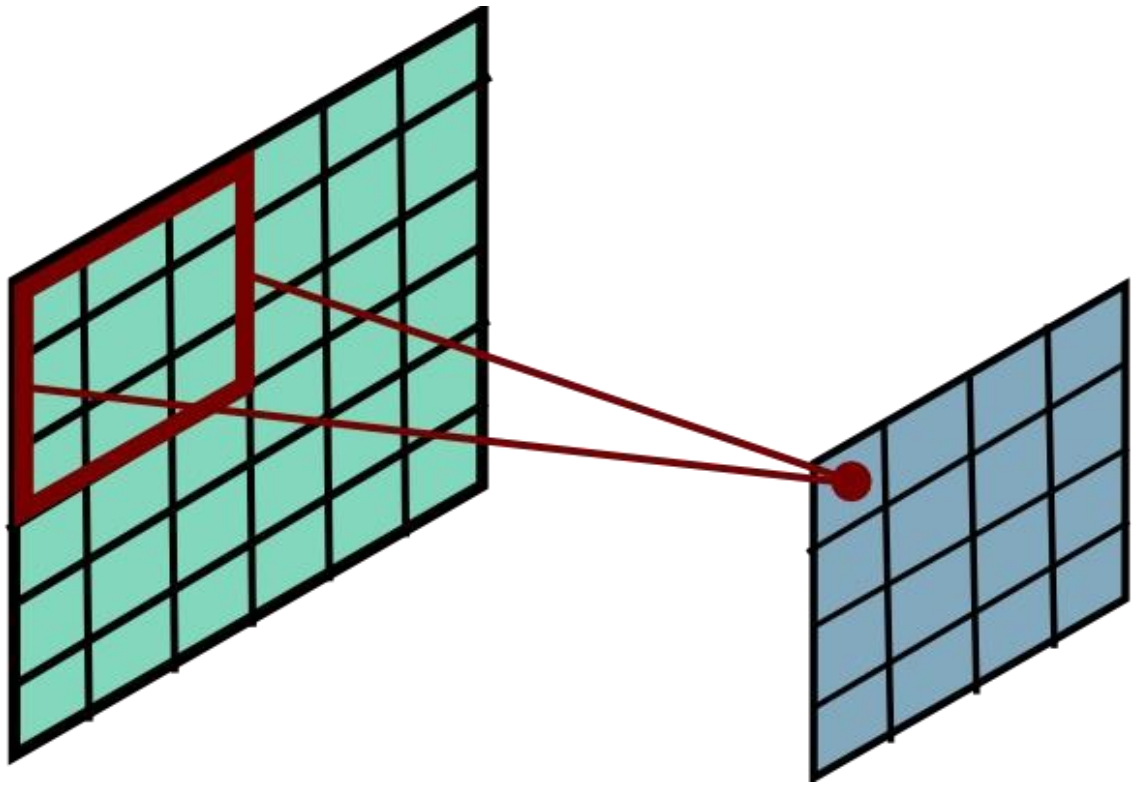
$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n)$$

Convolution Example

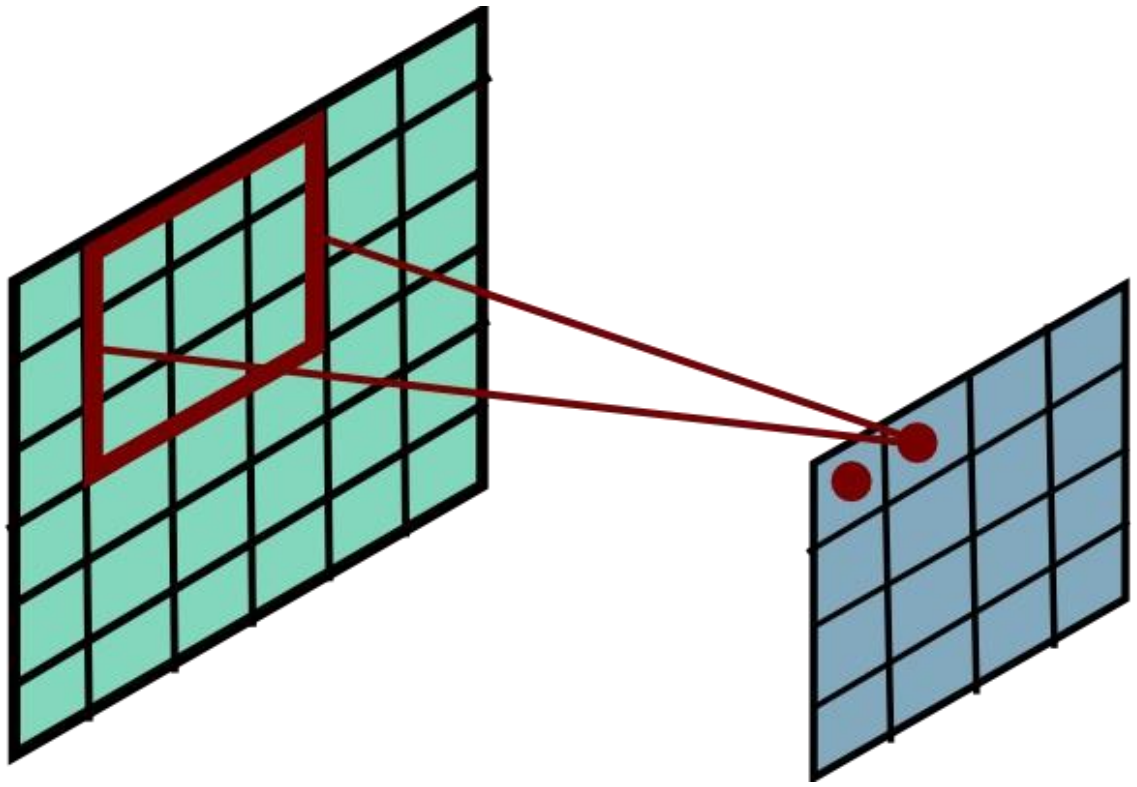


Each kernel corresponds to a feature map!

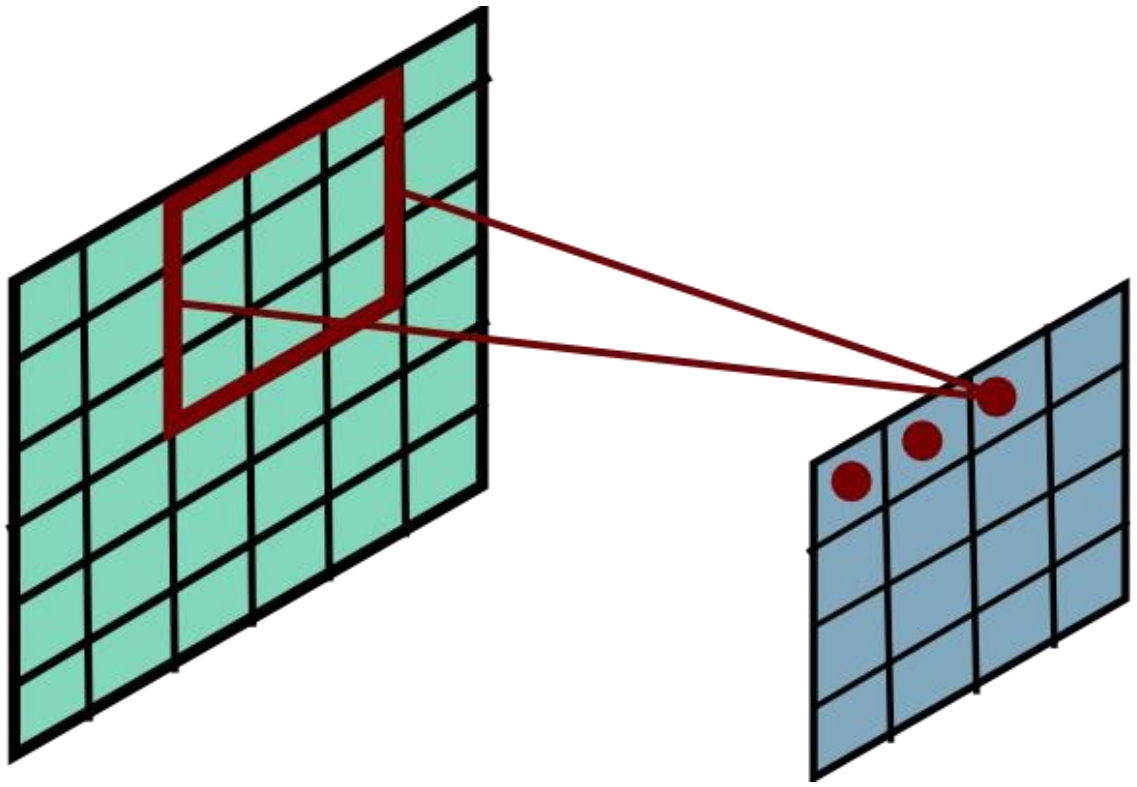
Convolutional Layer



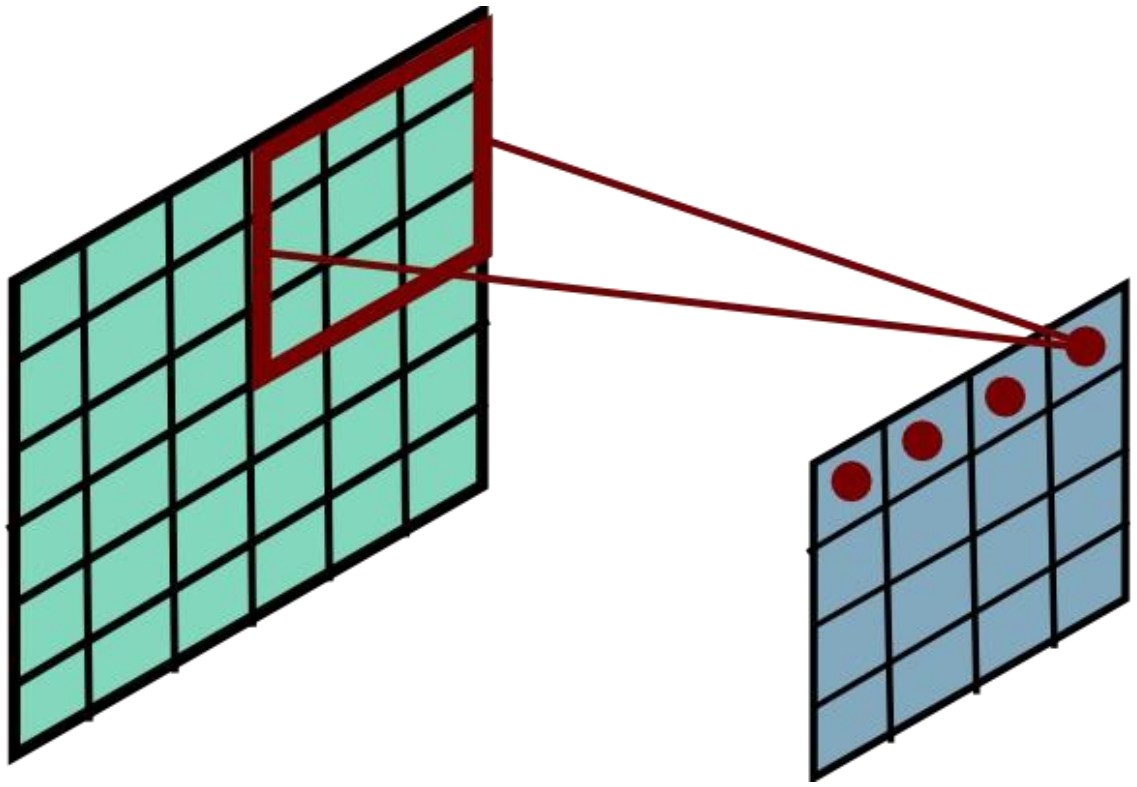
Convolutional Layer



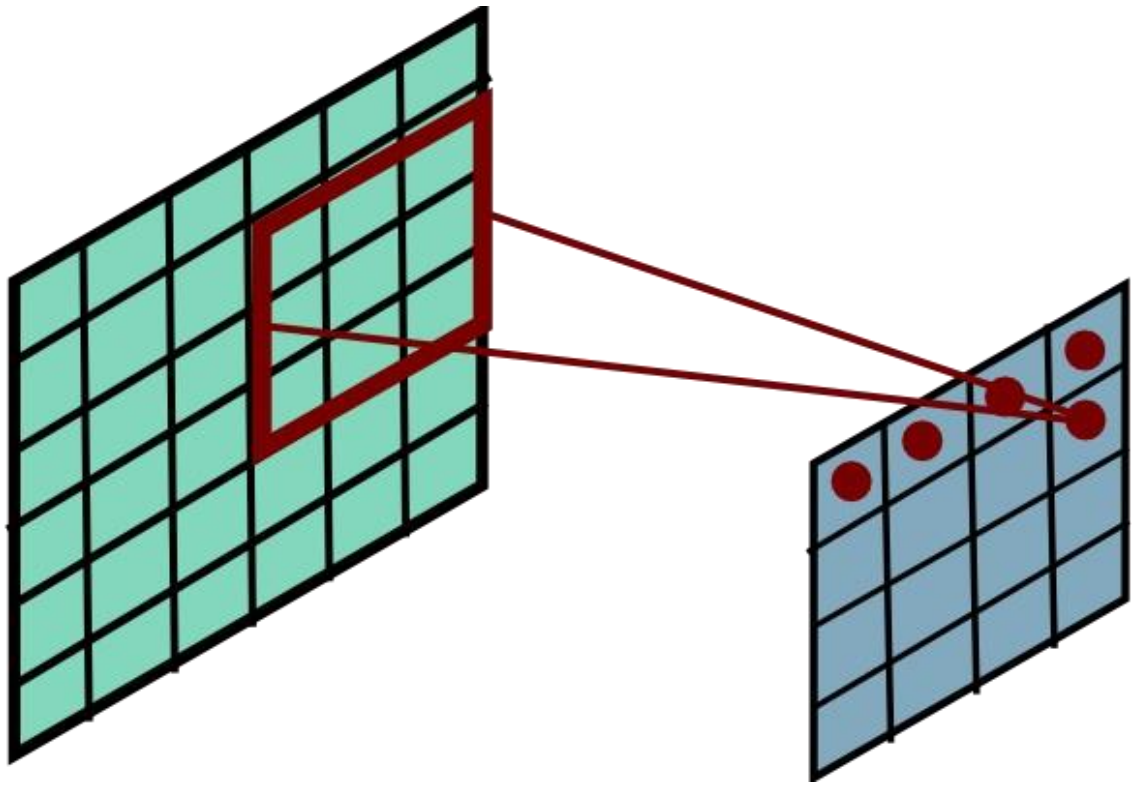
Convolutional Layer



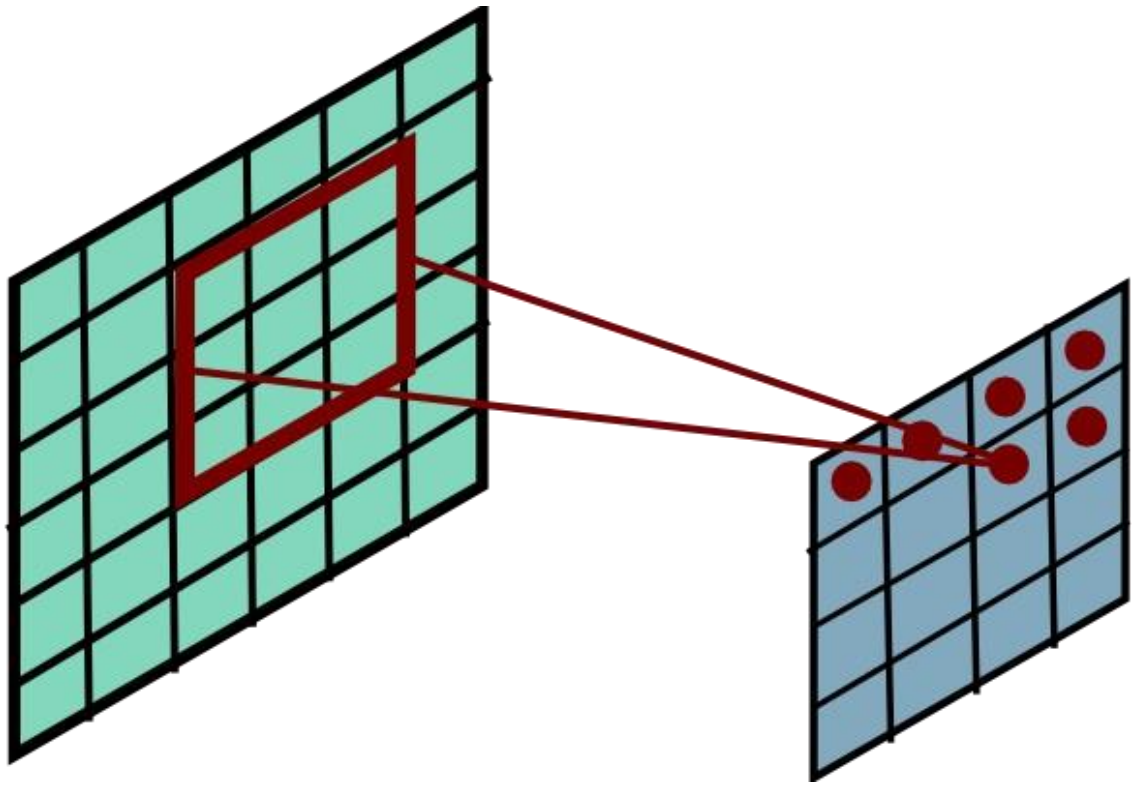
Convolutional Layer



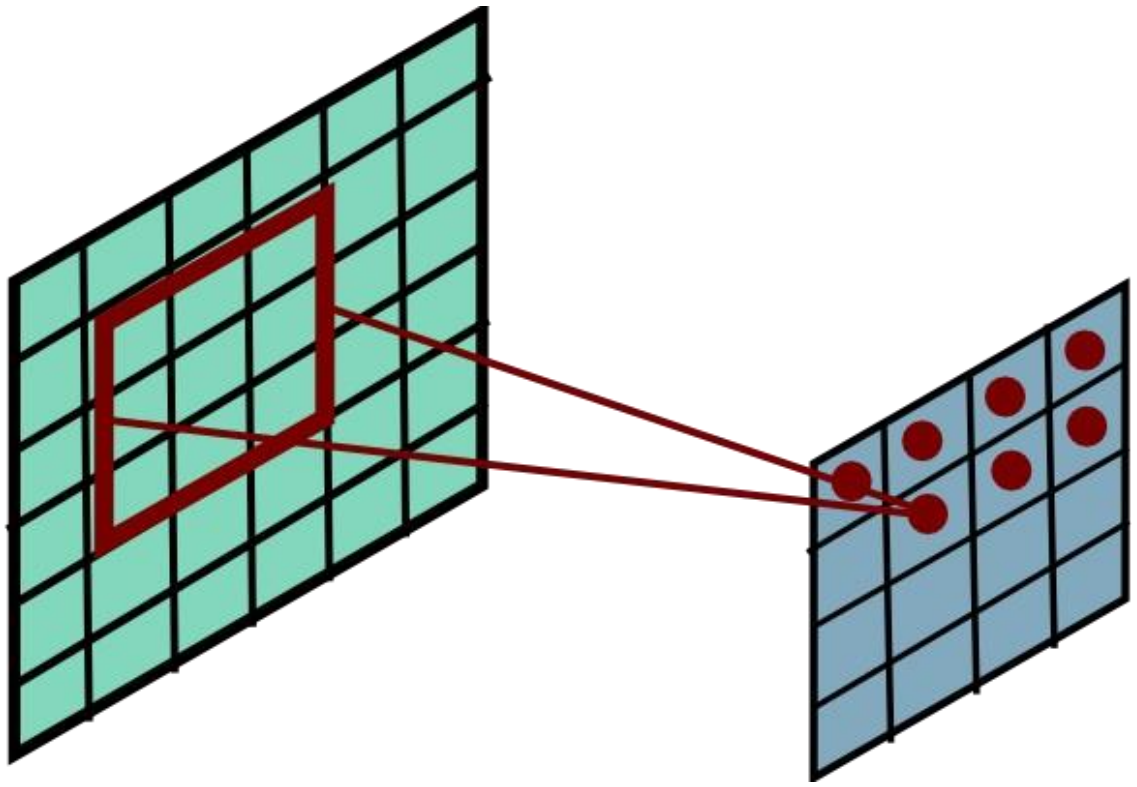
Convolutional Layer



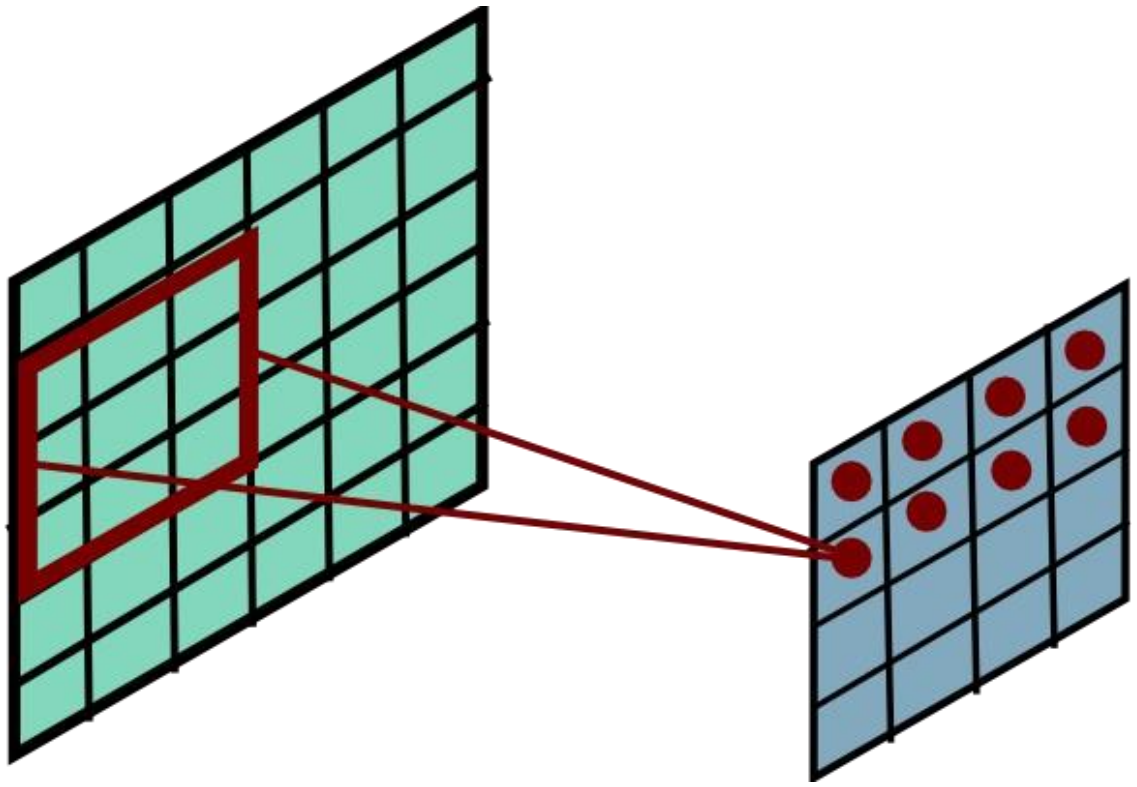
Convolutional Layer



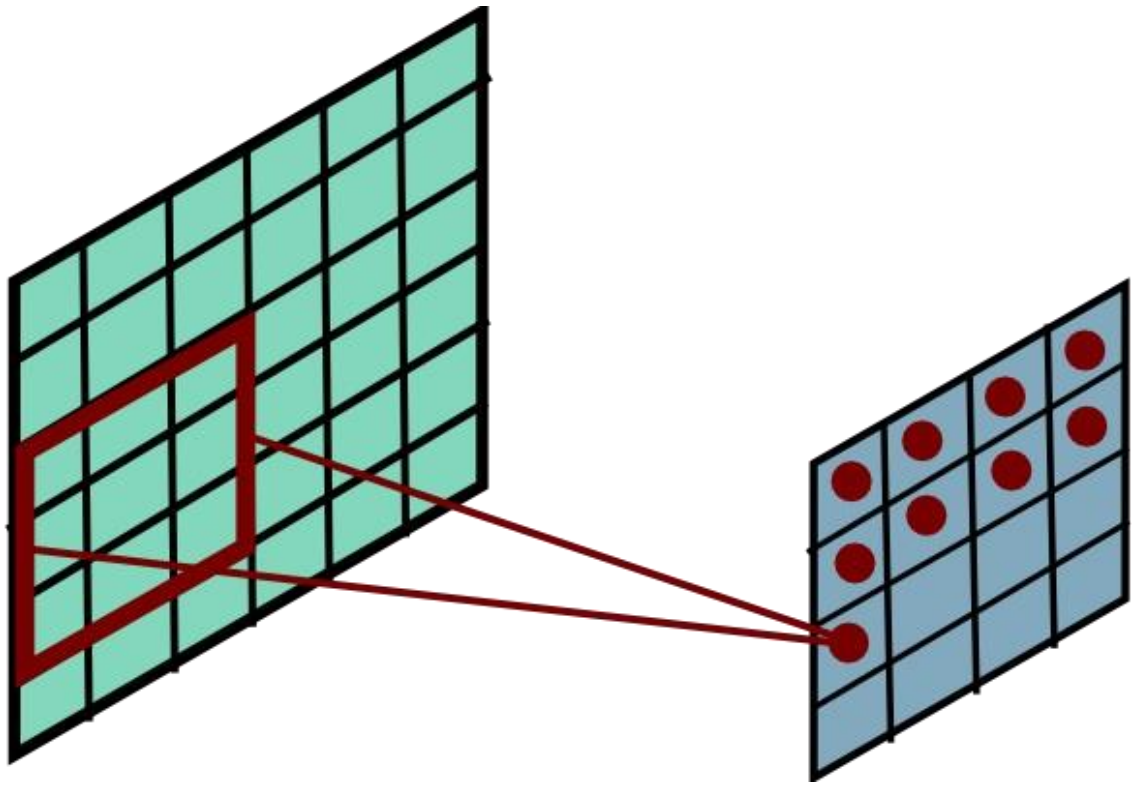
Convolutional Layer



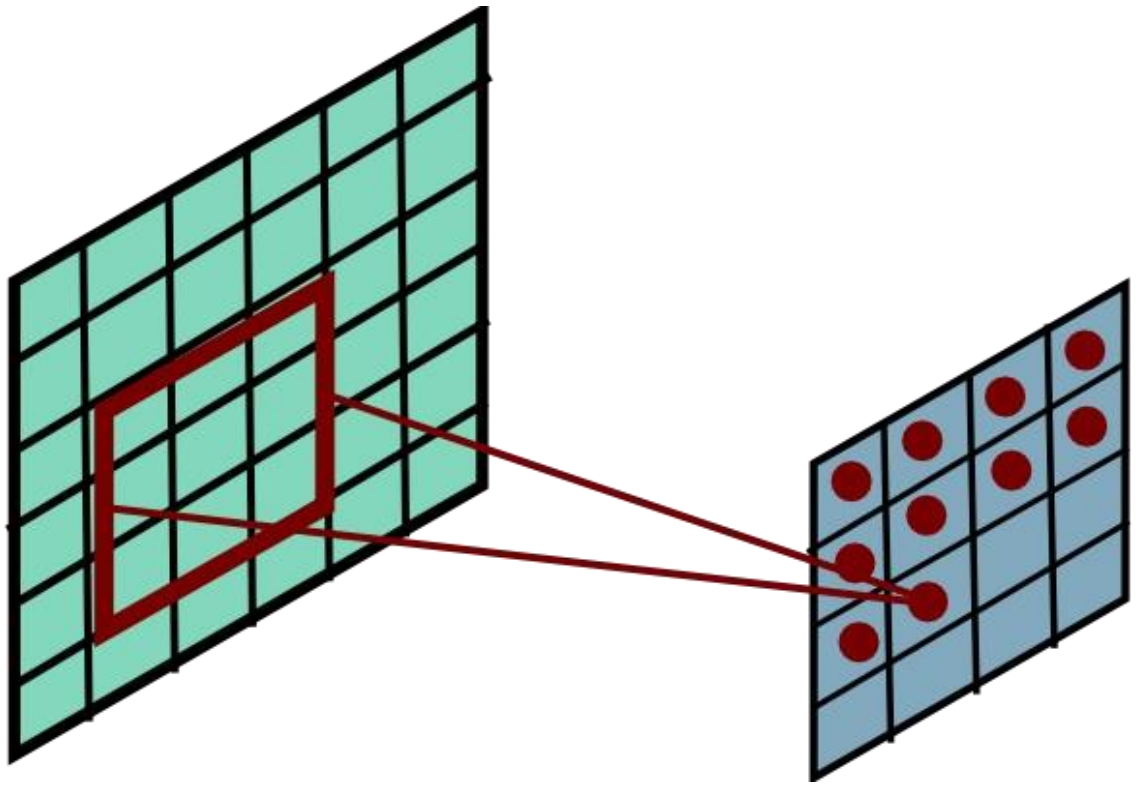
Convolutional Layer



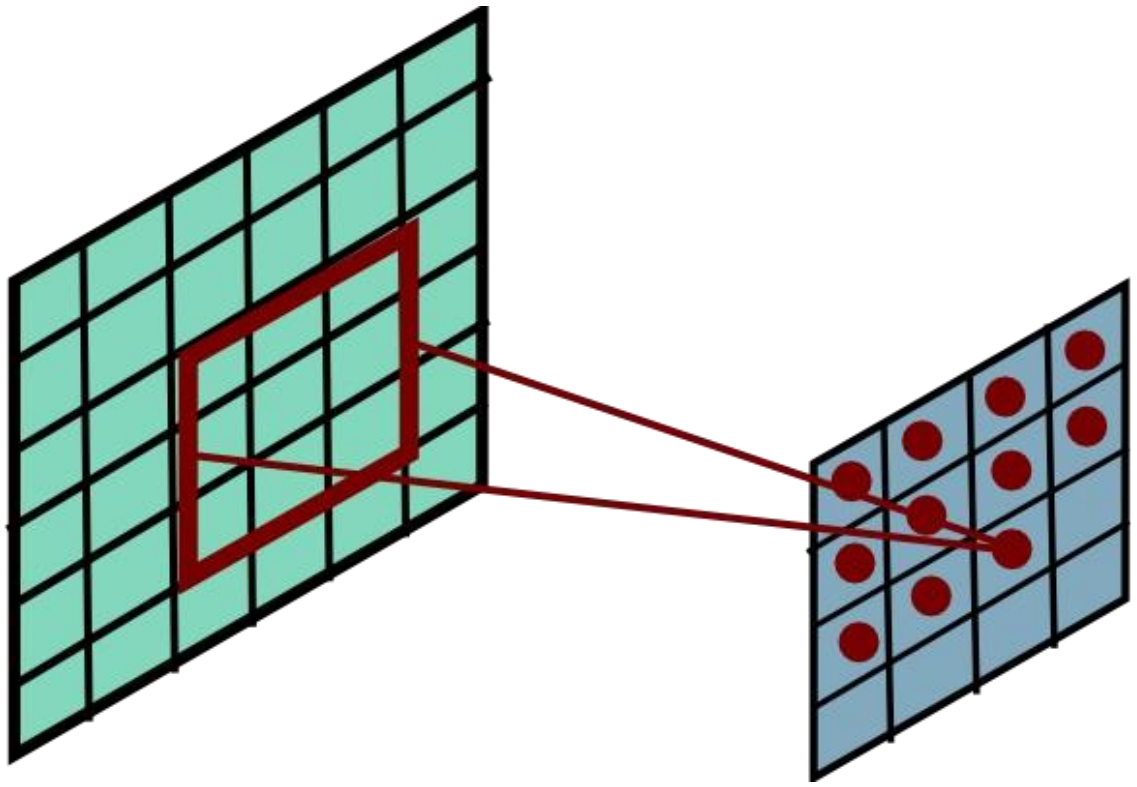
Convolutional Layer



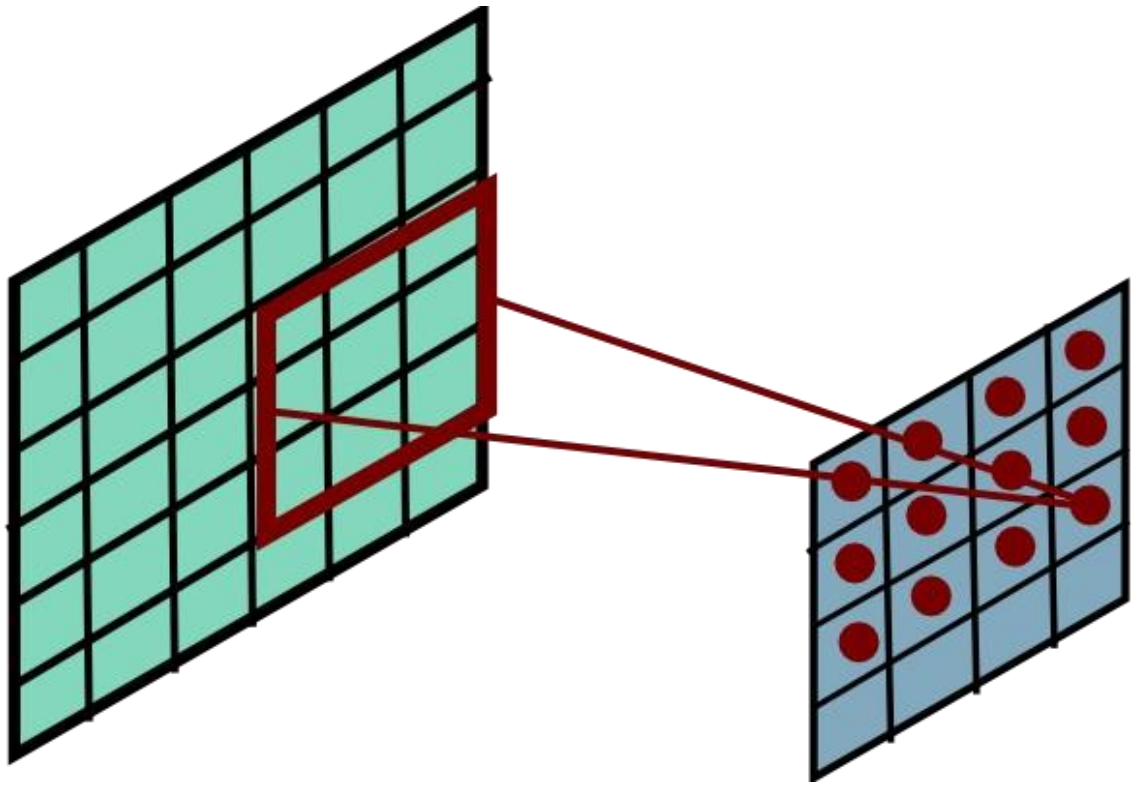
Convolutional Layer



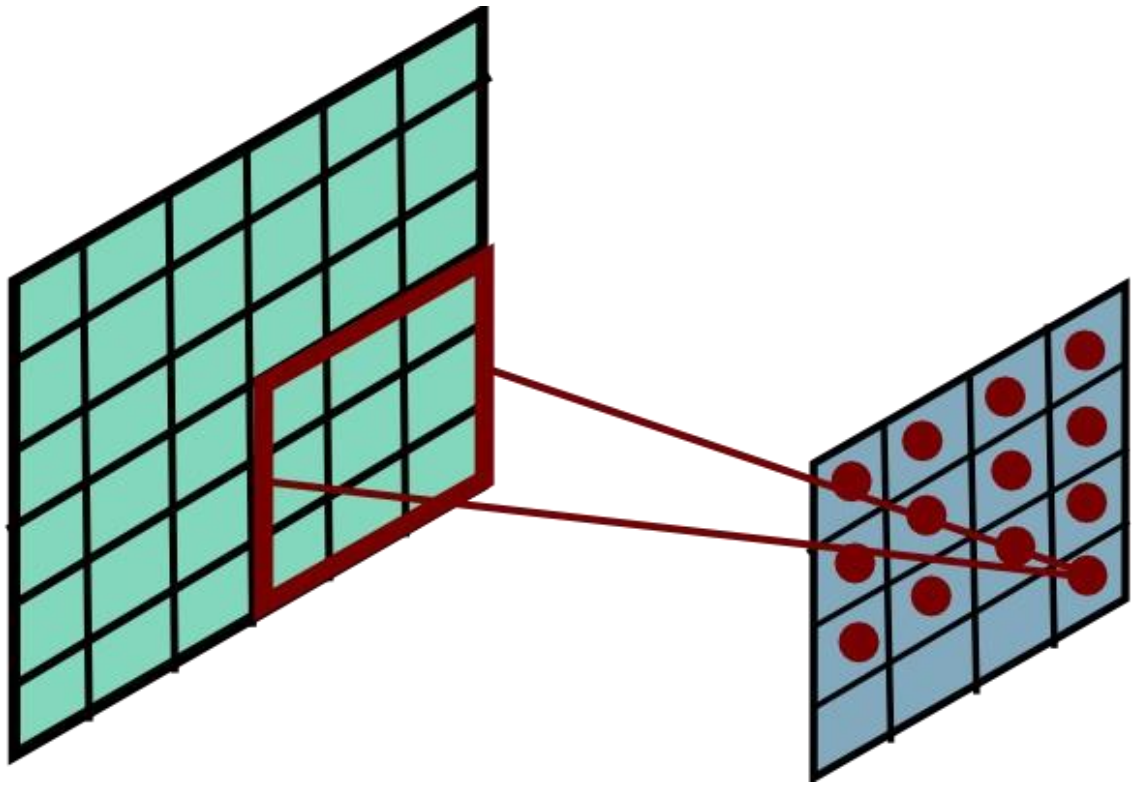
Convolutional Layer



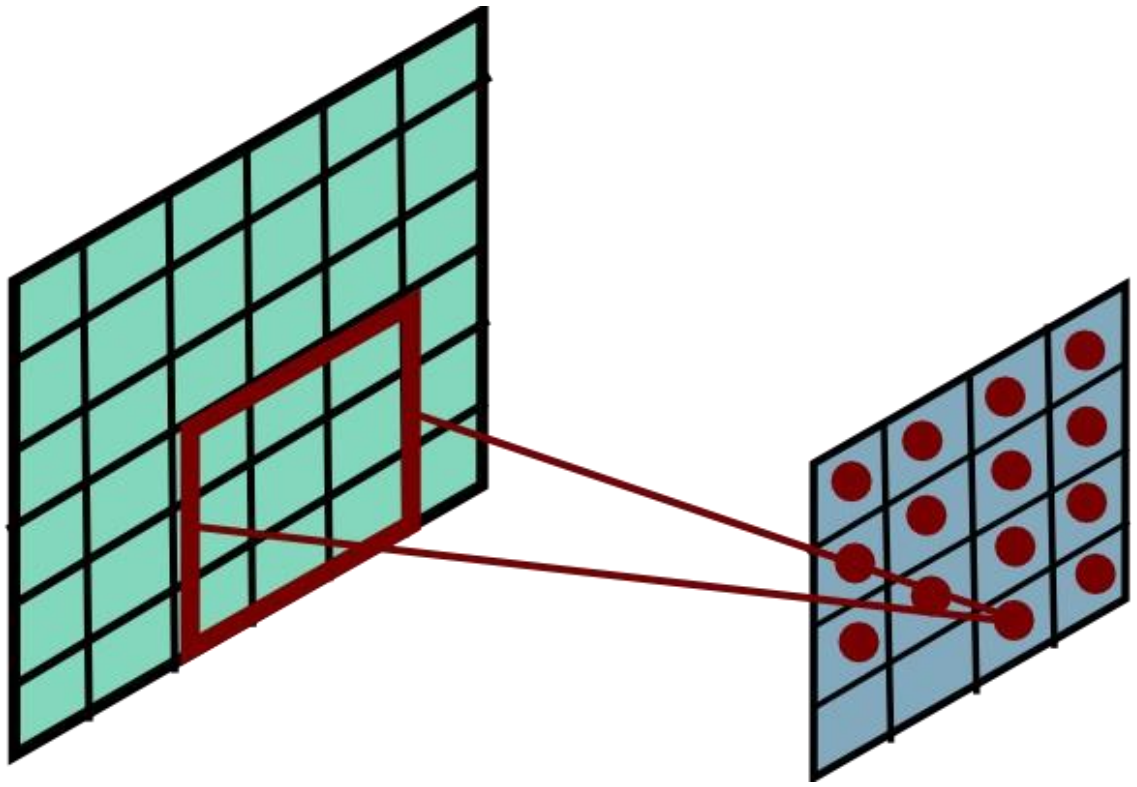
Convolutional Layer



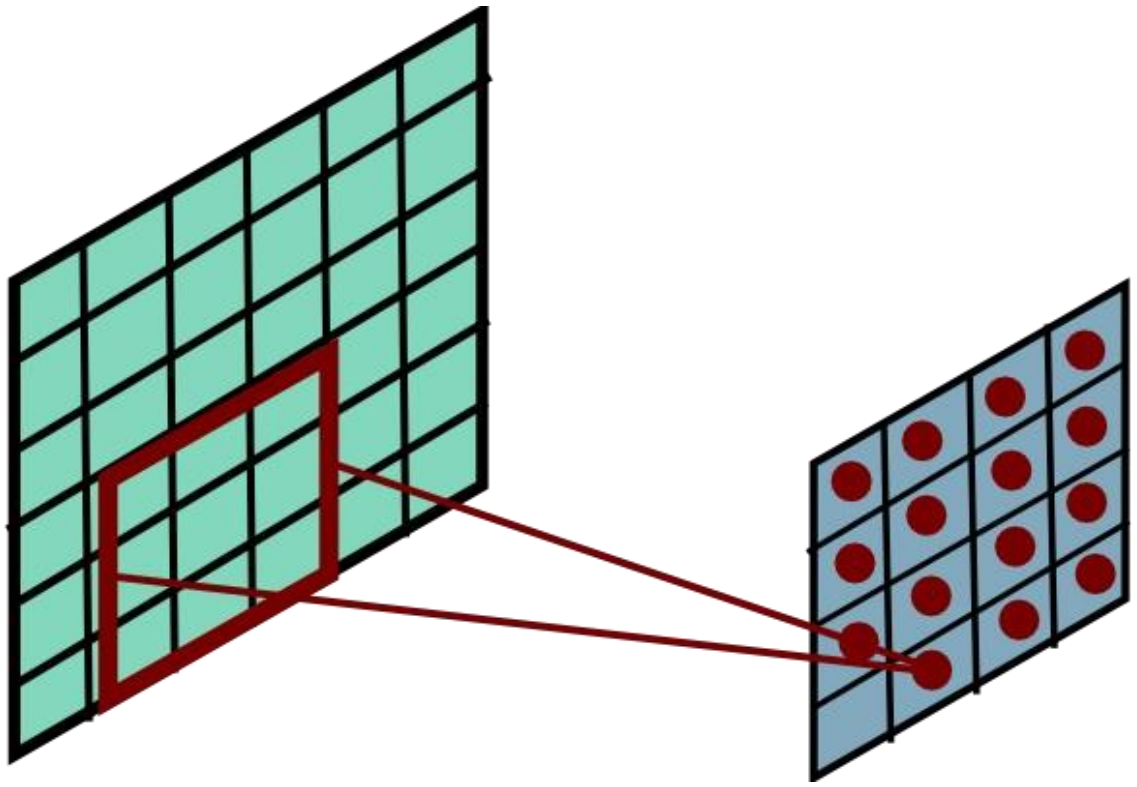
Convolutional Layer



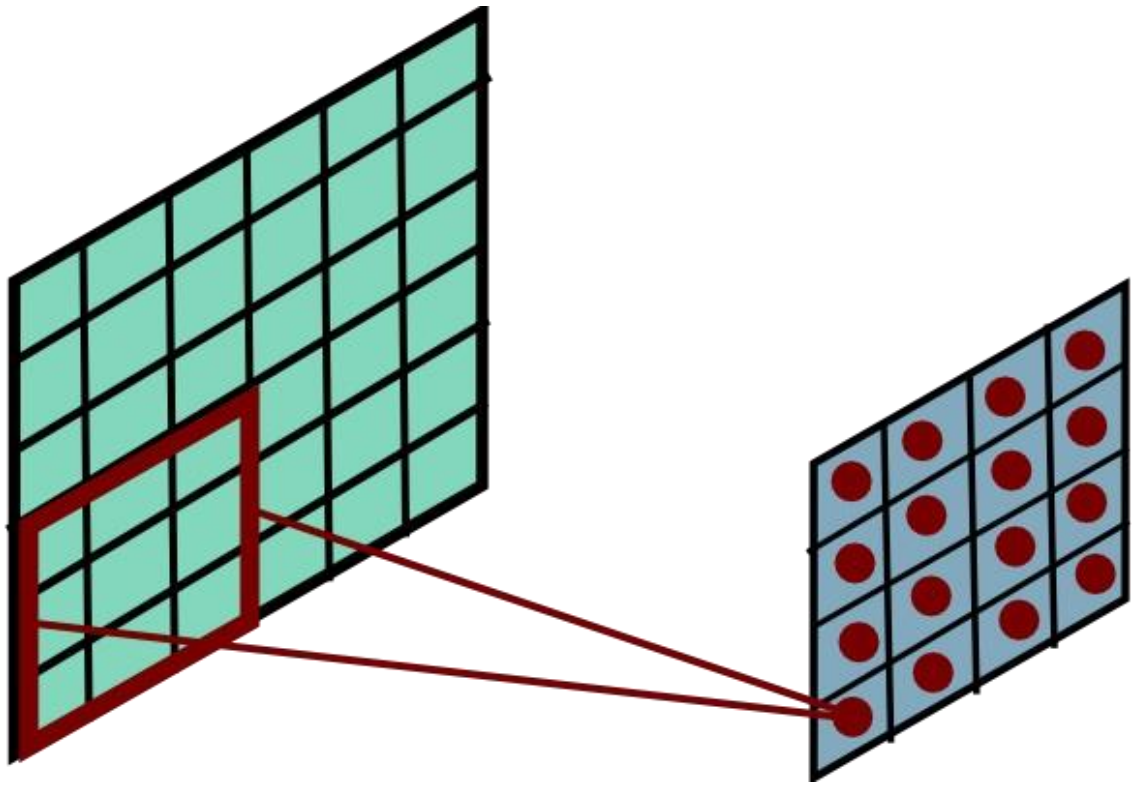
Convolutional Layer



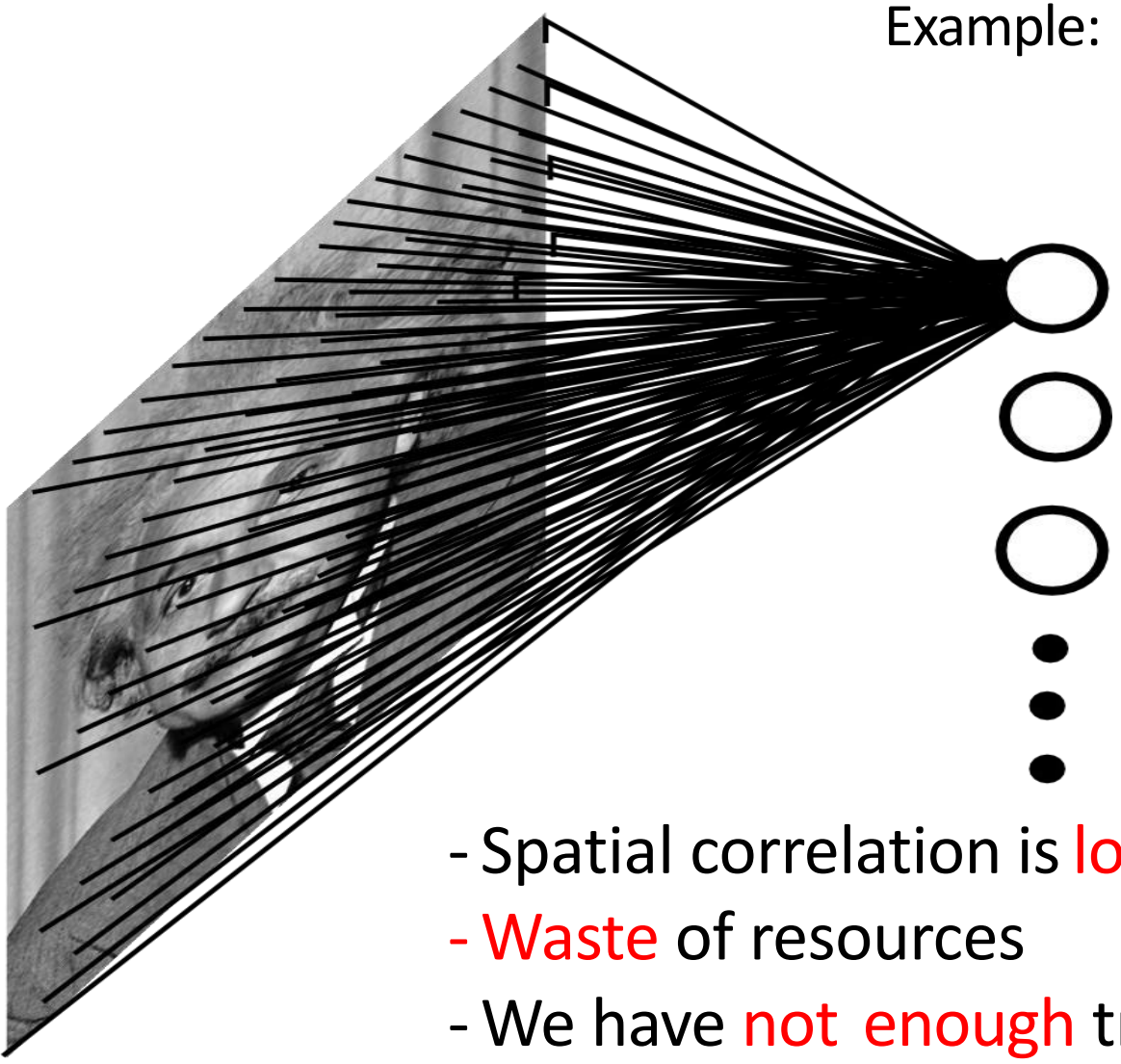
Convolutional Layer



Convolutional Layer



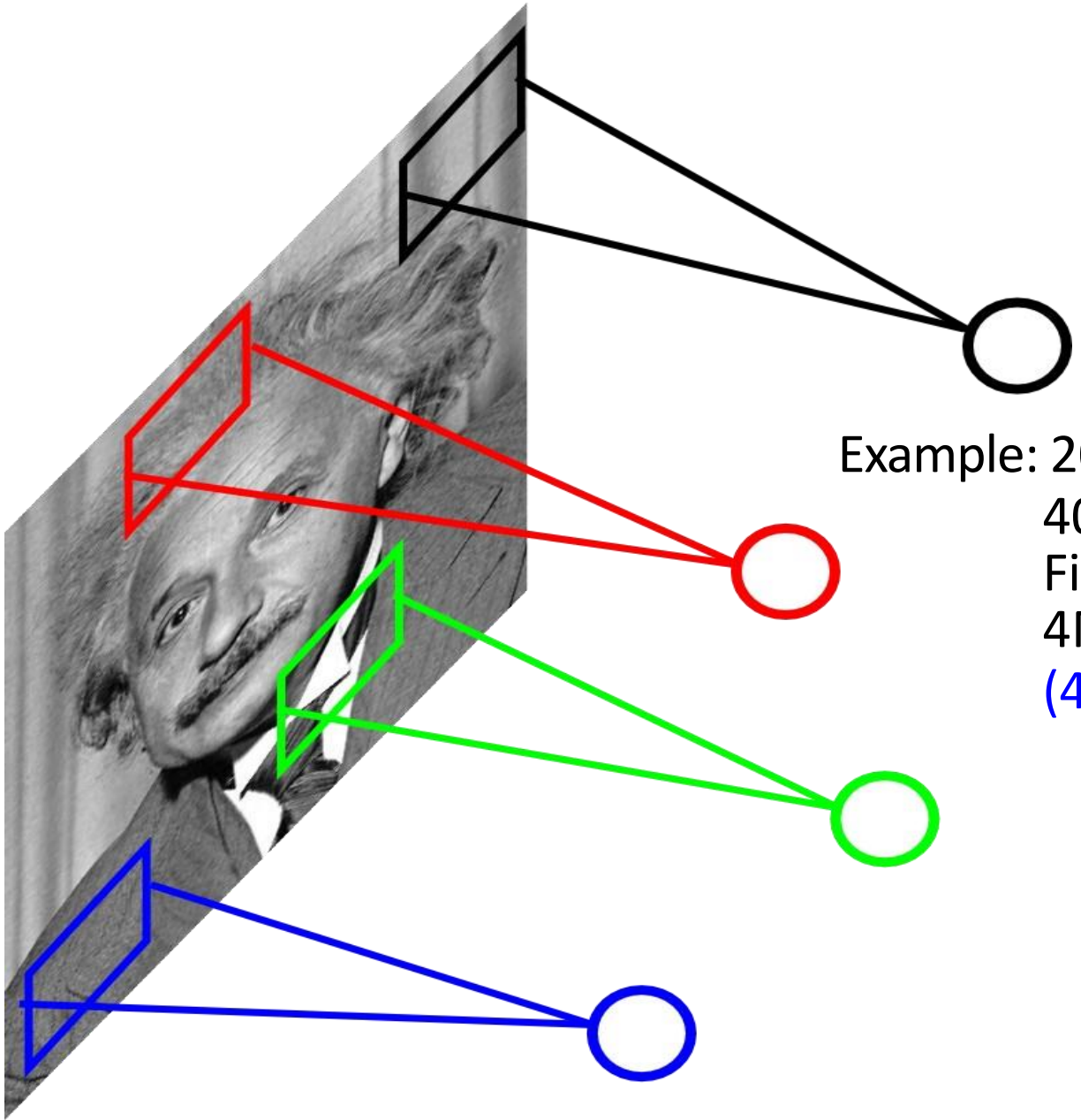
Fully Connected Layer



Example: 200x200 image
40K hidden units
~2B parameters!!!
(40Kx200x200)

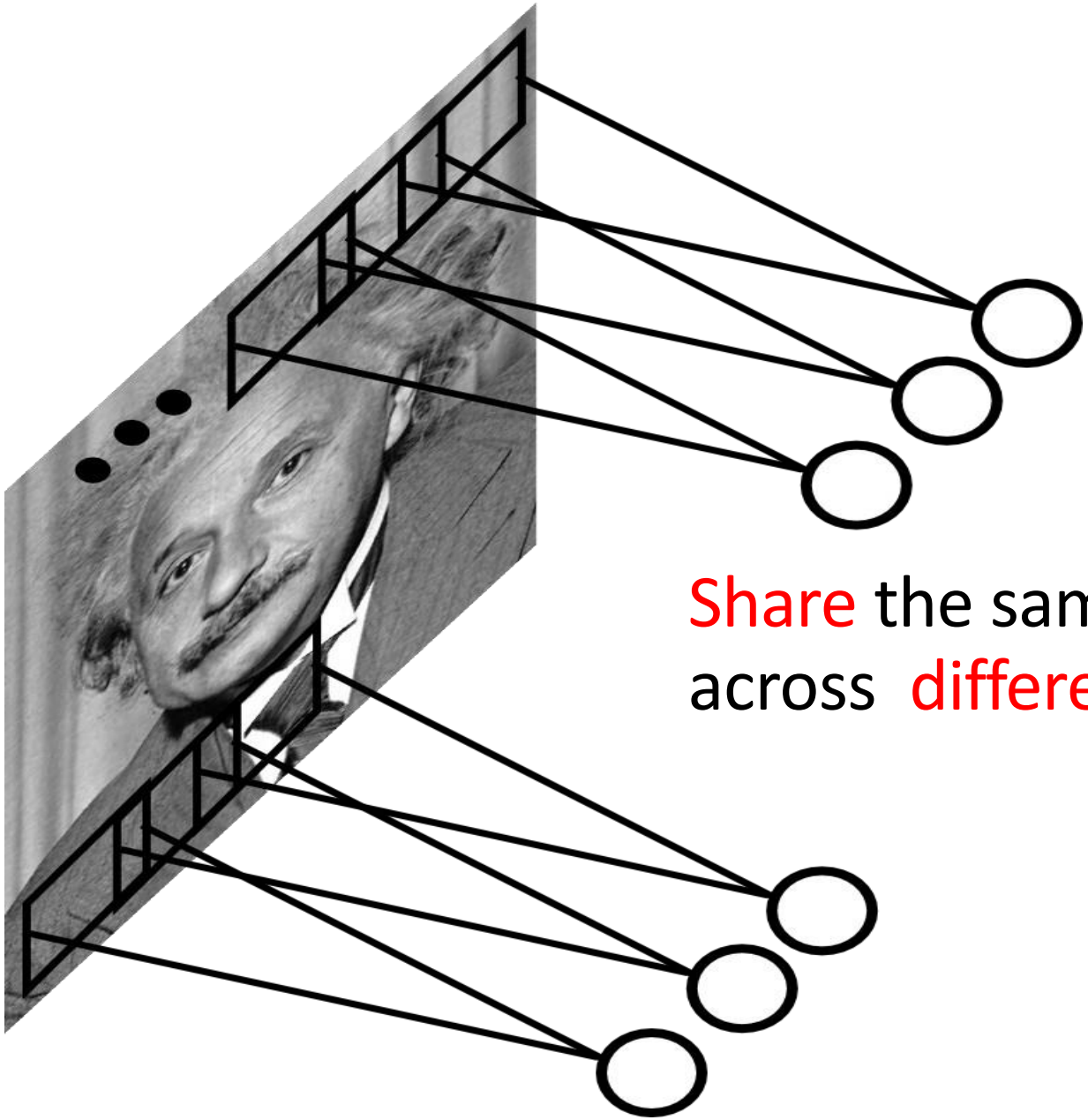
- Spatial correlation is local
- Waste of resources
- We have not enough training samples anyway..

Locally Connected Layer



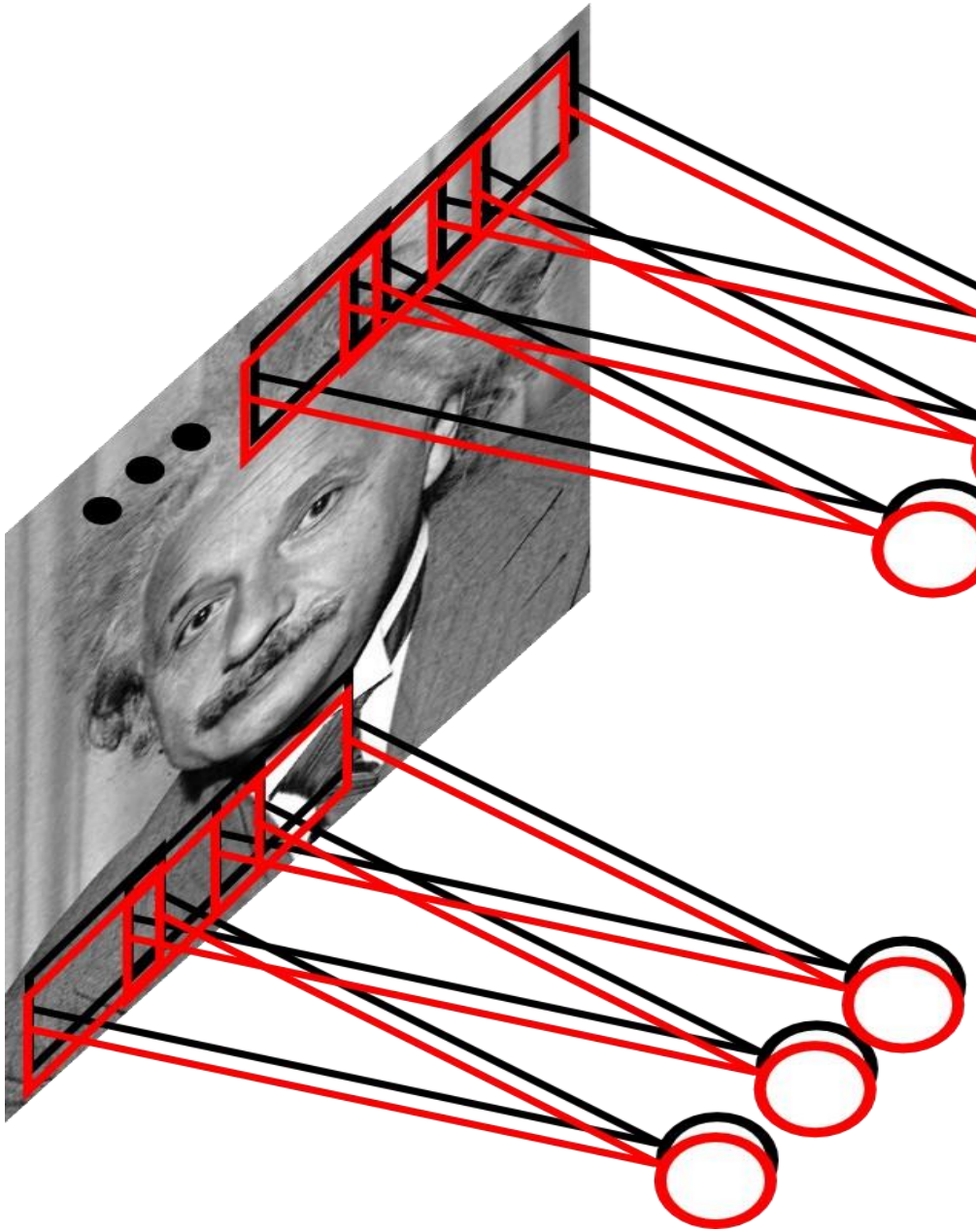
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters
(40Kx10x10)

Parameter Sharing



Share the same parameters across different locations.

Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Typical Layer Structure

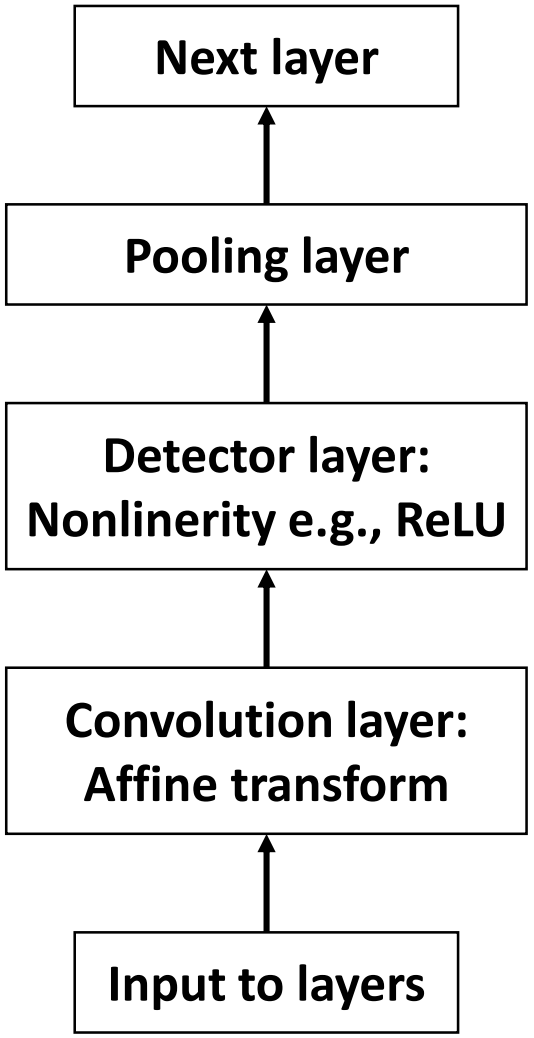
Three stages

1) Perform convolutions.

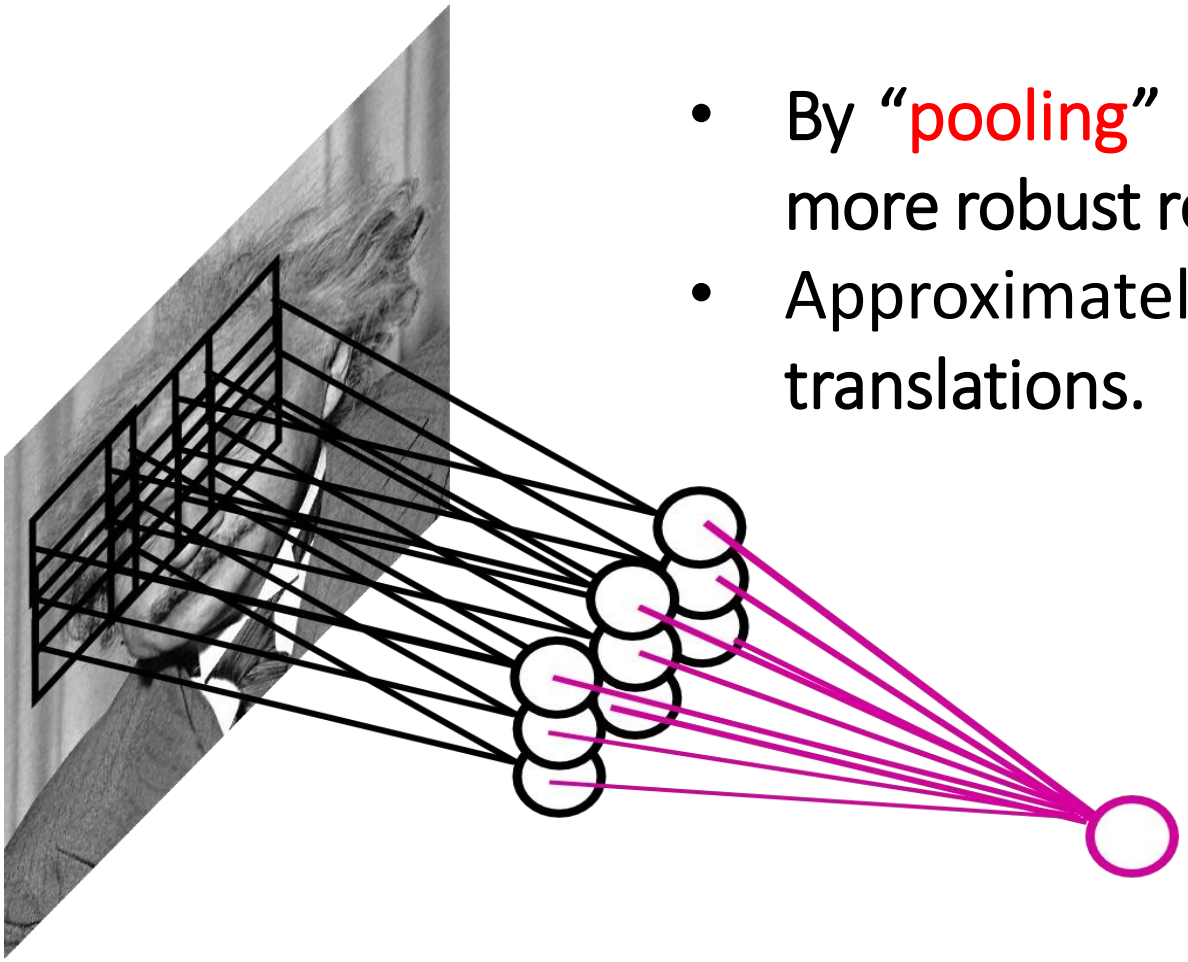
2) Run through a nonlinear activation function, e.g., ReLu. (detector stage).

3) Pooling function to modify the output.

With a summary statistic of the nearby outputs.



Pooling Layer



- By "pooling" (e.g., max), we gain more robust representations.
- Approximately *Invariant* to small translations.

Note: When a task involves incorporating information from very distant locations in the input, then the prior imposed by CNN may be inappropriate.

Pooling Strategies

Max-pooling: x-axis, y-axis

$$h_j^n(x, y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

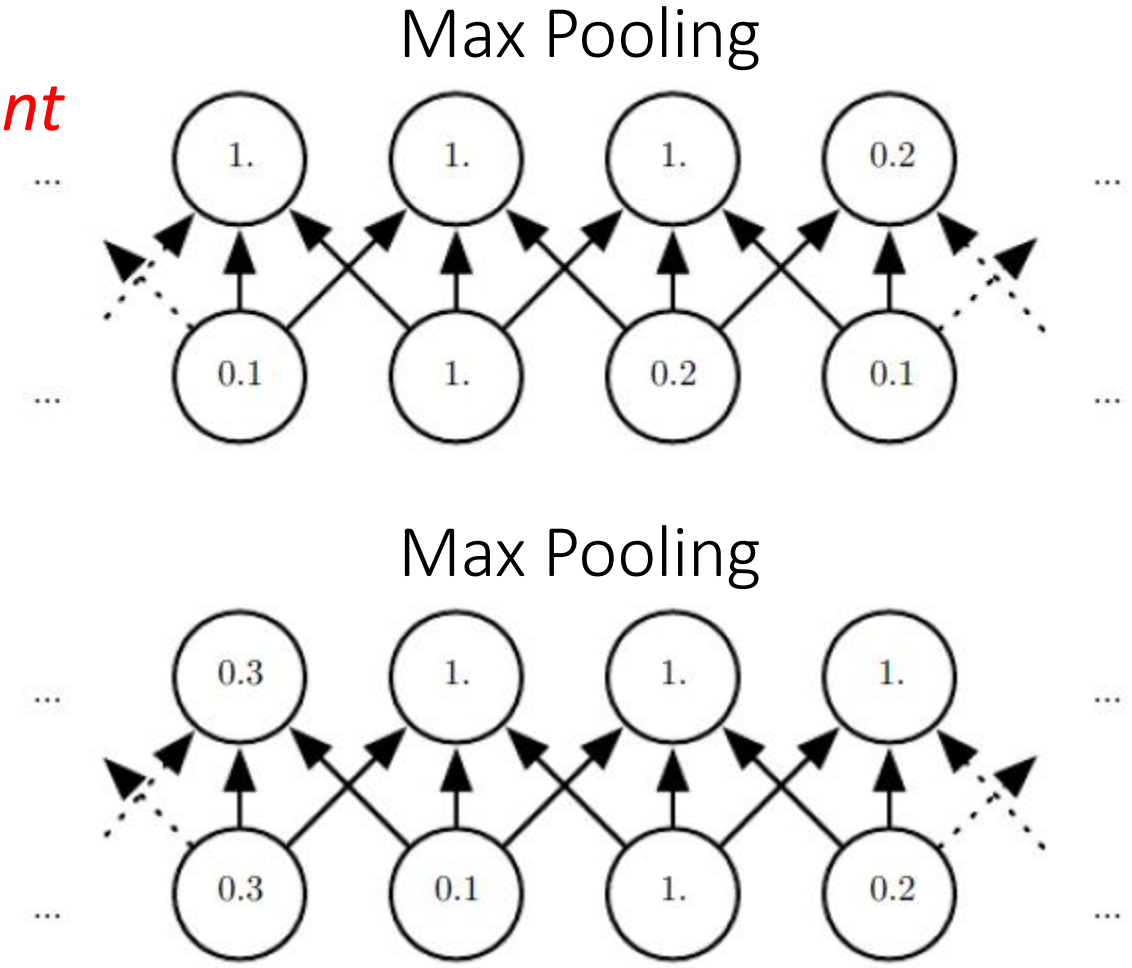
L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

Max Pooling

Approximately *Invariant*

to small translations



Care more about whether some feature is present than exactly where it is.

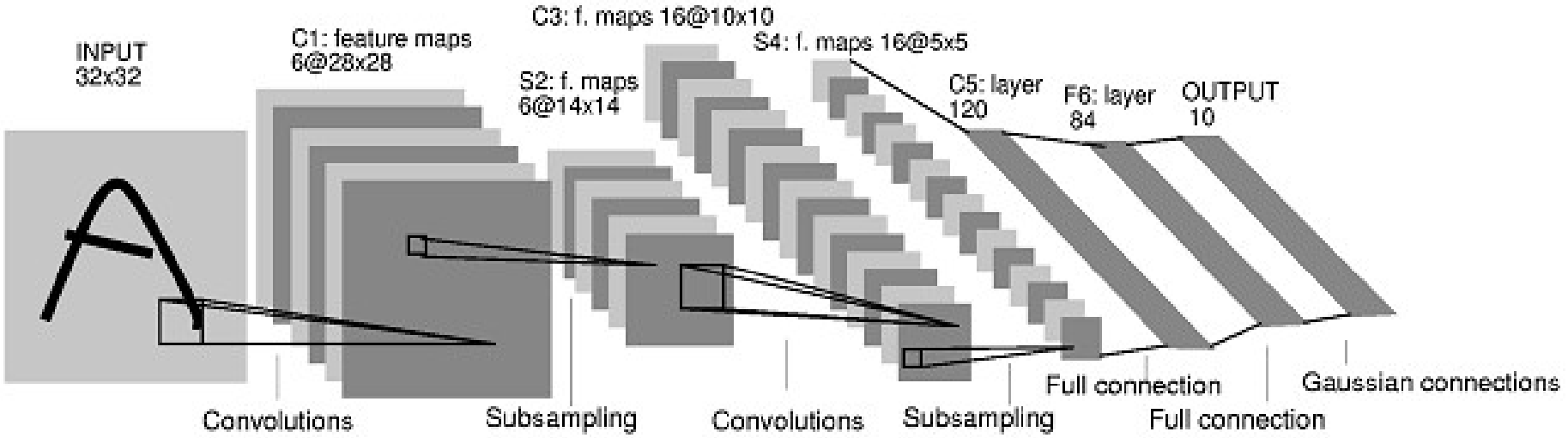
How to Train CNN…

- Convolutional Neural Networks are a special kind of multi-layer neural networks.

- Just back-propagation algorithm…

LeNet 5, LeCun 1998

handwritten and machine-printed character recognition.



Input: 32x32 pixel image

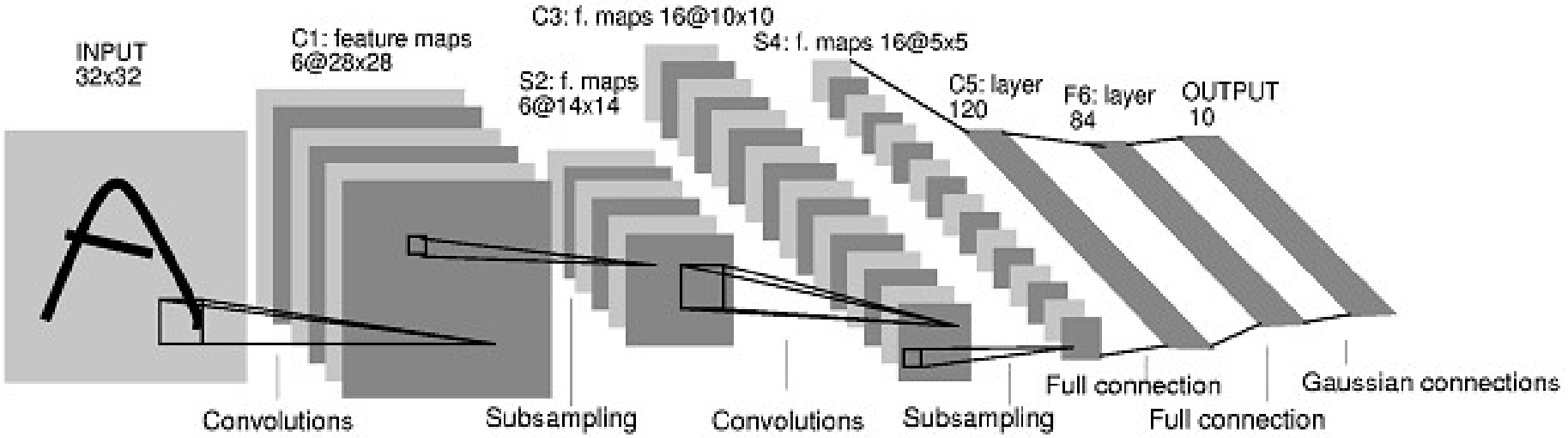
Cx: Convolutional layer

Sx: Subsample (Pooling) layer

Fx: Fully connected layer

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, Proceedings of the IEEE, 86(11):2278-2324, **1998**

LeNet 5, LeCun 1998



C1: 6 Kernel 5x5, outputs 6 feature maps of size 28x28.

S2: Pooling 2x2, outputs 6 feature maps of size 14x14.

C3: 16 Kernel 5x5, outputs 16 feature maps of size 10x10.

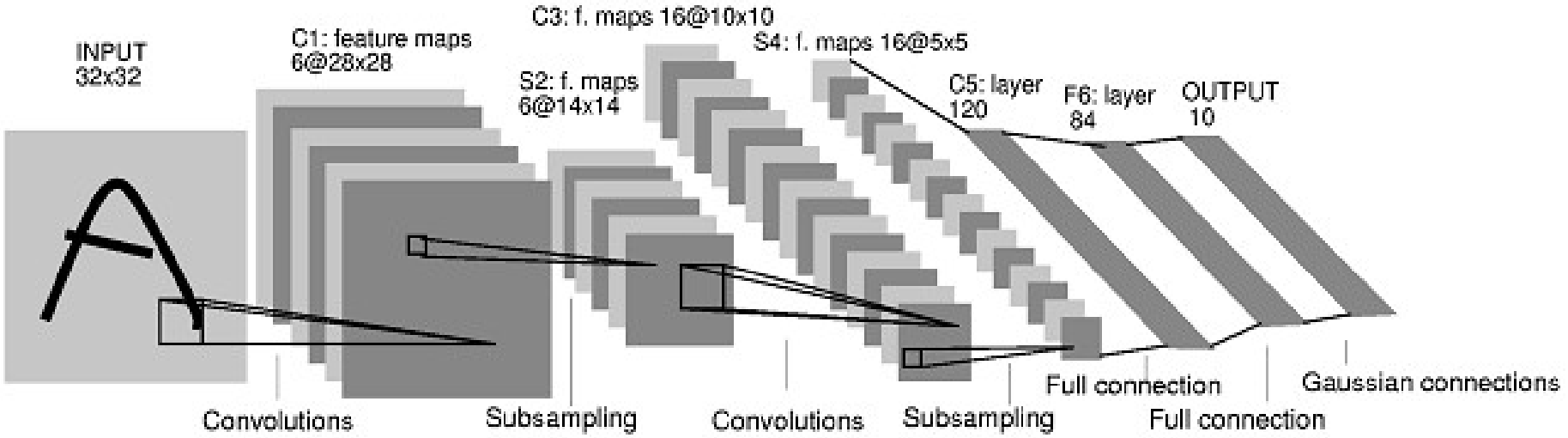
S4: Pooling 2x2, outputs 16 feature maps of size 5x5.

C5: 120 Kernel 5x5, outputs 120 feature maps of size 1x1.

F6: 84 fully connected units.

Output layer: 10RBF (One for each digit).

LeNet 5, LeCun 1998



Note: After one stage,

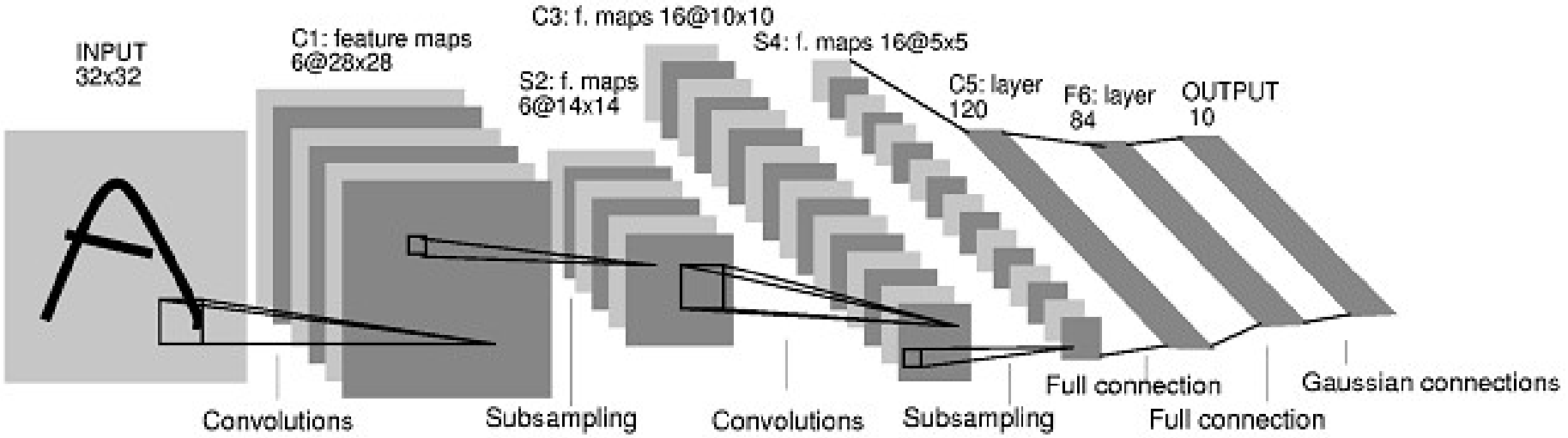
#feature maps →increased (conv. layer)

Spatial resolution → decreased (stride in conv. and pooling).

Reasons:

- gain invariance to spatial translation (pooling layer)
- increase specificity of features.

LeNet 5, LeCun 1998



Note: After several stages,
the spatial resolution is greatly reduced (about 5x5),
the number of feature maps is large (several hundreds).

Not make sense to convolve again (no translation invariance).
Fed into several fully connected layers.

Data Representation

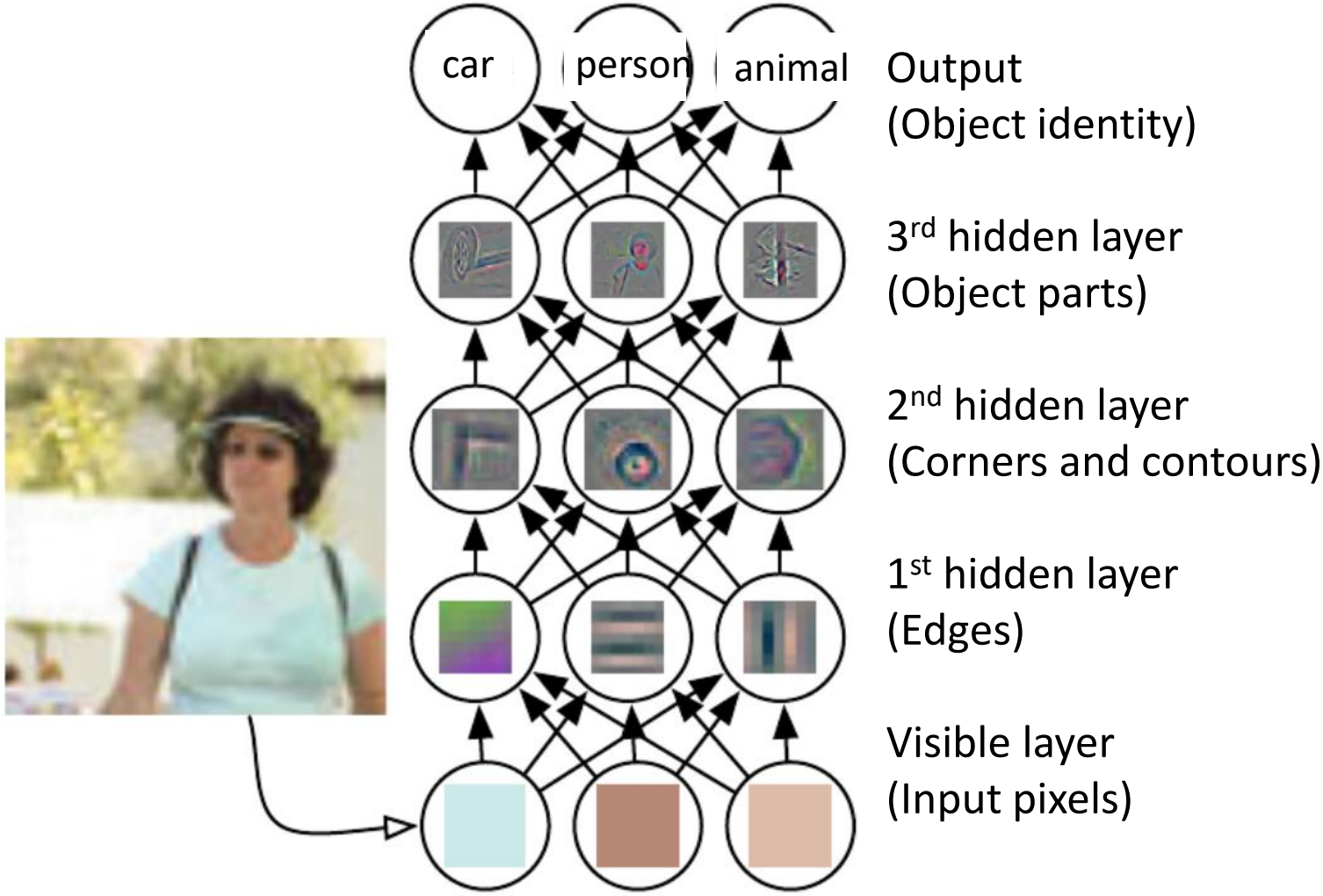


Illustration of a deep learning model.

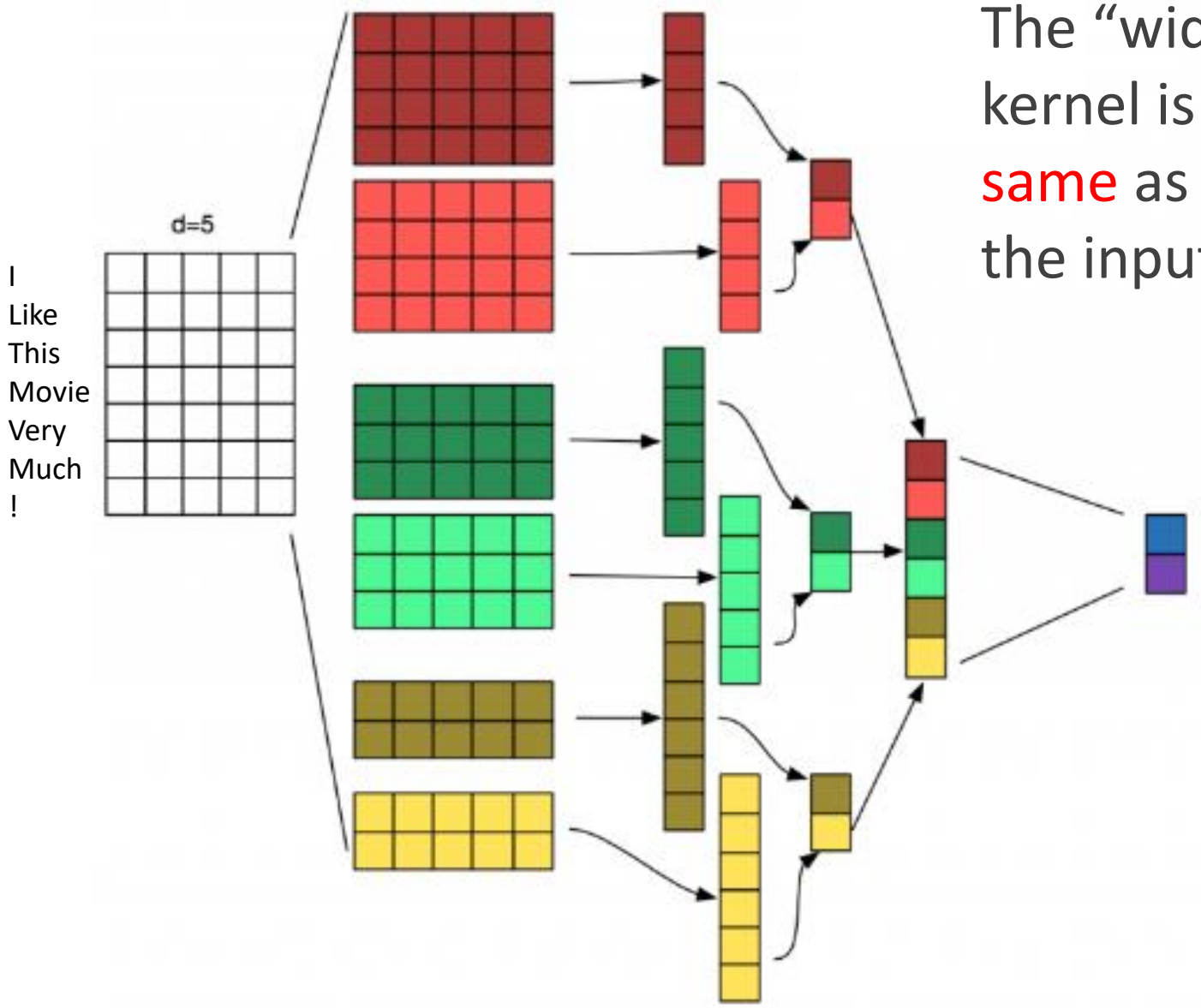
Applications for CNN

- Computer Vision
  - Image classification, Object detection, etc.

How about NLP? Not Intuitive.

- Sentiment Analysis ✓
- Spam Detection ✓
- Topic Categorization ✓
- Entity Extraction Harder
- PoS Tagging Harder
- ……

Lose the local order of words



The “width” of the kernel is usually the same as the width of the input matrix.

Illustration of a CNN architecture for sentence classification.

Short Summary

- Convolutional Networks
  - Locally connected
  - Parameter sharing, reduce parameters
  - Kernel (filter)
    - Can be multiple kernels each layer
  - Output (feature map)

- Pooling
  - Max, average, L2 norm…
  - Approximately Invariant to small translations.