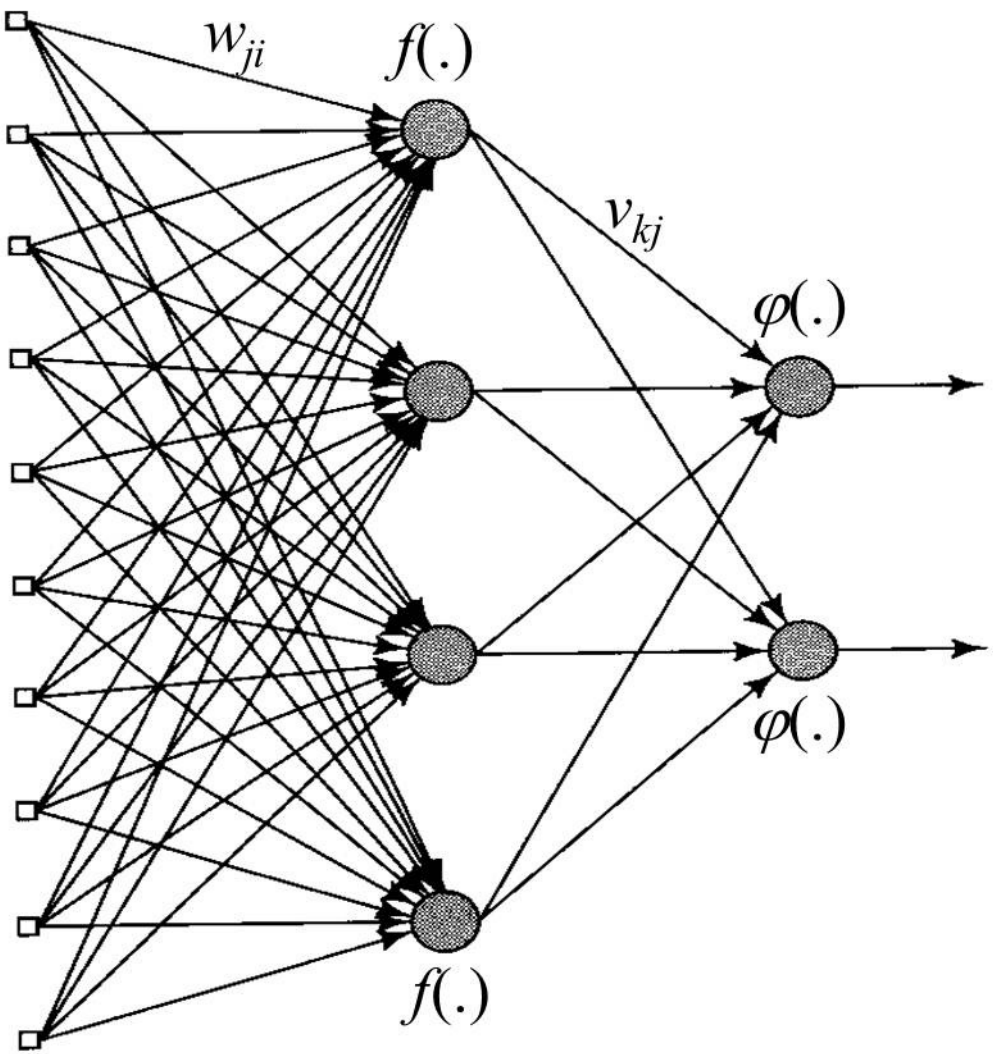
Neural Networks and Deep Machine Learning: from MLP to CNN

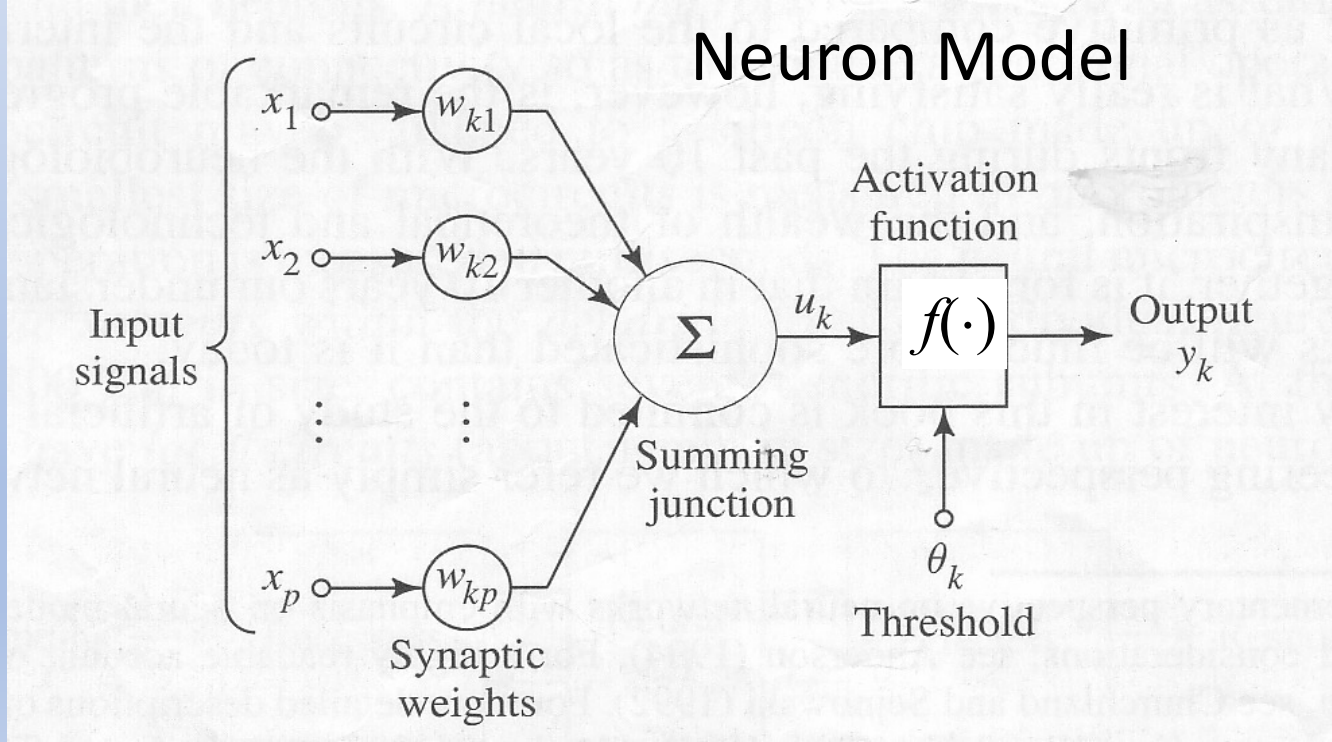
Outline

- Network Architecture and Neuron Model
- Optimization and Iterative (Error Correction) Learning
- Multilayer Perceptron (MLP) and Backpropagation Learning Algorithm
- Understanding Deep Learning and Convolutional neural Networks (CNN)

Neural Networks and Deep CNN



Network architecture of artificial neural networks (ANN), or simply, neural networks (NN)



Neural Networks and Deep CNN -- Neuron Model

In mathematical terms, we can describe the neuron as:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

$$y_k = f(u_k - \theta_k)$$

Where $x_1, x_2, \ldots, x_p$ are the input signals, $w_{k1}, w_{k2}, \ldots, w_{kp}$ are the weights of neuron k, u_k is the linear combiner output, θ_k is the threshold, $f(.)$ is the activation function and y_k is the neurons output.

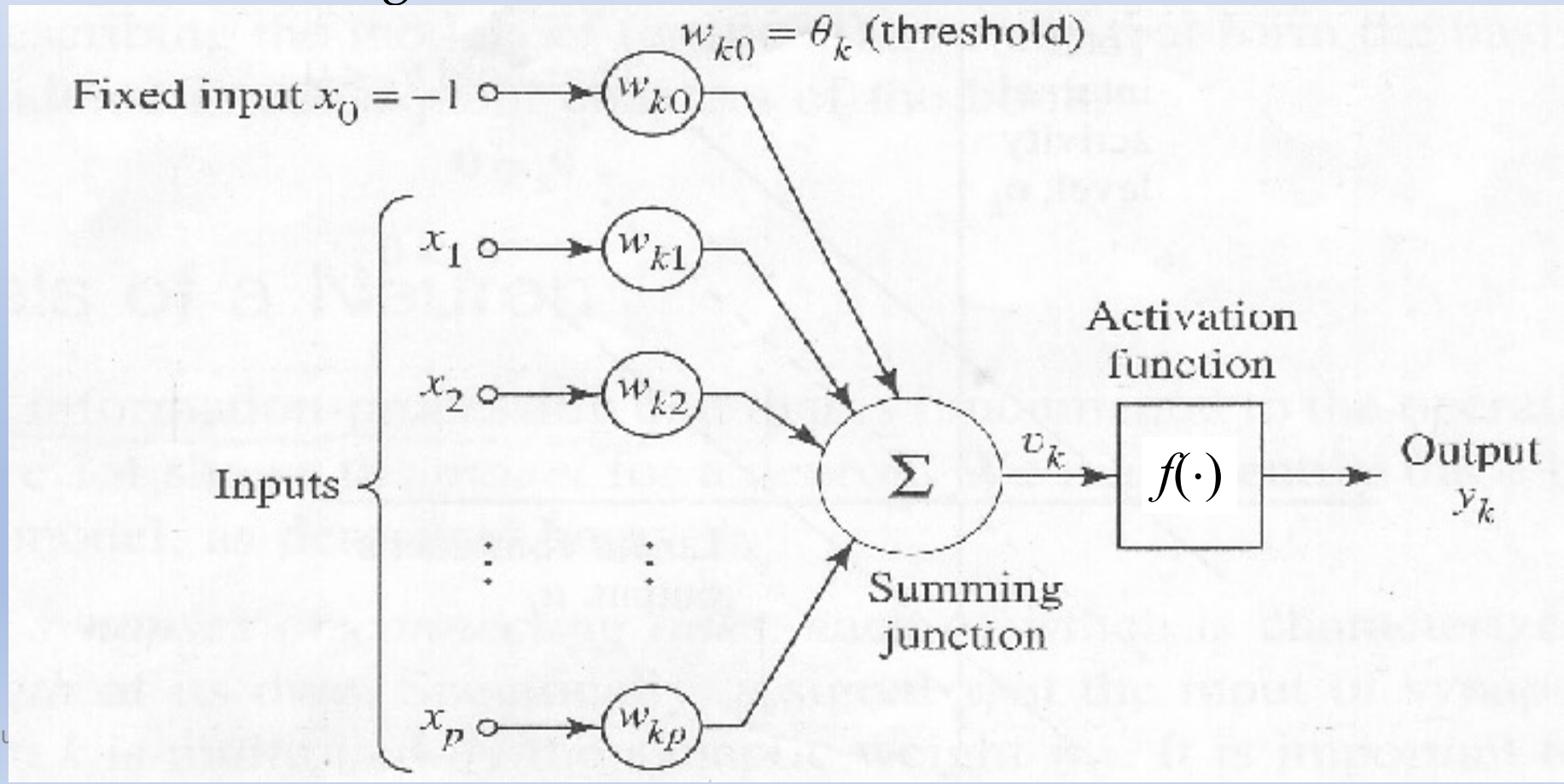
θ_k is an external parameter, we can consider this parameter as an input variable:

Then we have: $x_0 = 1, \quad w_{k0} = -\theta_k$

Neural Networks and Deep CNN -- Neuron Model

$$v_k = \sum_{j=0}^{p} w_{kj} x_j \qquad y_k = f(v_k)$$

Thus, the diagram of the model becomes:



Neural Networks and Deep CNN -- Activation Functions

Types of Activation Functions

Binary Function:

$$f(v)=\begin{cases}1 & v\geq 0\\ 0 & v<0\end{cases}$$

Piecewise-Linear Function

$$f(v)=\begin{cases}1 & v\geq 0.5\\ v & -0.5<v<0.5\\ 0 & v\leq -0.5\end{cases}$$

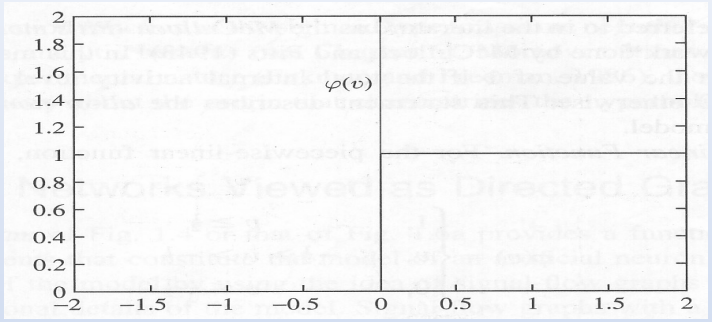
Sigmoid Function

$$f(v)=\frac{1}{1+\exp(-av)}$$

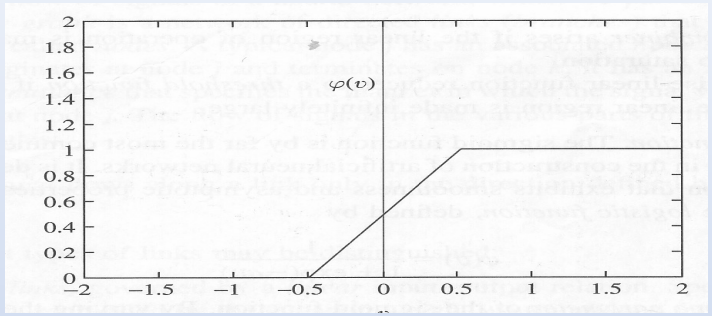
ReLU Function

$$f(v)=\max(0,v)=\begin{cases}v, & \text{for } v\geq 0\\ 0. & \text{for } v<0\end{cases}$$

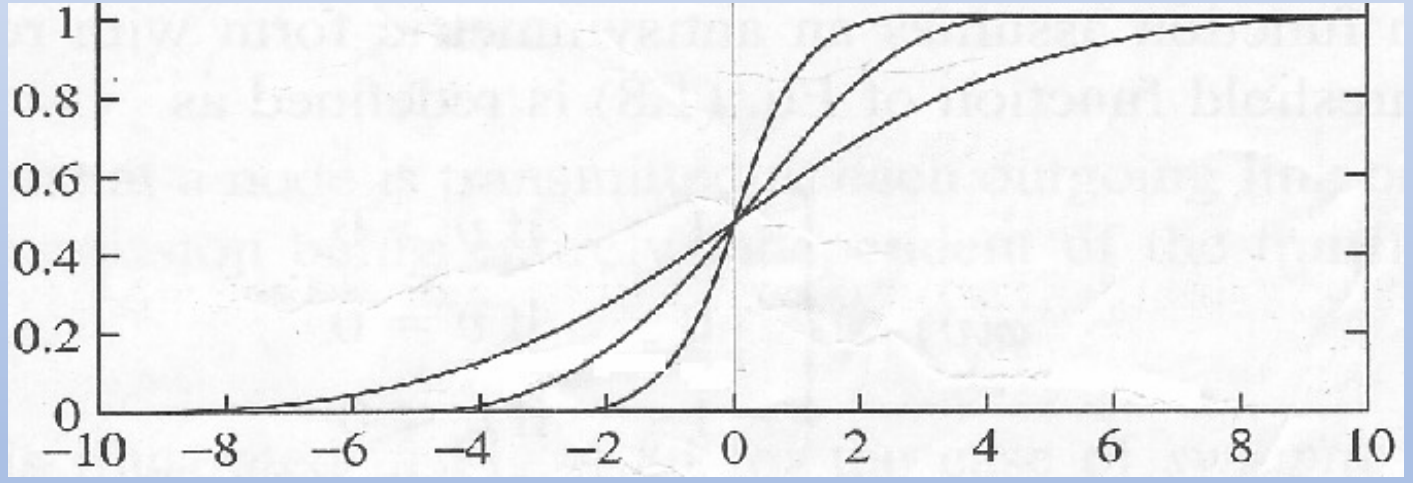
Neural Networks and Deep CNN -- Activation Functions



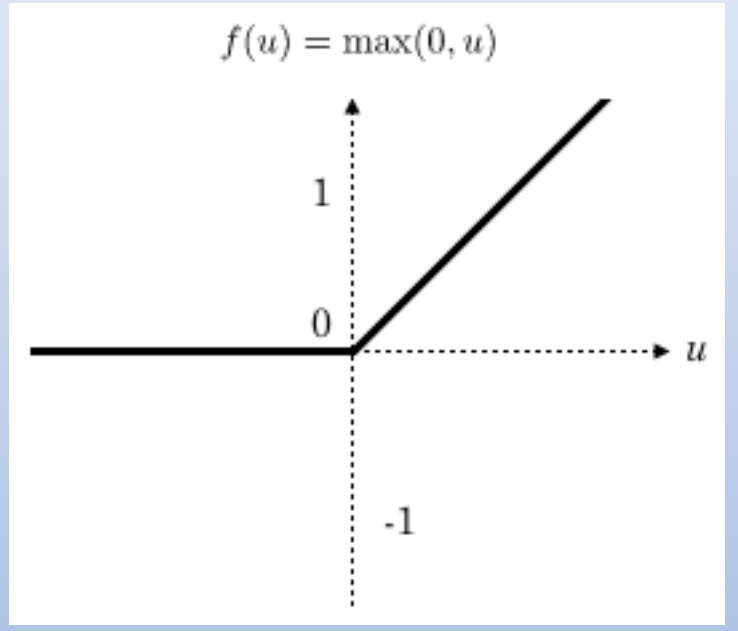
binary



Piecewise linear



sigmoid $$f(v) = \frac{1}{1+\exp(-av)}$$



ReLU

$$f(v) = \max(0, v) = \begin{cases} v, & \text{for } v \geq 0 \\ 0. & \text{for } v < 0 \end{cases}$$

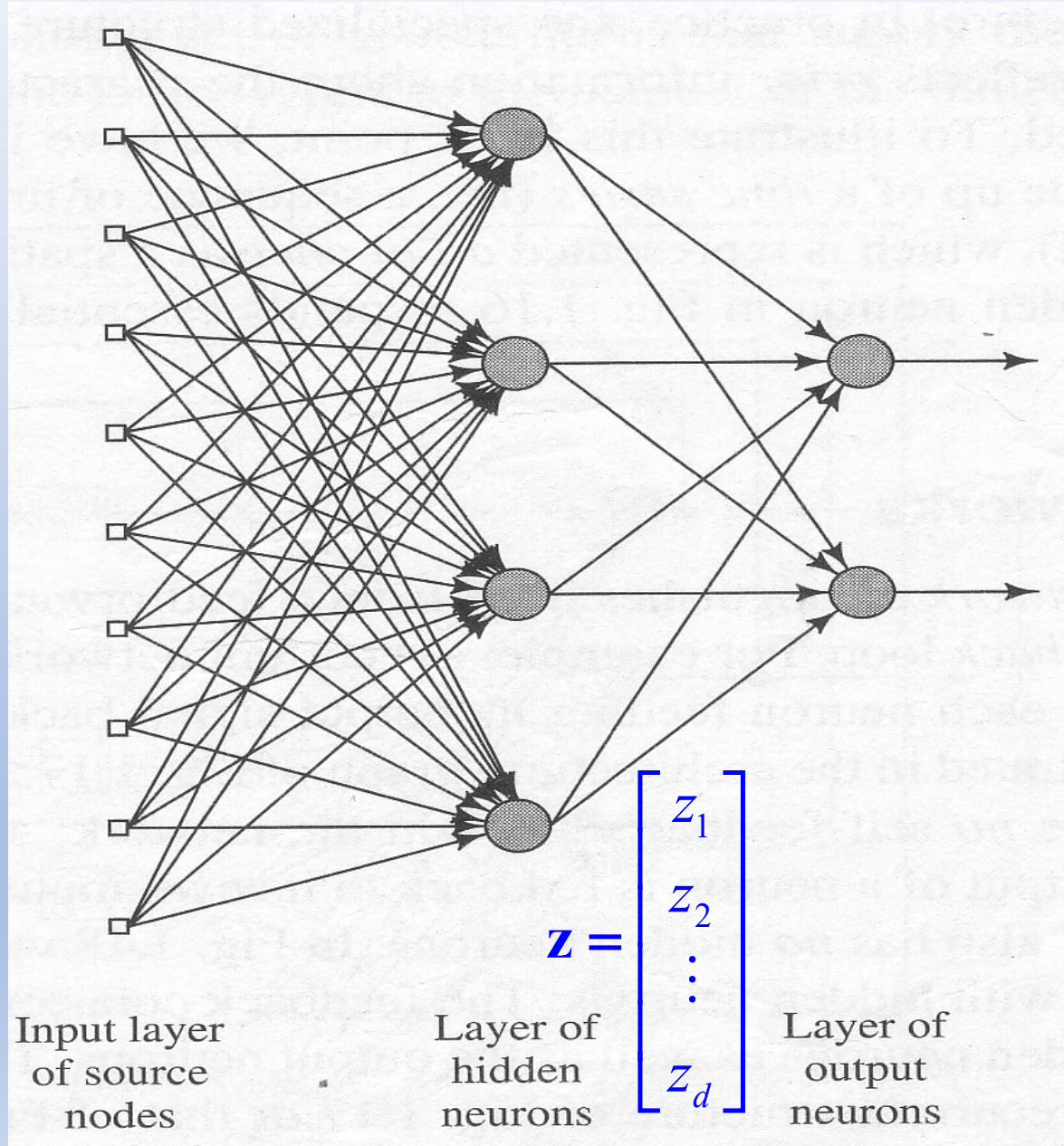
ANN – Multilayer Perceptron (MLP)

$$\mathbf{w}_j = \begin{bmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jn} \end{bmatrix}, \mathbf{v}_k = \begin{bmatrix} v_{k1} \\ v_{k2} \\ \vdots \\ v_{kd} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\mathbf{W} = [\mathbf{w}_1 \quad \mathbf{w}_2 \quad \cdots \quad \mathbf{w}_d]$$

$$\mathbf{V} = [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_c]$$



$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_c \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_c \end{bmatrix}$$

ANN – Multilayer Perceptron (MLP)

Mathematical expression of outputs from inputs
of neural networks in scalar, vector and matrix forms

$$z_j = f\left(\sum_{i=1}^{n} w_{ji} x_i\right) = f\left(\mathbf{w}_j{}^T \mathbf{x}\right)$$

$$y_k = f\left(\sum_{j=1}^{d} v_{kj} z_j\right) = f\left(\mathbf{v}_k{}^T \mathbf{z}\right) = f\left[\sum_{j=1}^{d} v_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i\right)\right]$$

$$\Downarrow$$

$$\mathbf{y} = f\left(\mathbf{V}^T \mathbf{z}\right) = f\left[\mathbf{V}^T f\left(\mathbf{W}^T \mathbf{x}\right)\right]$$

ANN – Multilayer Perceptron

- If the activation faction f is a linear function,

$$f(r) = r$$

$$\Downarrow$$

$$\mathbf{y} = f\left[\mathbf{V}^T f\left(\mathbf{W}^T \mathbf{x}\right)\right] = \mathbf{V}^T \mathbf{W}^T \mathbf{x} = (\mathbf{WV})^T \mathbf{x} = \mathbf{U}^T \mathbf{x}$$

$$\text{where } \mathbf{U} = \mathbf{WV}$$

- Multilayer network will be just the same as a single layer network.
- To design a multi-layer neural network, it is thus very important that the activation function should be a nonlinear function, at least for hidden neurons.

ANN – Single Layer Neural Network

- Let's first study the single layer network of single output

$$y = \mathbf{w}^T \mathbf{x}$$

- If we have a target output t, the error of the output is:

$$e = t - y = t - \mathbf{w}^T \mathbf{x}$$

- The mean square error is:

$$J(\mathbf{w}) = E\{e^2\} = E\{(t-y)^2\} = E\{(t-\mathbf{w}^T\mathbf{x})^2\}$$

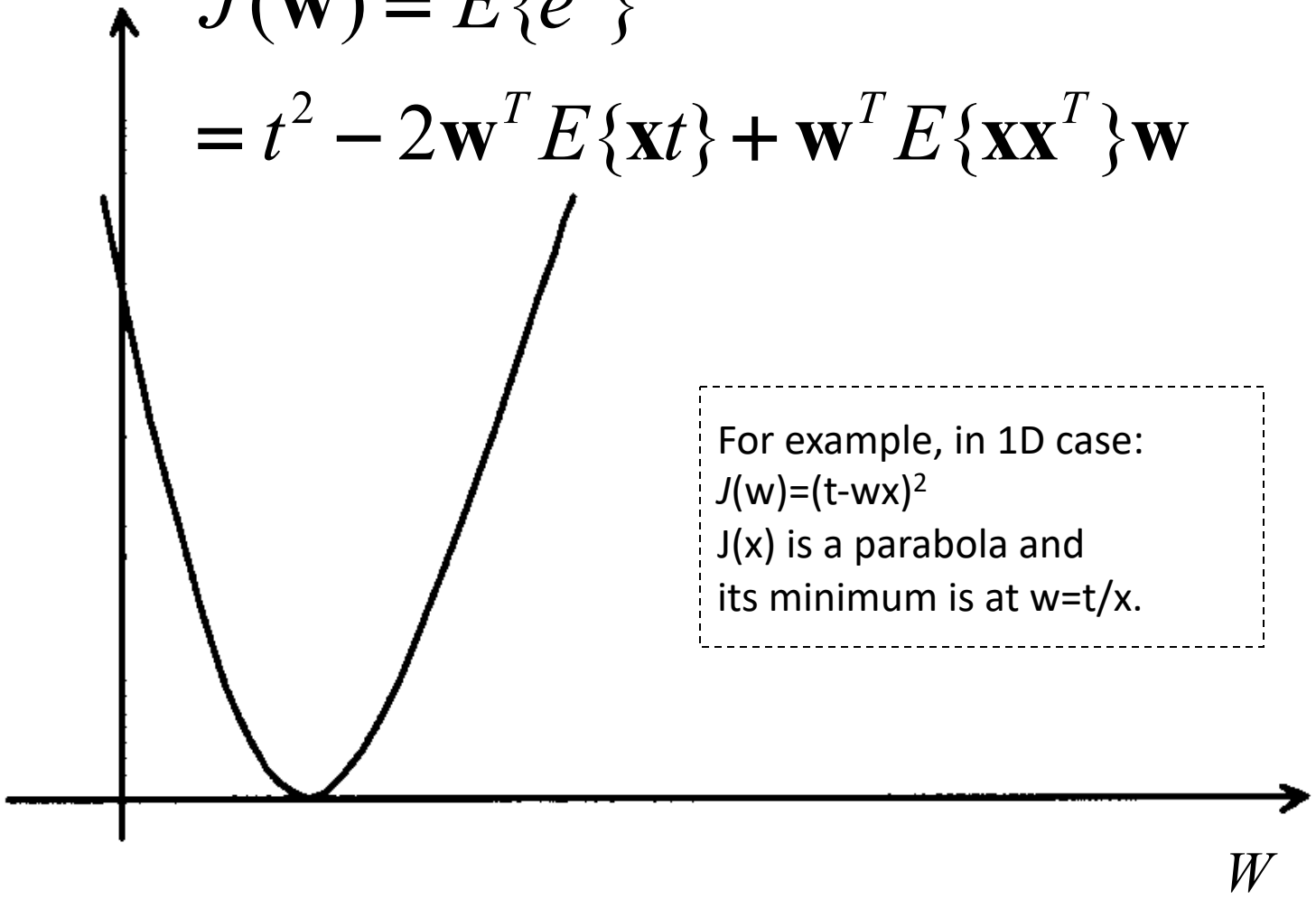
$$= t^2 - 2\mathbf{w}^T E\{\mathbf{x}t\} + \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w}$$

- Obviously, the error function is a quadratic function (second order) of $\mathbf{w}$. It has only one minimum point.

ANN – Single Layer Neural Network

$$J(\mathbf{w}) = E\{e^2\}$$

$$= t^2 - 2\mathbf{w}^T E\{\mathbf{x}t\} + \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w}$$



For example, in 1D case:
$J(w)=(t-wx)^2$
J(x) is a parabola and
its minimum is at w=t/x.

ANN – Single Layer Neural Network

- The optimal network parameter is obtained by minimizing the mean square error. This leads to the least mean square solution of $\mathbf{w}$:

$$\frac{\partial E\{e^2\}}{\partial \mathbf{w}} = \frac{\partial (t^2 - 2\mathbf{w}^T E\{\mathbf{x}t\} + \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w})}{\partial \mathbf{w}} = 0$$

$$\Downarrow$$

$$E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} = E\{\mathbf{x}t\}$$

$$E\{\mathbf{x}\mathbf{x}^T\} \cong \frac{1}{q}\sum_{i=1}^{q}\mathbf{x}_i\mathbf{x}_i^T, \quad E\{\mathbf{x}t\} \cong \frac{1}{q}\sum_{i=1}^{q}\mathbf{x}_i t_i$$

- Suppose we have q samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L$,

ANN – Single Layer Neural Network

- Let $\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_q], \quad \mathbf{t} = [t_1 \quad t_2 \quad \cdots \quad t_q]$
- Then: $q \cdot E\{\mathbf{x}\mathbf{x}^T\} \cong \sum_{i=1}^{q} \mathbf{x}_i \mathbf{x}_i^T = \mathbf{X}\mathbf{X}^T$

$$q \cdot E\{\mathbf{x}t\} \cong \sum_{i=1}^{q} \mathbf{x}_i t_i = \mathbf{X}\mathbf{t}^T$$

$$E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} = E\{\mathbf{x}t\}$$

$$\approx\Downarrow$$

$$\mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{t}^T \quad \Rightarrow \quad \mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{t}^T$$

$$y = \mathbf{w}^T\mathbf{x} = \mathbf{t}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{x}$$

- It is also called least square method.

ANN – Single Layer Neural Network

- We can also obtained the optimal weights W by iteratively adjust/update the value of W using gradient descent method:

$$\mathbf{w} \Leftarrow \mathbf{w} + \Delta\mathbf{w}$$

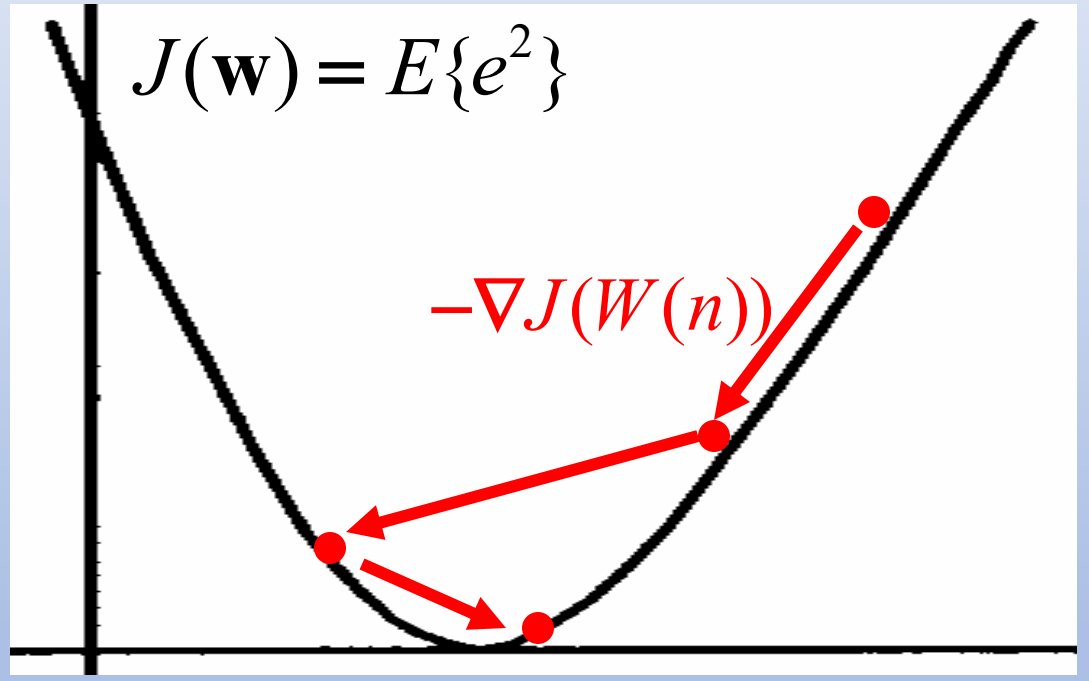
$$\Delta\mathbf{w} = -\eta\nabla J(\mathbf{w})$$

$$\nabla J(\mathbf{w}) = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$

$$= \frac{\partial E\{(t - \mathbf{w}^T\mathbf{x})^2\}}{\partial \mathbf{w}}$$

$$= -2E\{(t - \mathbf{w}^T\mathbf{x})\mathbf{x}\}$$

$$= -2E\{e\mathbf{x}\} \cong -2e_i\mathbf{x}_i$$

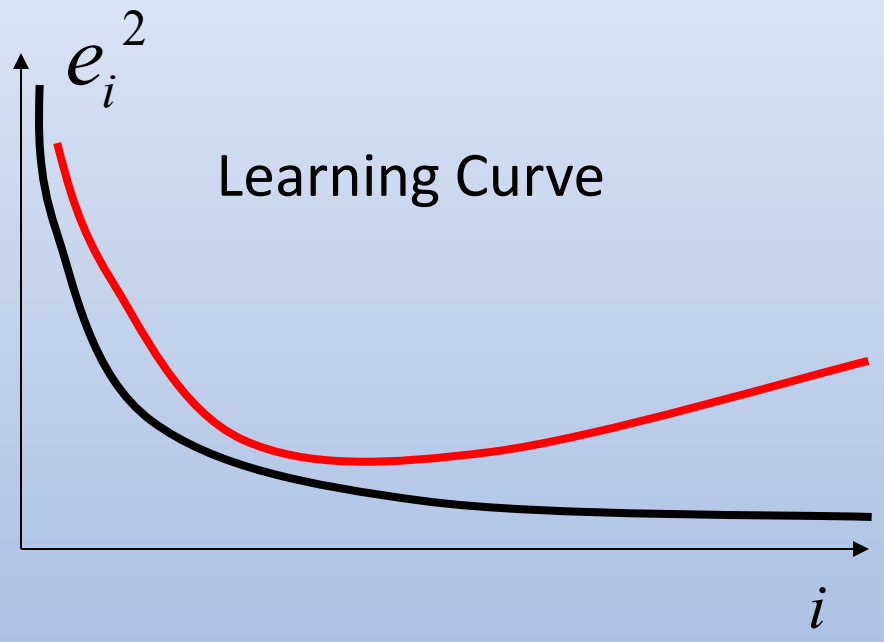


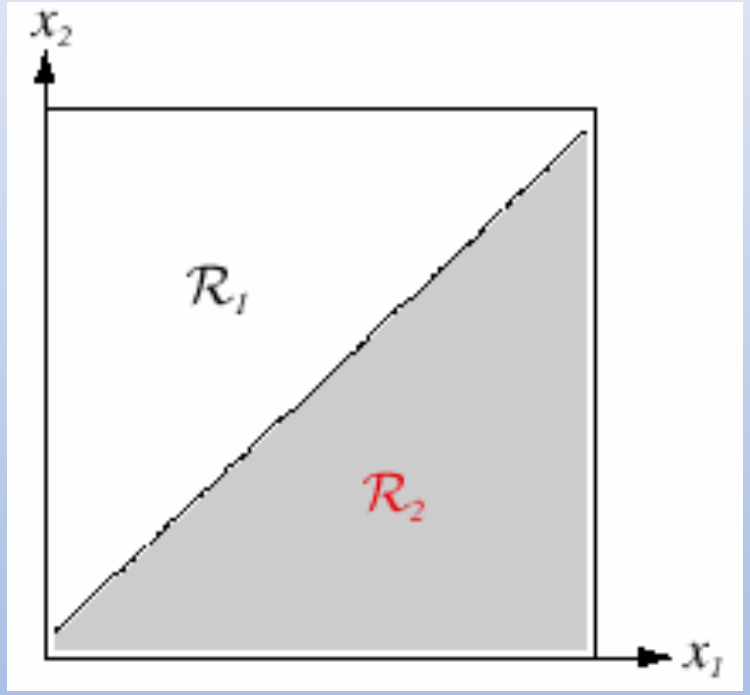
$$\mathbf{w} \Leftarrow \mathbf{w} + \eta e_i\mathbf{x}_i = \mathbf{w} + \eta(t_i - y_i)\mathbf{x}_i$$

$$\rightarrow \mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{t}^T$$

ANN – Learning Curve and Decision Boundary

$$\mathbf{w} \Leftarrow \mathbf{w} + \eta(t_i - y_i)\mathbf{x}_i$$





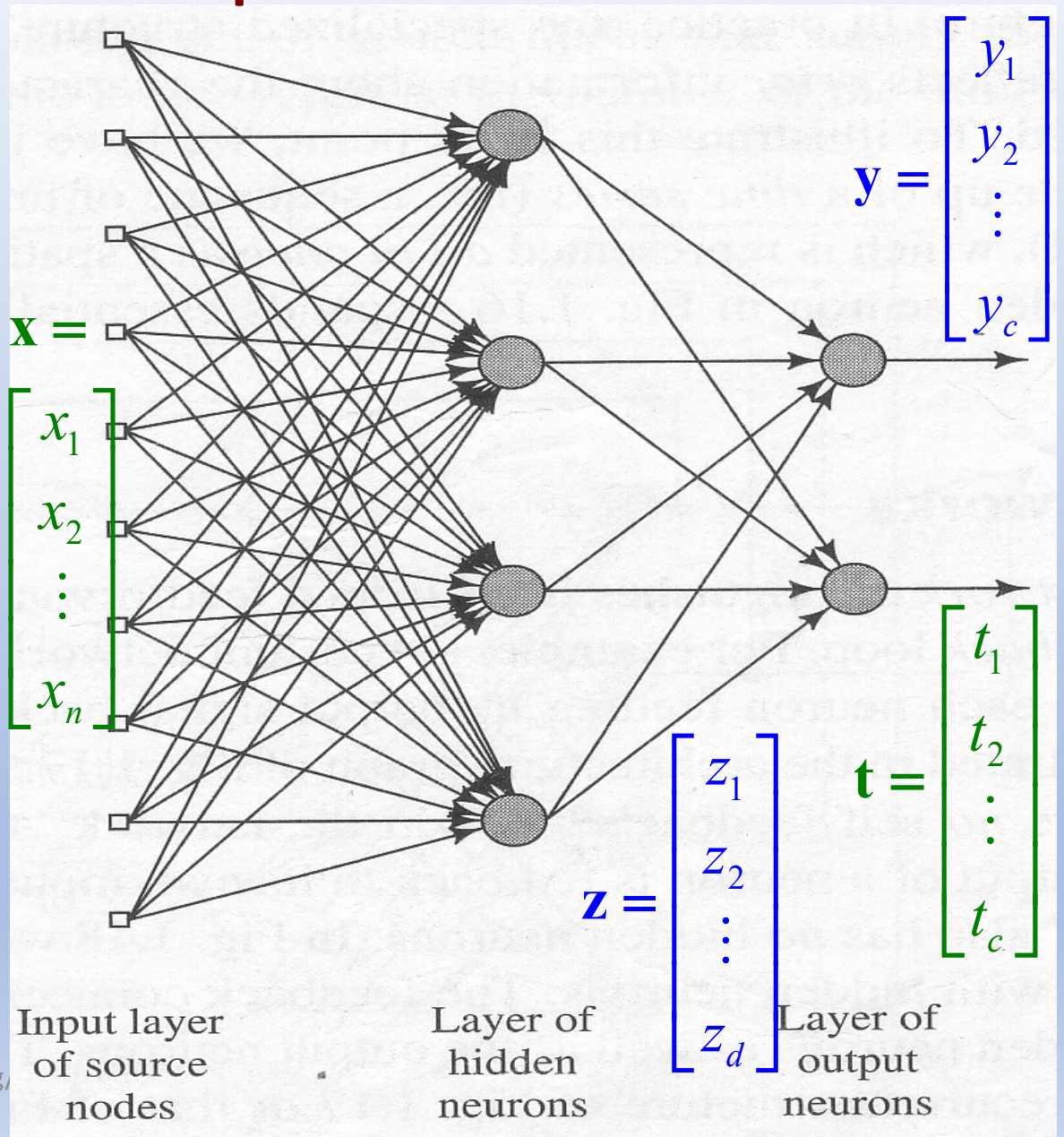
Decision boundary is a strait line or plane or hyper-plane.

$$\mathbf{w}^T\mathbf{x} = 0$$

ANN – Multilayer Perceptron

$$\mathbf{y} = f\left(\mathbf{V}^T\mathbf{z}\right) = f\left[\mathbf{V}^T f\left(\mathbf{W}^T\mathbf{x}\right)\right]$$

Any decision can be implemented by a two-layer network.
Any function from input to output can be implemented in a two-layer net, given sufficient number of hidden units, proper nonlinearities, and weights.



ANN – Multilayer Perceptron

- These results are of greater theoretical interest than practical, since the construction of such a network requires sufficient number of hidden units and the weight values are to be learnt!
- How to find the nonlinear function based on data is the central problem in network-based pattern recognition.
- Our goal now is to learn weights based on the training patterns and the desired outputs.
- The power of backpropagation is that it enables us to compute an effective error for each hidden unit, and thus derive a learning rule for the weights.

ANN – Multilayer Perceptron/backpropagation

$$\mathbf{y} = f\left(\mathbf{V}^T\mathbf{z}\right) = f\left[\mathbf{V}^T f\left(\mathbf{W}^T\mathbf{x}\right)\right]$$

$$J(\mathbf{w},\mathbf{v}) = E\{e^2\} = E\{\|\mathbf{t}-\mathbf{y}\|^2\} = E\left\{\left\|\mathbf{t} - f\left[\mathbf{V}^T f\left(\mathbf{W}^T\mathbf{x}\right)\right]\right\|^2\right\}$$

$$\text{To minimize } J(\mathbf{w},\mathbf{v}), \text{ let } \nabla J(\mathbf{w},\mathbf{v}) = 0$$

There is no analytical solution to this equation due to the nonlinear function f.

However, we can use gradient decent method so long as the gradient is computable for a given sample $\mathbf{x}_i$.

$$\mathbf{w} \Leftarrow \mathbf{w} - \eta\nabla J(\mathbf{w})$$

$$\mathbf{v} \Leftarrow \mathbf{v} - \eta\nabla J(\mathbf{v})$$

ANN – Multilayer Perceptron/backpropagation

- We use scalar form of weights to derive the learning formula.

$$J(\mathbf{w},\mathbf{v}) = E\{e^2\} = E\{\|\mathbf{t}-\mathbf{y}\|^2\} = \frac{1}{2}\sum_{k=1}^{c}(t_k - y_k)^2$$

$$= \frac{1}{2}\sum_{k=1}^{c}\left[t_k - f(q_k)\right]^2 = \frac{1}{2}\sum_{k=1}^{c}\left[t_k - f\left(\sum_{j=1}^{d} v_{kj} z_j\right)\right]^2$$

$$\frac{\partial J(\mathbf{w},\mathbf{v})}{\partial v_{kj}} = \frac{\partial J}{\partial q_k}\cdot\frac{\partial q_k}{\partial v_{kj}} = -(t_k - y_k) f'(q_k) z_j$$

ANN – Multilayer Perceptron/backpropagation

$$J(\mathbf{w},\mathbf{v}) = \frac{1}{2}\sum_{k=1}^{c}\left[t_k - f\left(\sum_{j=1}^{d} v_{kj} z_j\right)\right]^2 = \frac{1}{2}\sum_{k=1}^{c}\left[t_k - f\left[\sum_{j=1}^{d} v_{kj} f\left(q_j\right)\right]\right]^2$$

$$= \frac{1}{2}\sum_{k=1}^{c}\left[t_k - f\left[\sum_{j=1}^{d} v_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i\right)\right]\right]^2$$

$$\frac{\partial J(\mathbf{w},\mathbf{v})}{\partial w_{ji}} = \frac{\partial J}{\partial z_j}\cdot\frac{\partial z_j}{\partial q_j}\cdot\frac{\partial q_j}{\partial w_{ji}}$$

$$= -\left[\sum_{k=1}^{c}(t_k - y_k) f'(q_k) v_{kj}\right] f'(q_j) x_i$$

ANN – Multilayer Perceptron/backpropagation

$$v_{kj} \Leftarrow v_{kj} - \eta \frac{\partial J(\mathbf{w}, \mathbf{v})}{\partial v_{kj}}$$

$$w_{ji} \Leftarrow w_{ji} - \eta \frac{\partial J(\mathbf{w}, \mathbf{v})}{\partial w_{ji}}$$

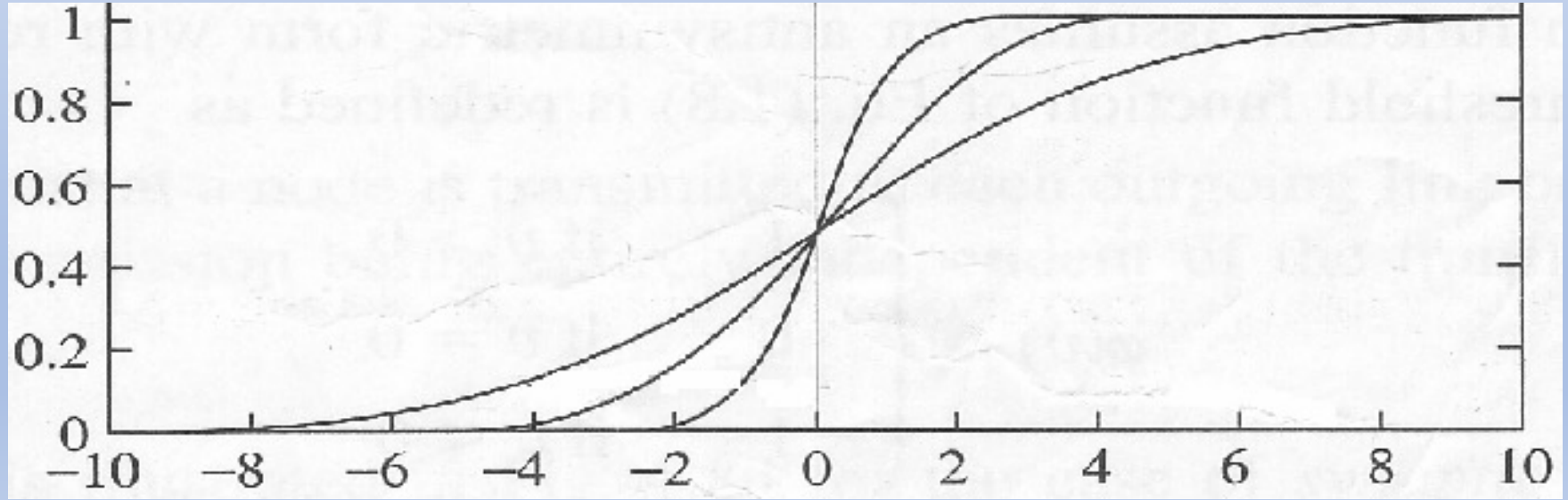
$$v_{kj} \Leftarrow v_{kj} + \eta (t_k - y_k) f'(q_k) z_j$$

$$w_{ji} \Leftarrow w_{ji} + \eta \left[\sum_{k=1}^{c} v_{kj} (t_k - y_k) f'(q_k) \right] f'(q_j) x_i$$

ANN – Multilayer Perceptron/backpropagation

$$f(q) = \frac{1}{1+\exp(-aq)}, \quad 0 \le f(q) \le 1$$

$$f'(q) = \frac{a\exp(-aq)}{[1+\exp(-aq)]^2} = af(q)[1-f(q)]$$



ANN – Multilayer Perceptron/backpropagation

- Such network is called multilayer perceptron (MLP) having two modes of operation:

  - Feedforward

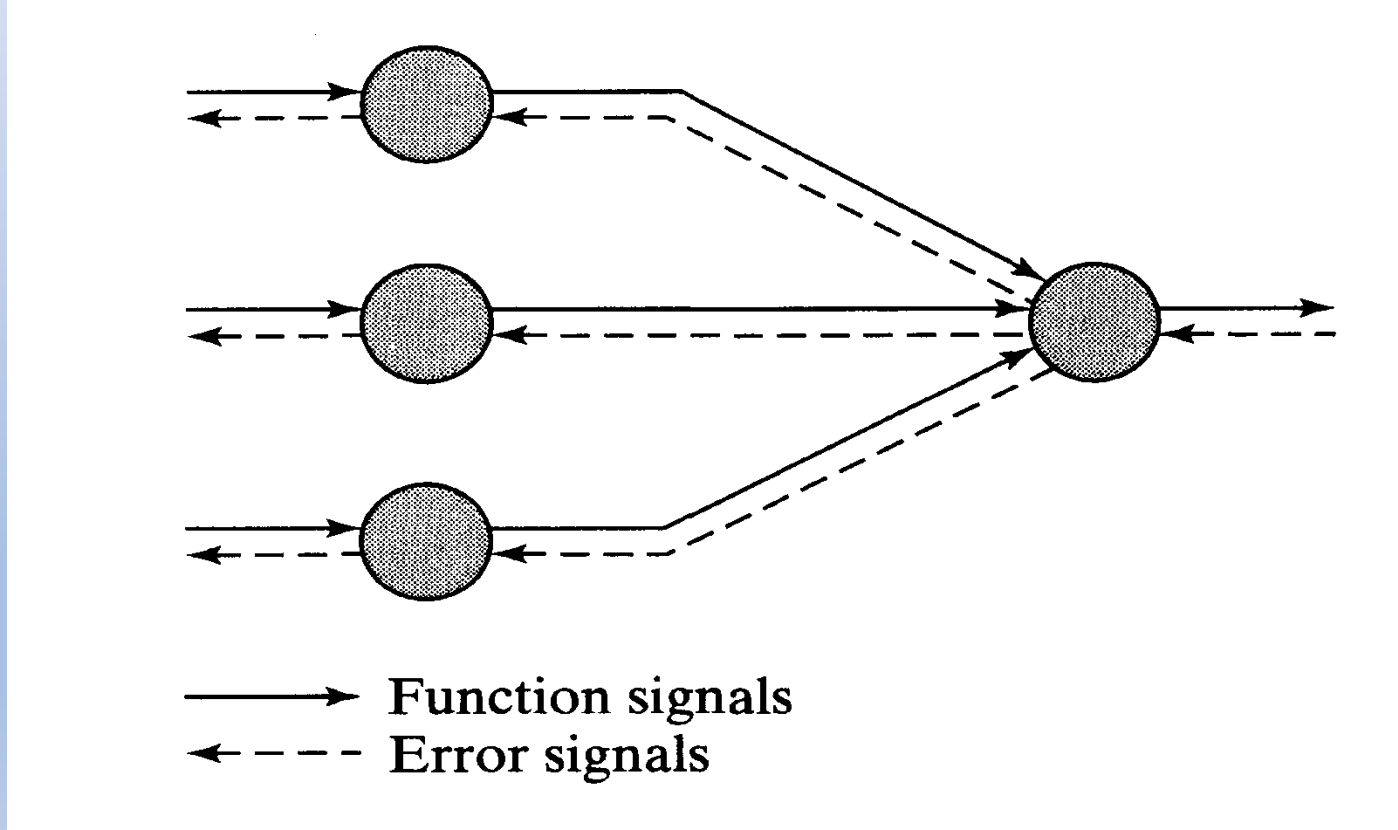
    The feedforward operations consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)

  - Learning

    The supervised learning consists of presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output

ANN – Multilayer Perceptron/backpropagation

- This learning algorithm is called backpropagation:
  The following figure shows a portion of the MLP. Two kinds of signals are identified in this network.



ANN – Multilayer Perceptron/backpropagation

Backpropagation training procedure:

Assume there are *n* the training samples:

$$\{\mathbf{x}(1), \mathbf{t}(1)\}, \{\mathbf{x}(2), \mathbf{t}(2)\}, \cdots, \{\mathbf{x}(n), \mathbf{t}(n)\}$$

(1) Initialization.

Assuming that no prior information is available, pick the synaptic weights from a uniform distribution whose mean is zero and whose variance is chosen to make the standard deviation of the activation signals lie at the transition between linear and saturated parts of the sigmoidal activation function.

(2) Presentation of training samples

Present the network with an epoch of training samples. For each sample in the set, perform the sequence of forward and backward computations.

ANN – Multilayer Perceptron/backpropagation

(3) Forward computation

Let a training samples in the epoch be denoted by $\{\mathbf{x}(i), \mathbf{t}(i)\}$, with the $\mathbf{x}(i)$ applied to the input layer and the desired response $\mathbf{t}(i)$ presented to the output layer. Compute the activation signals and function signals of the network by proceeding through the network, layer by layer.

(4) Backward computation

Compute the local gradient of the network, and adjust the synaptic weights of network.

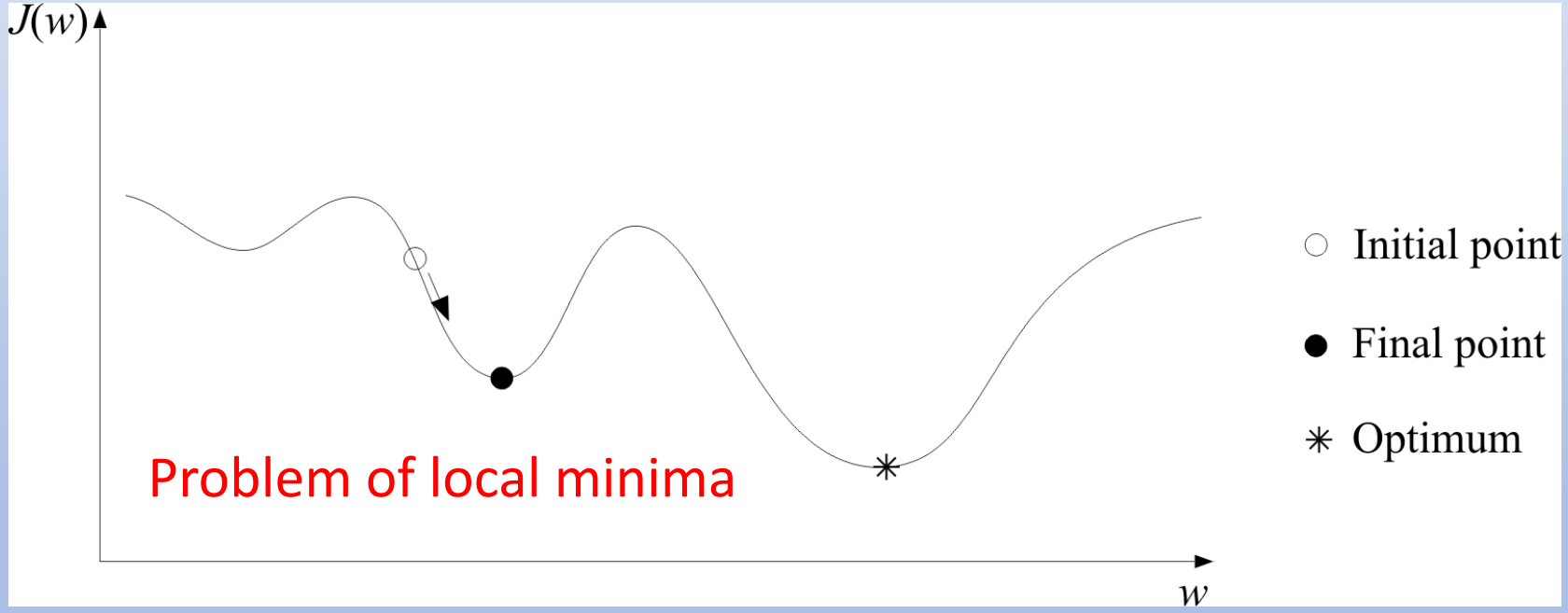
(5) Iteration

Iterate the forward and backward computations in steps (3)-(4) by representing new epochs of training samples to the network until the stopping criterion is met.

ANN – Problem of Local Minima

Note, the order of presentation of training samples should be randomized from epoch to epoch.

The stopping criterion can be the number of iterations, or the rate of change of the average error small enough.



X.D. Jiang and A. Kam, Constructing and Training Feed-Forward Neural Networks for Pattern Classification, *Pattern Recognition*, vol. 36, no. 4, pp. 853-867, April 2003.

ANN – Learning Curve

- Before training starts, the error on the training set is high; through the learning process, the error becomes smaller
- The error depends on the amount of training data and the expressive power (such as the number of weights) in the network
- The error on an independent test set is always higher than on the training set, and it can decrease as well as increase. (Over-fitting/generalization problem)
- A validation set is used to decide when to stop training ; we do not want to over fit the network and decrease the power of the classifier generalization

"We stop training at a minimum of the error on the validation set"

ANN – Learning Curve

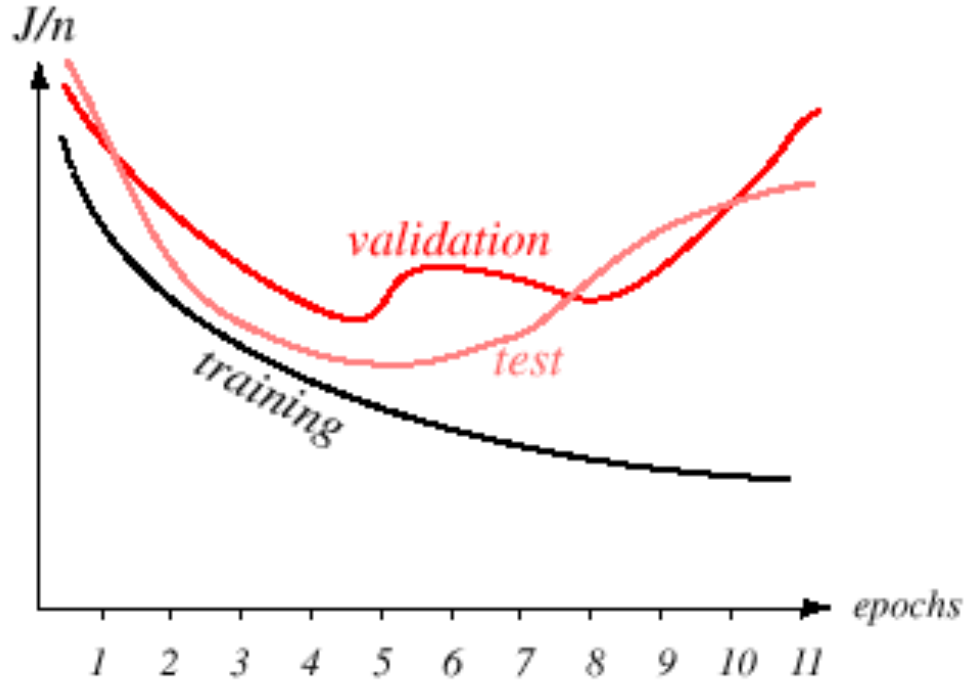


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

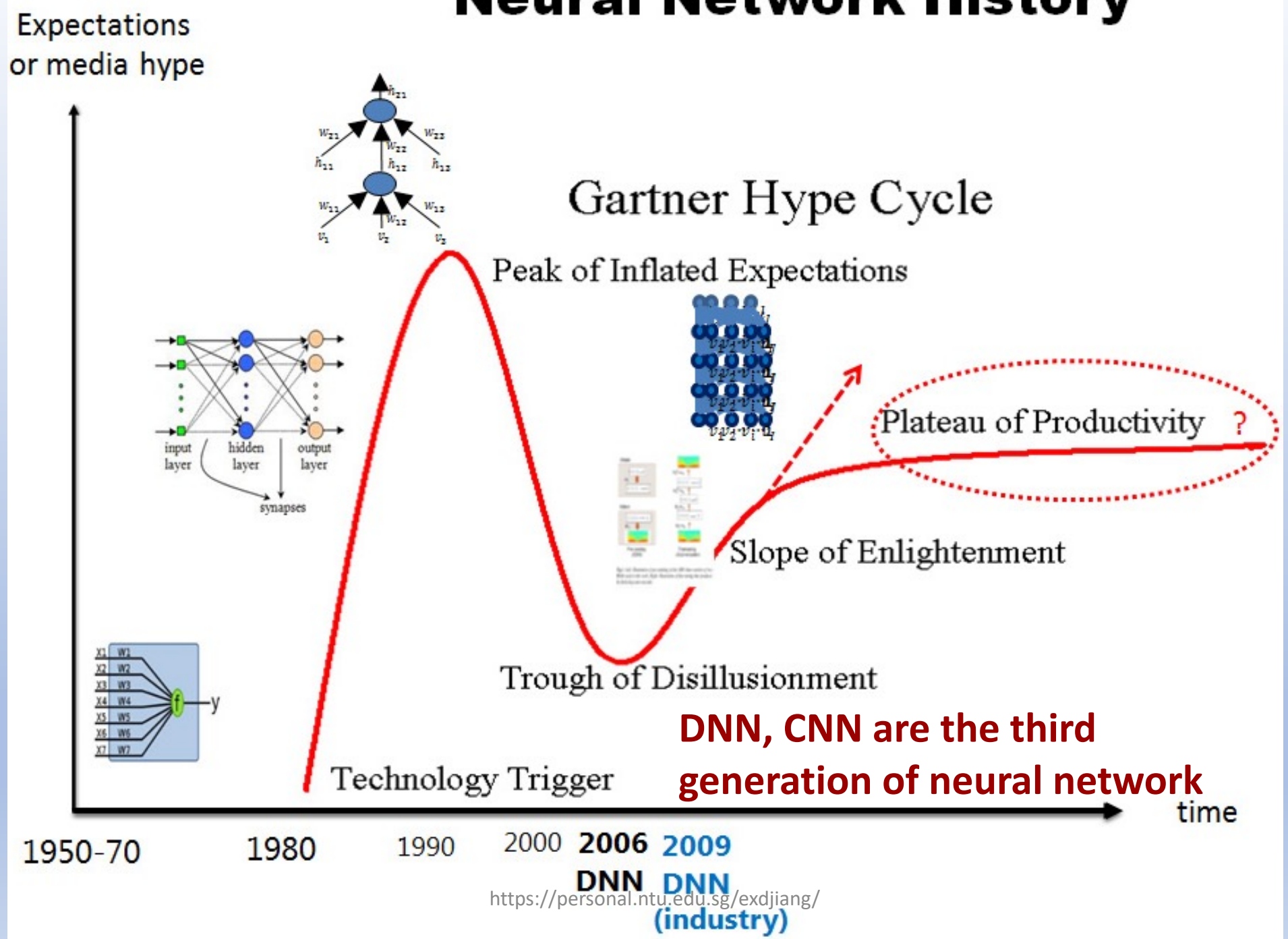
ANN – Conclusions of Neural Networks

- Neural networks are Universal Approximators

  It has been shown that any nonlinear continuous function can be approximated arbitrarily close, both, by a two-layer perceptron, with sigmoid activations, provided a large enough number of nodes are used.

  However, these results are of greater theoretical interest than practical, since the construction of such a network requires a large enough number of nodes and the weight values are to be learnt! It is still an open question how to find the nonlinear functions based on that training data. Problems of local minima (under-fitting the training data) and poor generalization (over-fitting the training data) are central issues of machine learning.

Neural Network History



DNN, CNN are the third generation of neural network

Supervised learning: find an unknown mapping or continuous function

$$f(\bullet):\ \mathbf{y} = f(\mathbf{x}),\quad \mathbf{x} \in \mathbf{R}^n,\ \ \mathbf{y} \in \mathbf{Z}^c$$

using finite discrete known training samples:

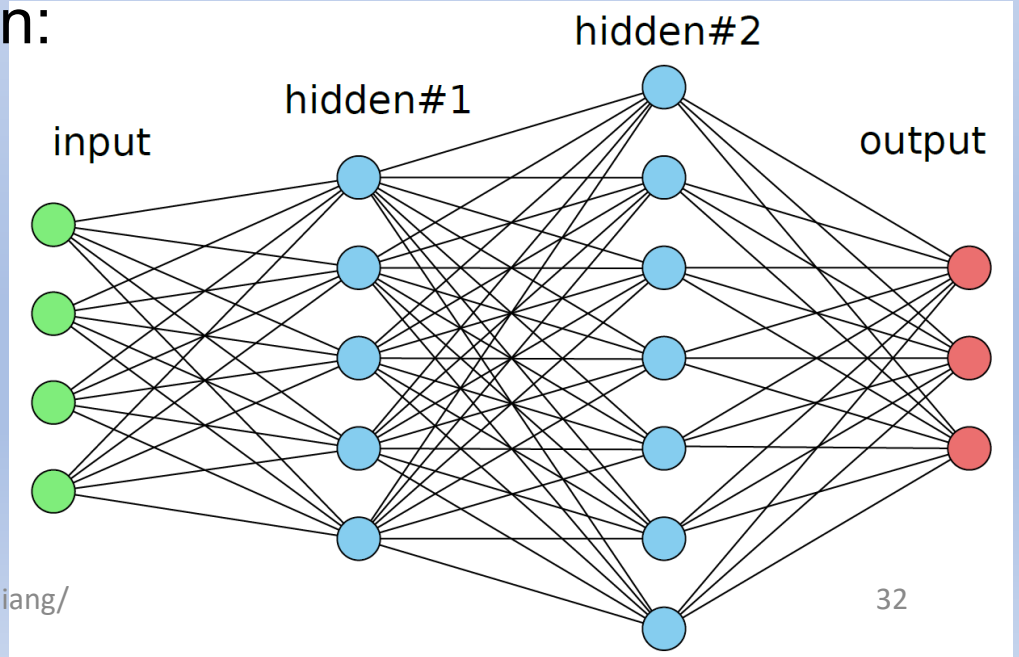
$$\{\mathbf{x}_i, \mathbf{y}_i\},\ \mathbf{y}_i = f(\mathbf{x}_i),\ \text{for } i=1, 2, ..., l.$$

This is to find: $\hat{f}(\bullet)$ by $\min_{\hat{f}} e^2 = \min_{\hat{f}} \sum_{\forall i} \left\| \mathbf{y}_i - \hat{f}(\mathbf{x}_i) \right\|_2^2$

- Neural network (NN or MLP) solution:

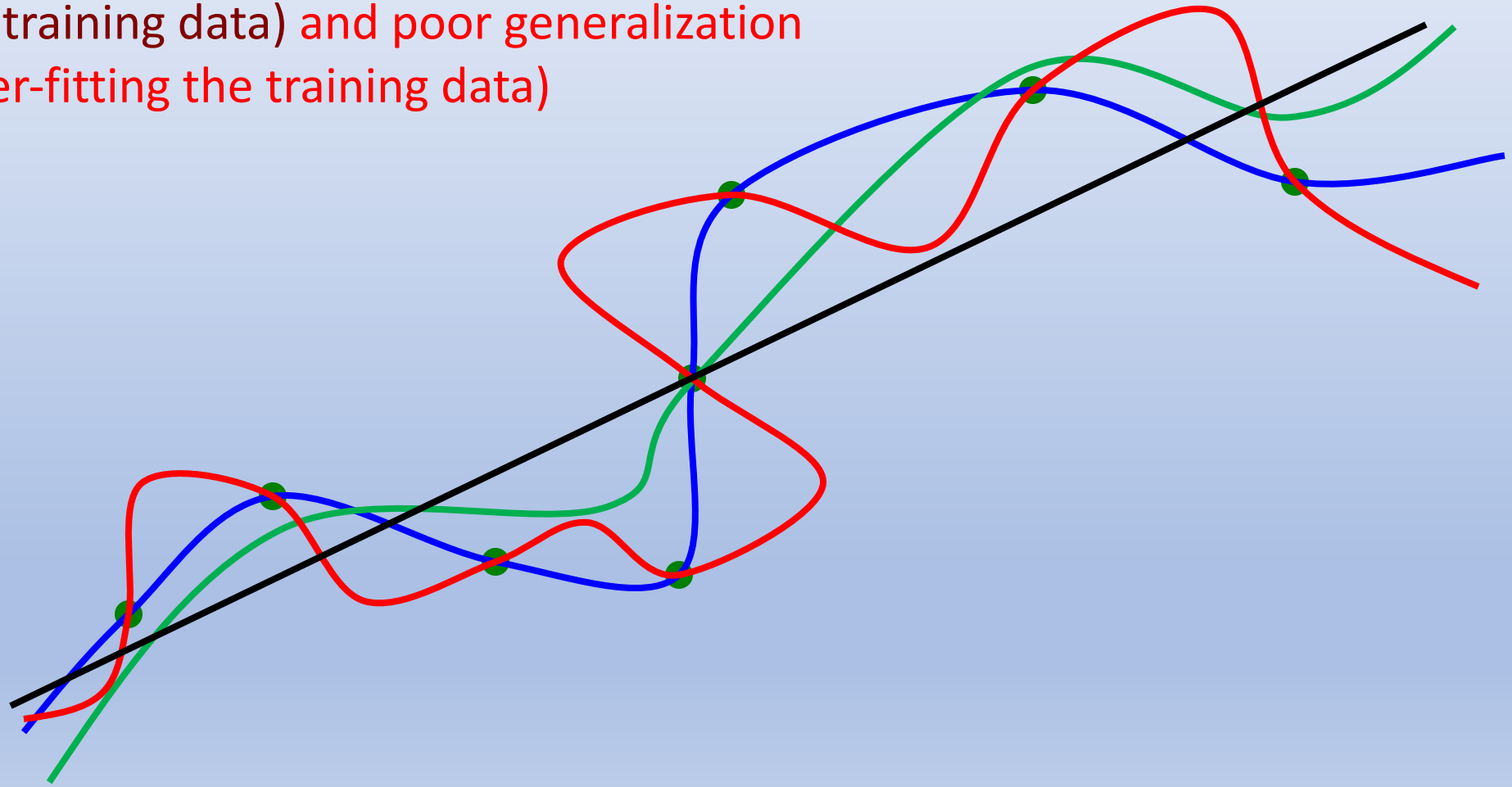
$$\mathbf{y} = h\left(\mathbf{W}_m^T \cdots h\left(\mathbf{W}_2^T h\left(\mathbf{W}_1^T \mathbf{x}\right)\right)\right)$$

$h(\bullet)$: simple nonlinear function



ANN – Understand Under- and Over-fitting

Problems of local minima (under-fitting the training data) and poor generalization (over-fitting the training data)



Problems of machine learning:

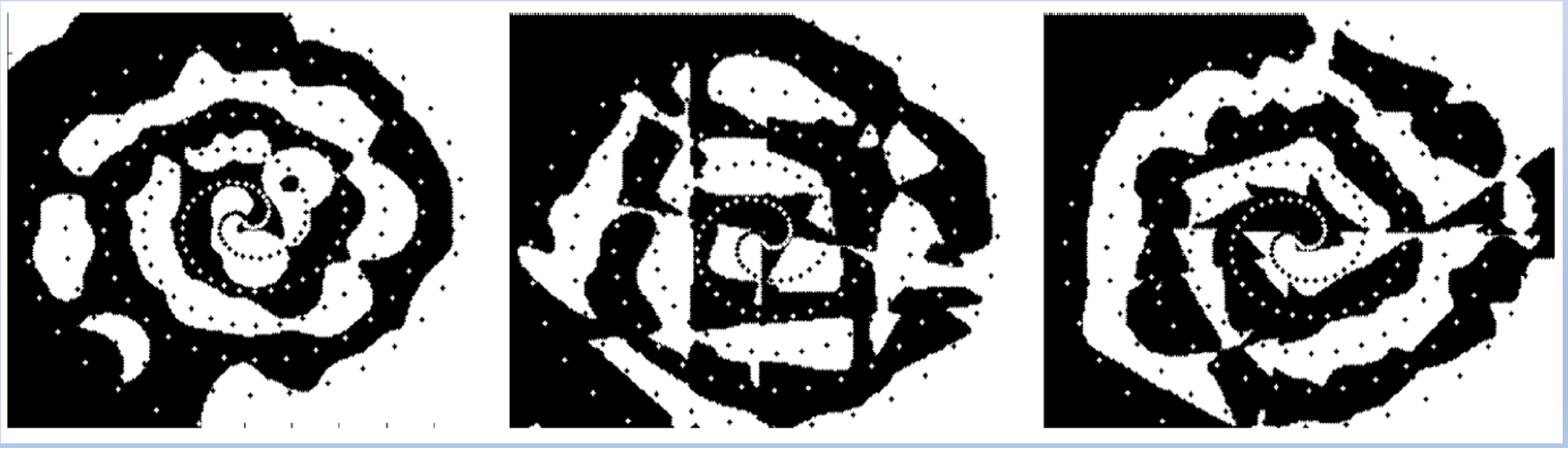
$$\min_{\hat{f}} e^2 = \min_{\hat{f}} \sum_{\forall i} \left\| \mathbf{y}_i - \hat{f}(\mathbf{x}_i) \right\|_2^2 \Rightarrow 0$$

Example of MLP to solve highly nonlinear double spiral problem

can only find: $\mathbf{y}_i = \hat{f}(\mathbf{x}_i)$ for i=1, 2, ..., l.

not $\mathbf{y} = f(\mathbf{x})$, for the wole population $\mathbf{x} \in \mathbf{R}^n$, $\mathbf{y} \in \mathbf{Z}^c$

Problem of overfitting or poor generalization is serious



X. Jiang, A. Wah, Constructing and Training Feed-Forward Neural Networks for Pattern Classification, ***Pattern Recognition***, vol. 36, pp. 853-867, Apr 2003.

- Problems of machine learning:

- How to make

$$\hat{f}(\mathbf{x}) \Rightarrow f(\mathbf{x}), \text{ for the wole population } \mathbf{x} \in \mathbf{R}^n, \ \mathbf{y} \in \mathbf{Z}^c \ ?$$

- Regularization! Using human knowledge to restrict or constrain $\hat{f}$,

$$\text{so that } \hat{f}(\mathbf{x}) \Rightarrow f(\mathbf{x}), \text{ for the whole population } \mathbf{x} \in \mathbf{R}^n, \ \mathbf{y} \in \mathbf{Z}^c$$

- Regularization! Using human knowledge to restrict or constrain $\hat{f}$,

How to regularize $\hat{f}$?

$$\min_{\hat{f}} e^2 = \min_{\hat{f}} \sum_{\forall i} \left\| \mathbf{y}_i - \hat{f}\left(\mathbf{x}_i\right) \right\|_2^2$$

$$\Downarrow$$

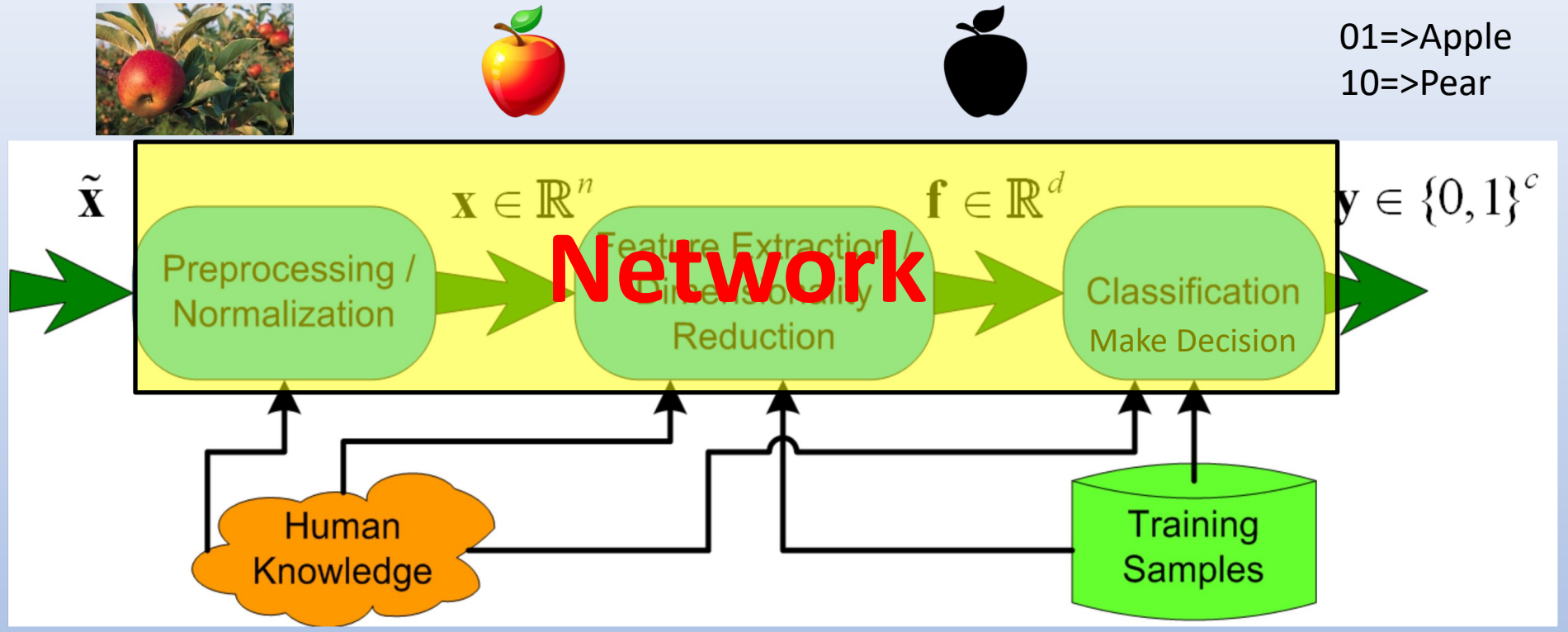
$$\min_{\hat{f}\in\Omega} e^2 = \min_{\hat{f}\in\Omega} \sum_{\forall i} \left\| \mathbf{y}_i - \hat{f}\left(\mathbf{x}_i\right) \right\|_2^2$$

$$\text{or } \min_{\hat{f}} \sum_{\forall i} \left\| \mathbf{W}\mathbf{y}_i - \hat{f}\left(\mathbf{W}\mathbf{x}_i\right) \right\|_2^2$$

$$\min_{\hat{f}} \left[\sum_{\forall i} \left\| \mathbf{y}_i - \hat{f}\left(\mathbf{x}_i\right) \right\|_2^2 + \lambda\Phi(\hat{f}) \right]$$

or all of the above

Functionalities of Image Recognition Modules

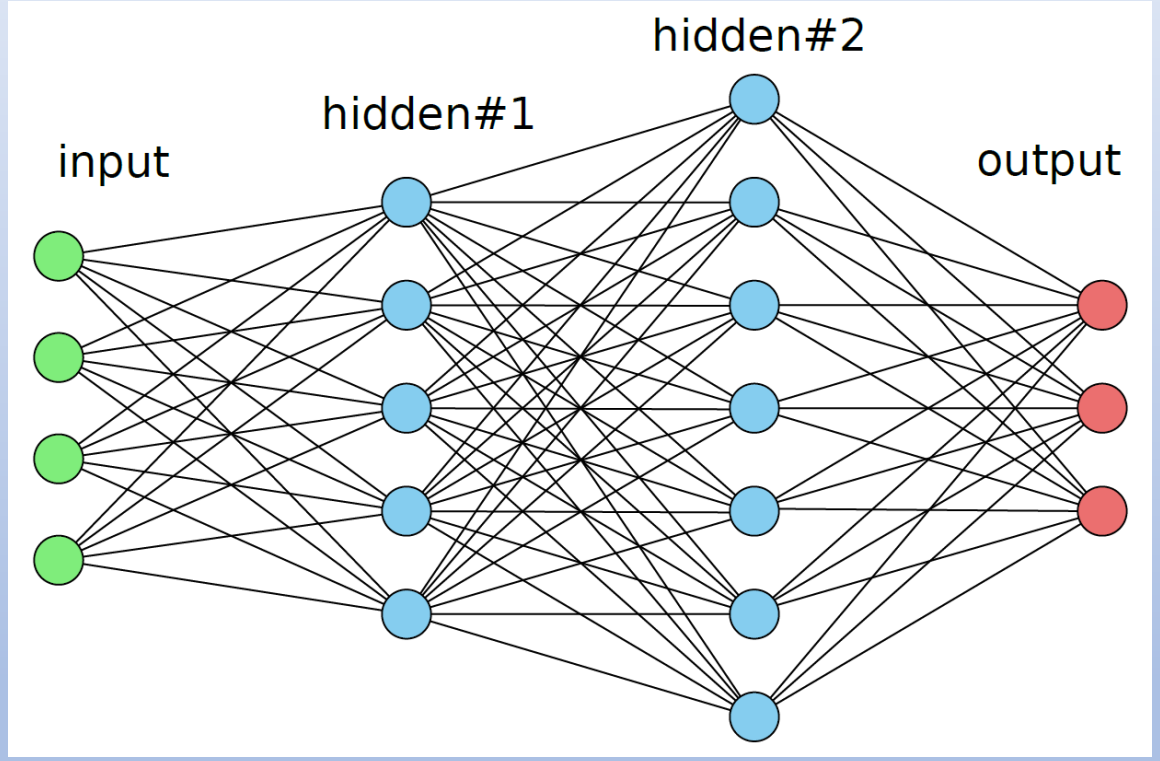


- Now the machine learning via CNN goes from classification to feature extraction/ dimensionality reduction and even goes to all other steps through DNN, CNN.
- Is human knowledge useless?

Understanding Deep Learning and Convolutional Neural Networks, CNN

$$\mathbf{y} = h\left(\mathbf{W}_m^T \cdots h\left(\mathbf{W}_2^T h\left(\mathbf{W}_1^T \mathbf{x}\right)\right)\right) \quad \Rightarrow \quad \mathbf{y} = f\left(\mathbf{x}\right)$$

- Theoretically, m=2 is sufficient
  But difficult to find solution.
- Deep Learning
  m>>2.
- Motivation:
  easier to find solution?

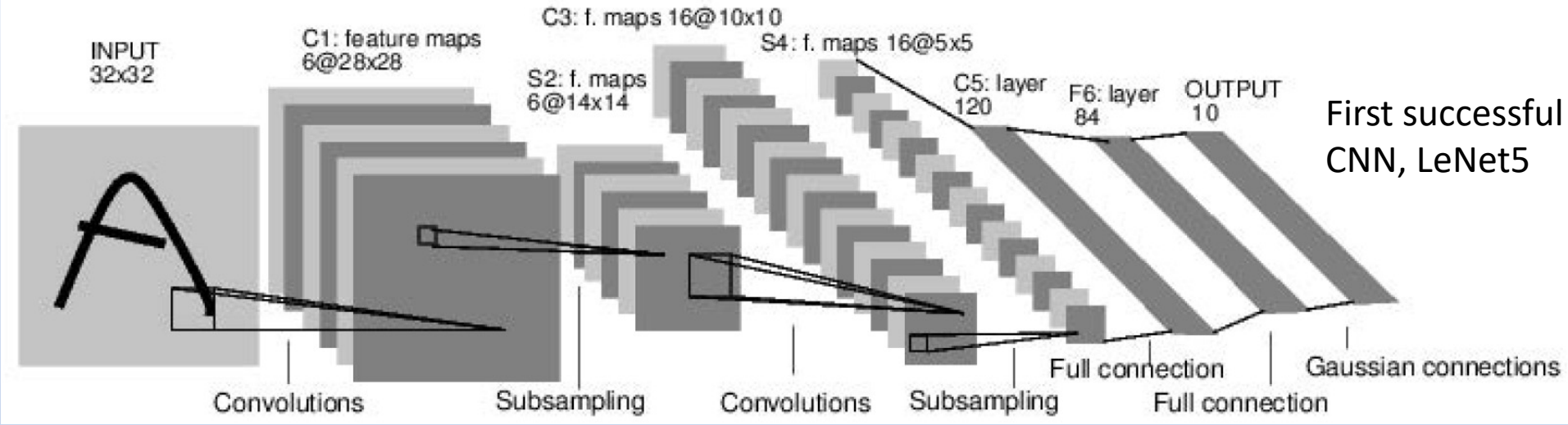


$$\mathbf{x}^{k+1} = h\left(\mathbf{W}_k^T \mathbf{x}^k\right), \quad k = 1,\ 2,\ \ldots,\ m$$

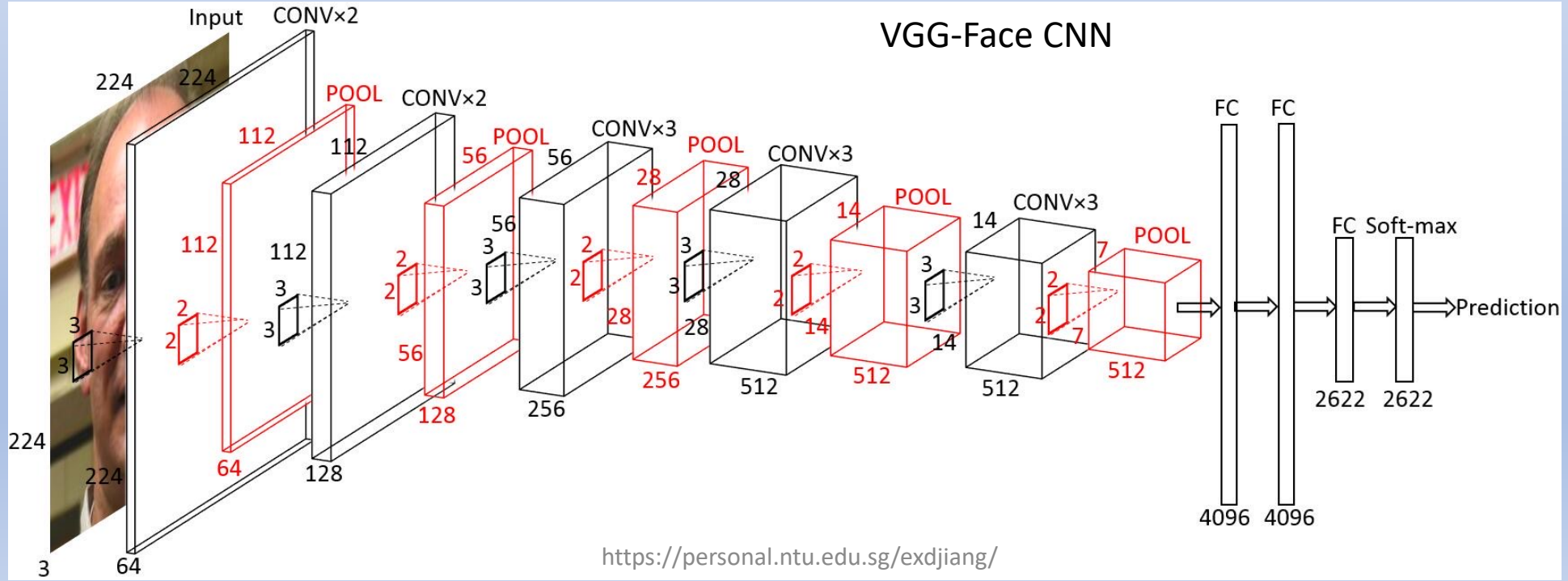
$$\mathbf{x}^1 = \mathbf{x}, \quad \mathbf{y} = \mathbf{x}^{m+1}, \quad \mathbf{o}^k = \mathbf{W}_k^T \mathbf{x}^k \Rightarrow \mathbf{o} = \mathbf{W}^T \mathbf{x}$$

$$\mathbf{x}^k \in \mathbf{R}^{n_k},\ \mathbf{o}^k, \mathbf{x}^{k+1} \in \mathbf{R}^{n_{k+1}},\ \mathbf{W}_k \in \mathbf{R}^{n_k \times n_{k+1}}$$

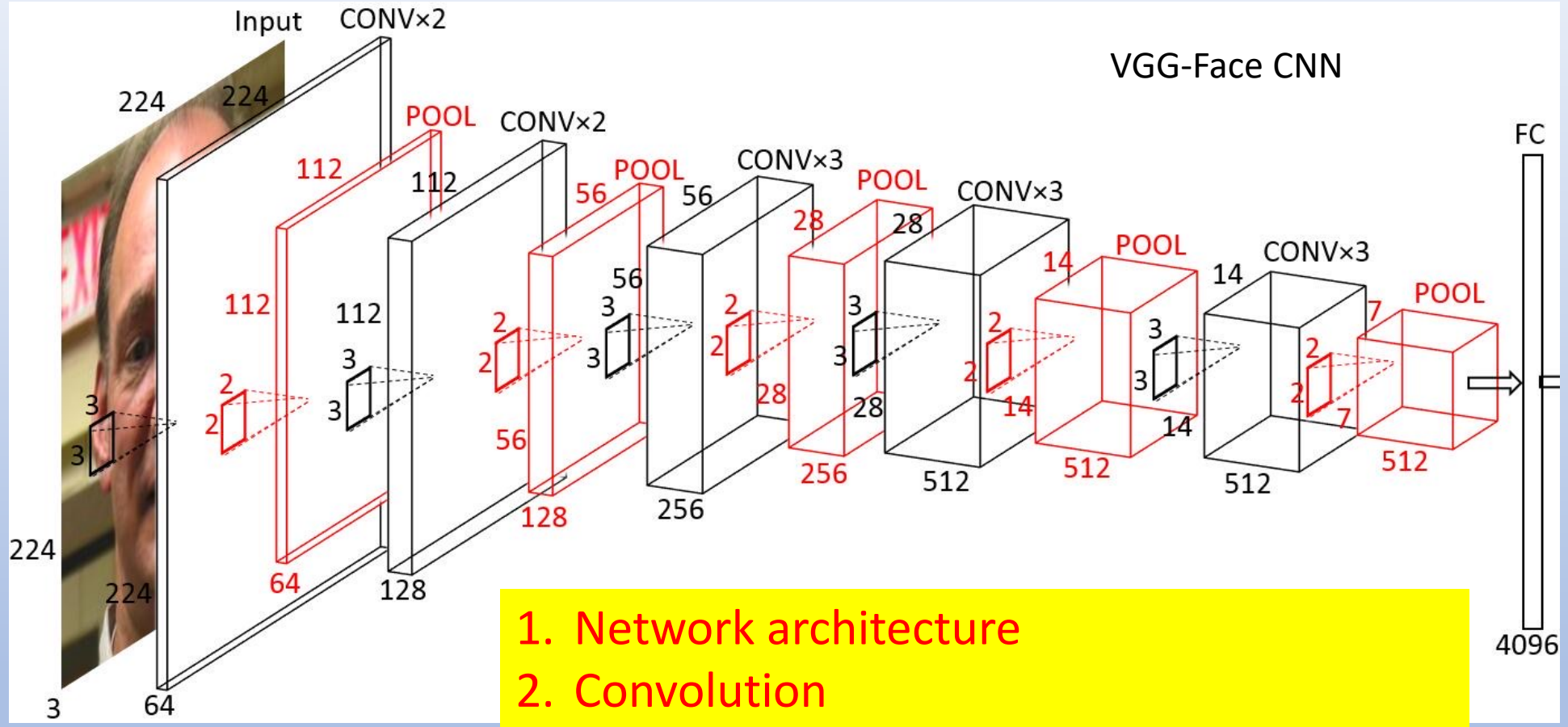
Convolutional network CNN appears to be quite different from MLP?



First successful CNN, LeNet5



Characteristics of Convolutional Network CNN

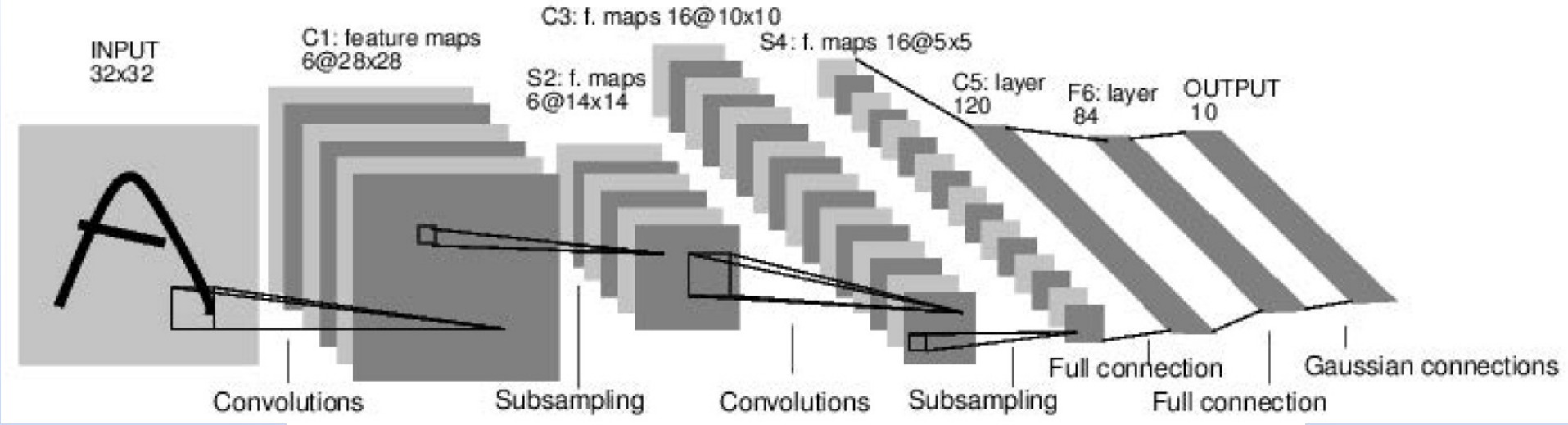


1. Network architecture
2. Convolution
3. Simple nonlinear activation function ReLu
4. Pooling
5. Deep, large number of layers

Compare MLP and CNN: Network architecture

$$n_k = 32 \times 32 = 1024$$

$$n_{k+1} = 6 \times 28 \times 28 = 4704$$



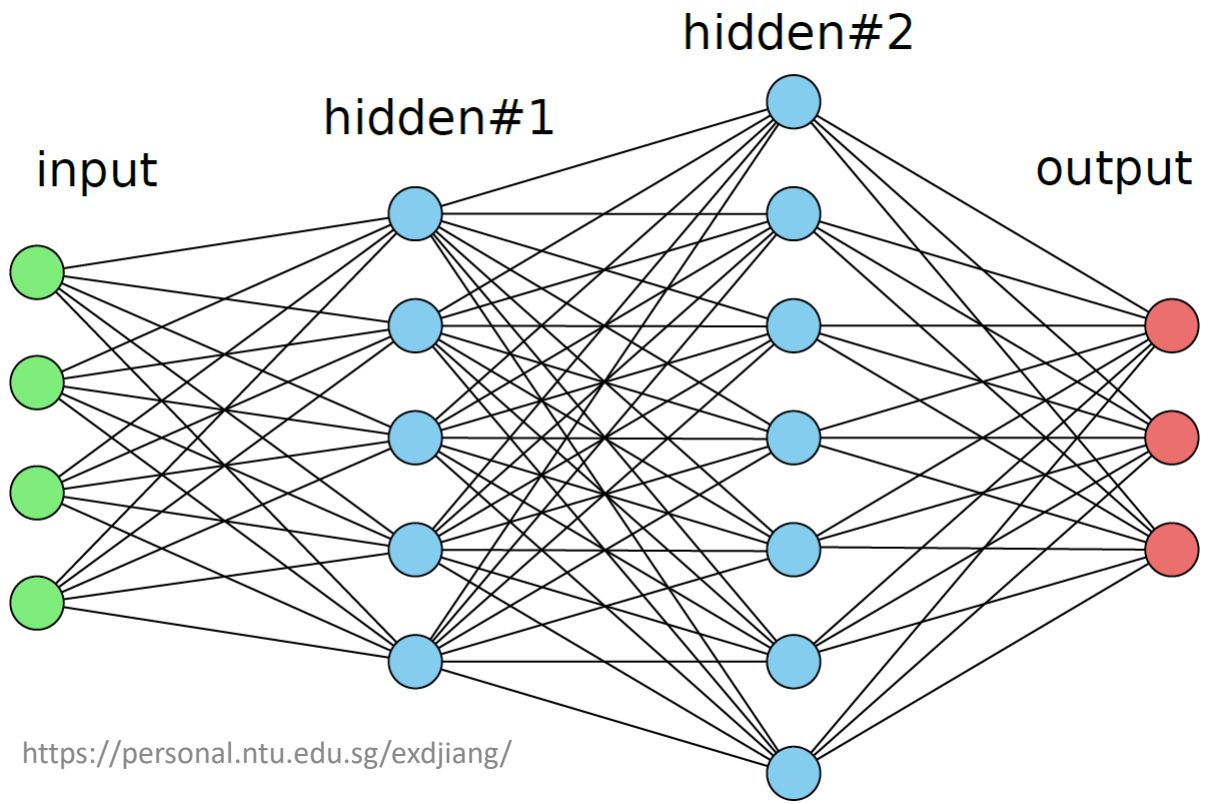
To compare CNN and MLP, we convert or only see the 1-D convolution.

$$\mathbf{x} = \{x_j\} \in \mathbf{R}^{n_k}$$

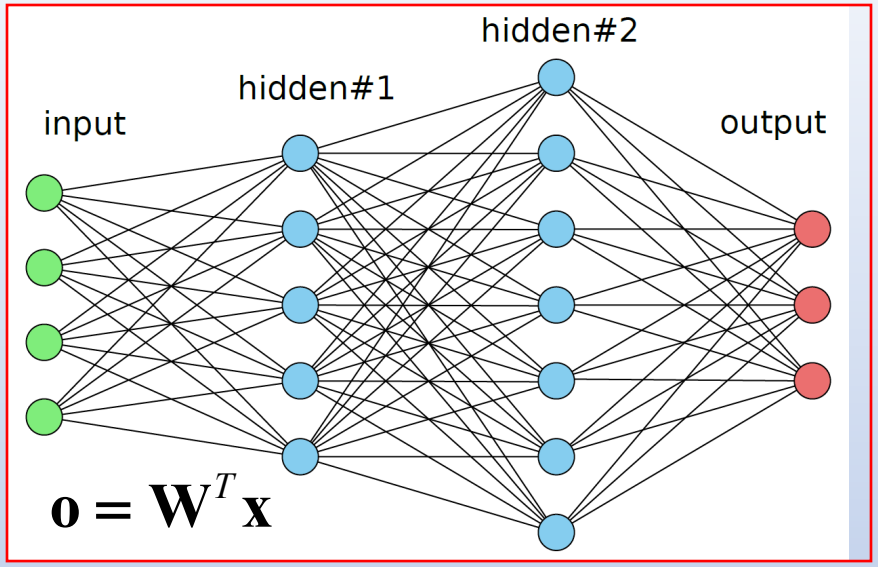
$$\mathbf{o} = \{o_j\} \in \mathbf{R}^{n_{k+1}},$$

$$\mathbf{W} \in \mathbf{R}^{n_k \times n_{k+1}}$$

$$\mathbf{o} = \mathbf{W}^T \mathbf{x} ?$$



Compare MLP and CNN: 1-D Convolution



$$\mathbf{w}_j^q = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{g}^q \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \mathbf{W}^q = \begin{pmatrix} \mathbf{g}^q & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & \ldots \mathbf{g}^q \ldots & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & \mathbf{g}^q \end{pmatrix}$$

$$\mathbf{o}^q = \mathbf{W}^{qT} \mathbf{x}$$

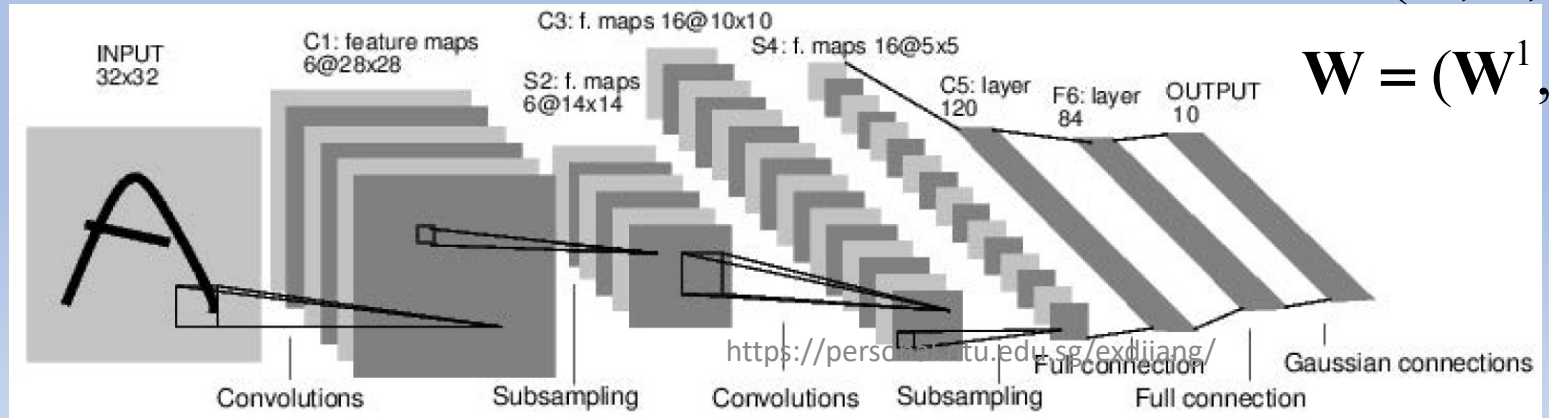
$$o_j^q = g_j^q * x_j, \quad g_j^q = 0, \quad j < -a, j > a$$

$$= \sum_{i=1}^{n_k} g_{j-i}^q x_i = \mathbf{w}_j^{qT} \mathbf{x}$$

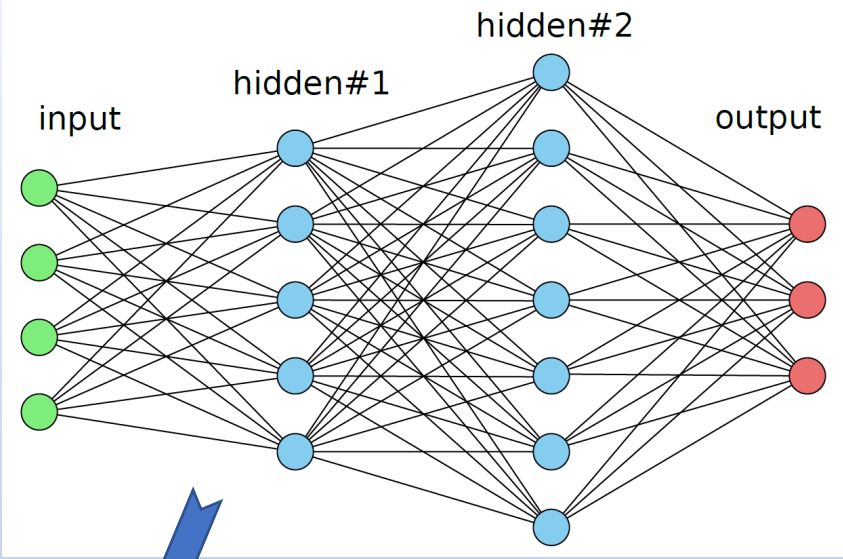
$$\mathbf{o} = (\mathbf{o}^1, \ldots, \mathbf{o}^q \ldots, \mathbf{o}^p)^T$$

$$\mathbf{W} = (\mathbf{W}^1, \ldots, \mathbf{W}^q \ldots, \mathbf{W}^p)$$

$$\mathbf{o} = \mathbf{W}^T \mathbf{x}$$



Compare MLP and CNN:



$$\mathbf{o} = \mathbf{W}^T \mathbf{x}$$

$$\mathbf{W} = (\mathbf{W}^1, ..., \mathbf{W}^q ..., \mathbf{W}^p)$$

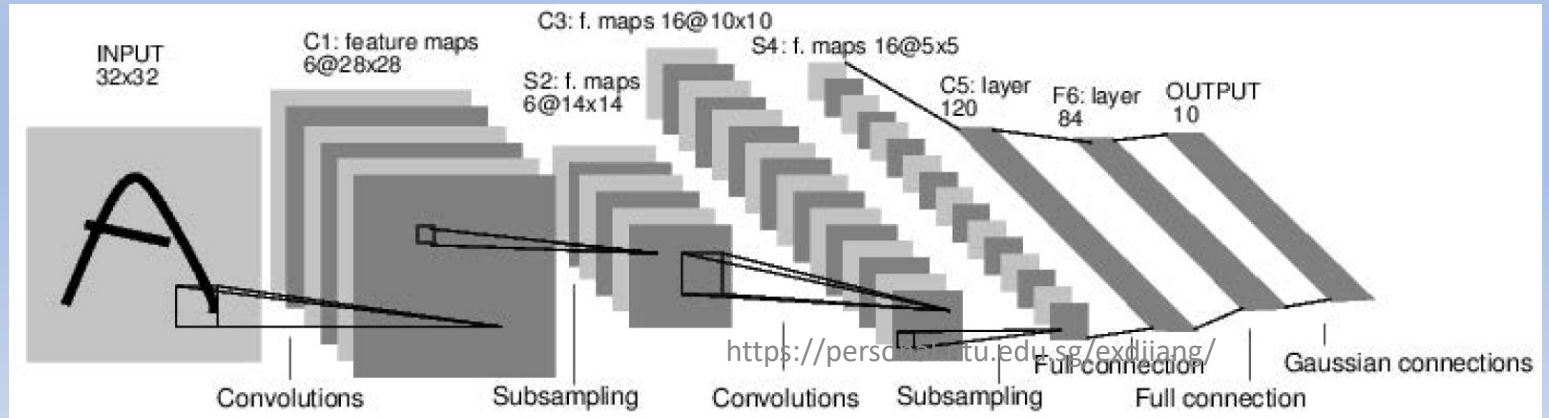
$$= \begin{pmatrix} \mathbf{g}^1 & 0 & 0 & \mathbf{g}^p & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ...\mathbf{g}^1 ... & 0 & ..... & 0 & ...\mathbf{g}^p ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \mathbf{g}^1 & 0 & 0 & \mathbf{g}^p \end{pmatrix}$$

CNN is a simplified MLP

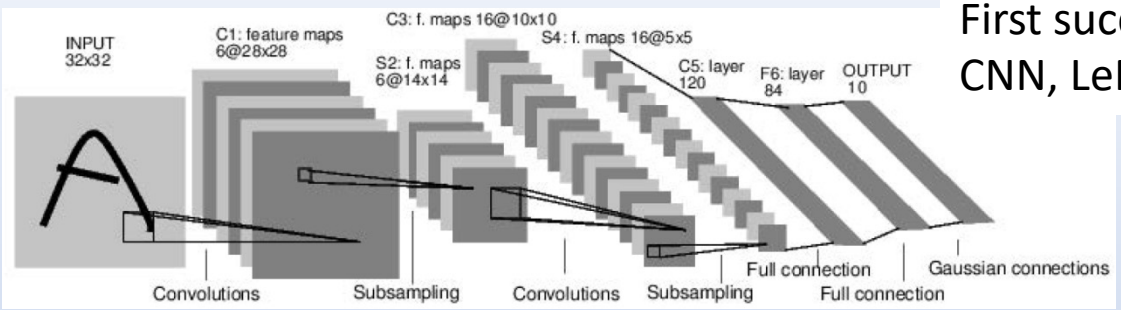
CNN is a regularized MLP

Simplification/regularization may Solve over-fitting problem.

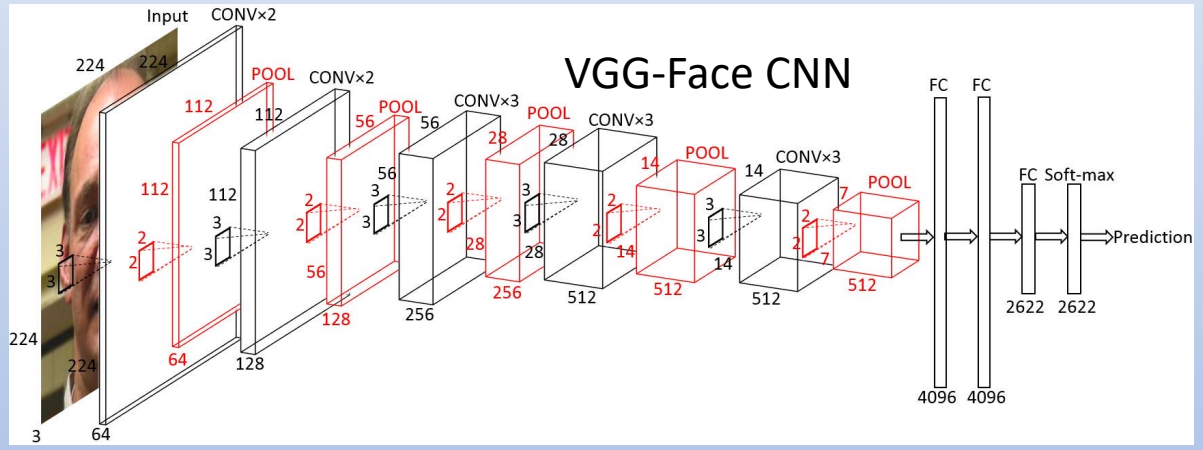
Not any arbitrary can solve over-fitting problem.



Merits of convolutional network, CNN



First successful CNN, LeNet5

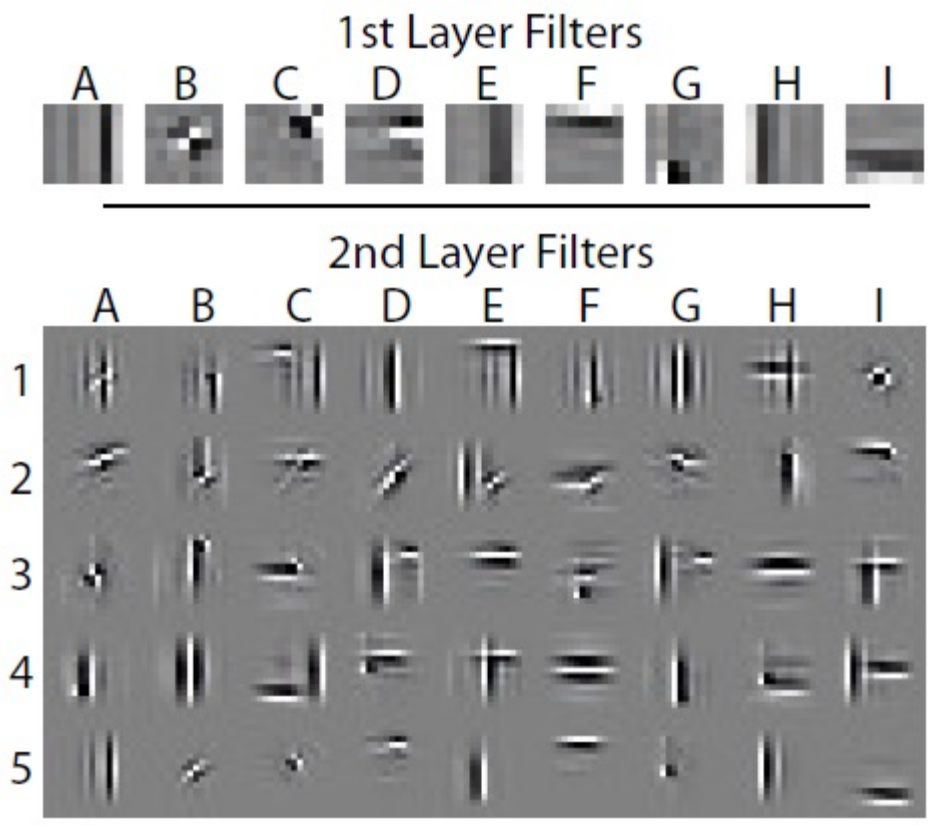


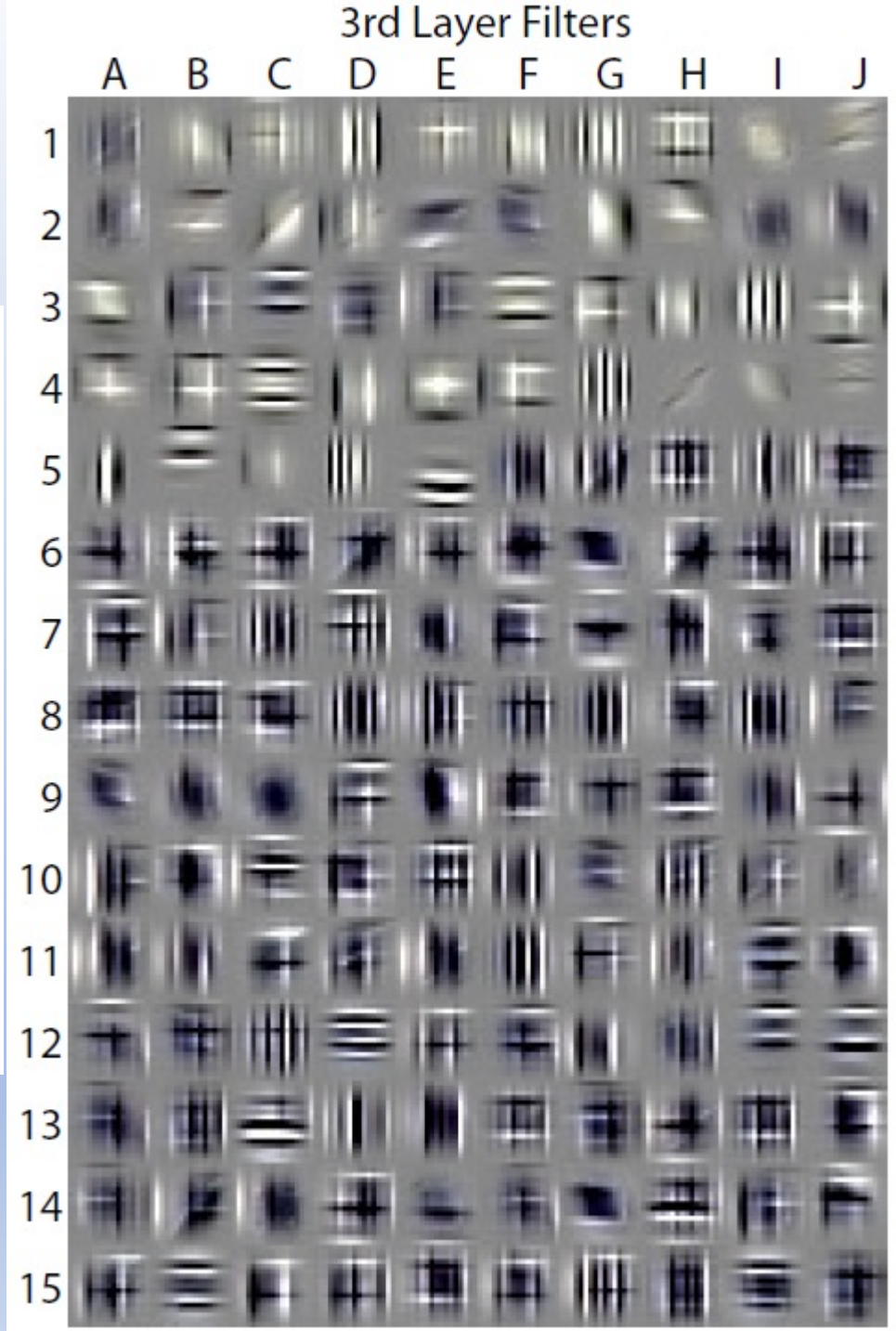
$$\mathbf{w}_j^q = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{g}^q \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$o_j^q = g_j^q * x_j$$
$$= \sum_{i=1}^{n_k} g_{j-i}^q x_i$$
$$= \mathbf{w}_j^{qT} \mathbf{x}$$

1. Small filter size, forcing all other weights zero, captures image local structure / pattern.
2. The convolution kernel, filter, is an image. Convolution process is the same as correlation processing, matched filter.
3. An input image patch similar to the filter mask produces high output while those dissimilar to the filter mask produce low outputs.
4. A filter is trained to extract a local image structure, blob, corner, line, edge, curve, etc.

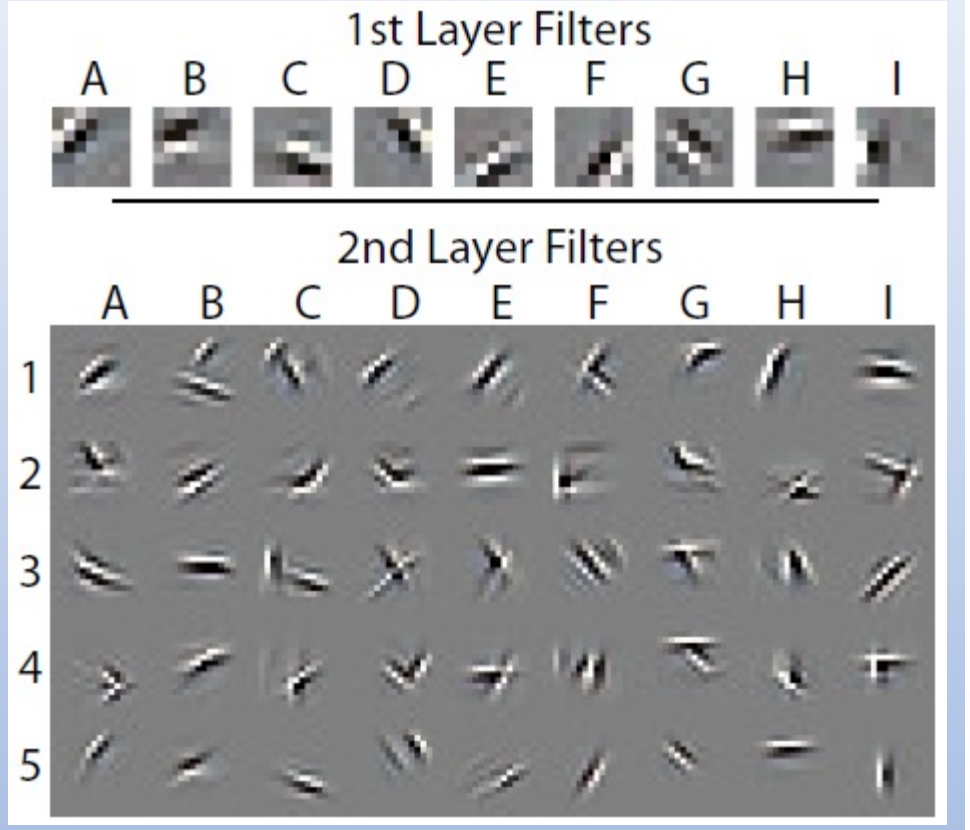
Further Examples:



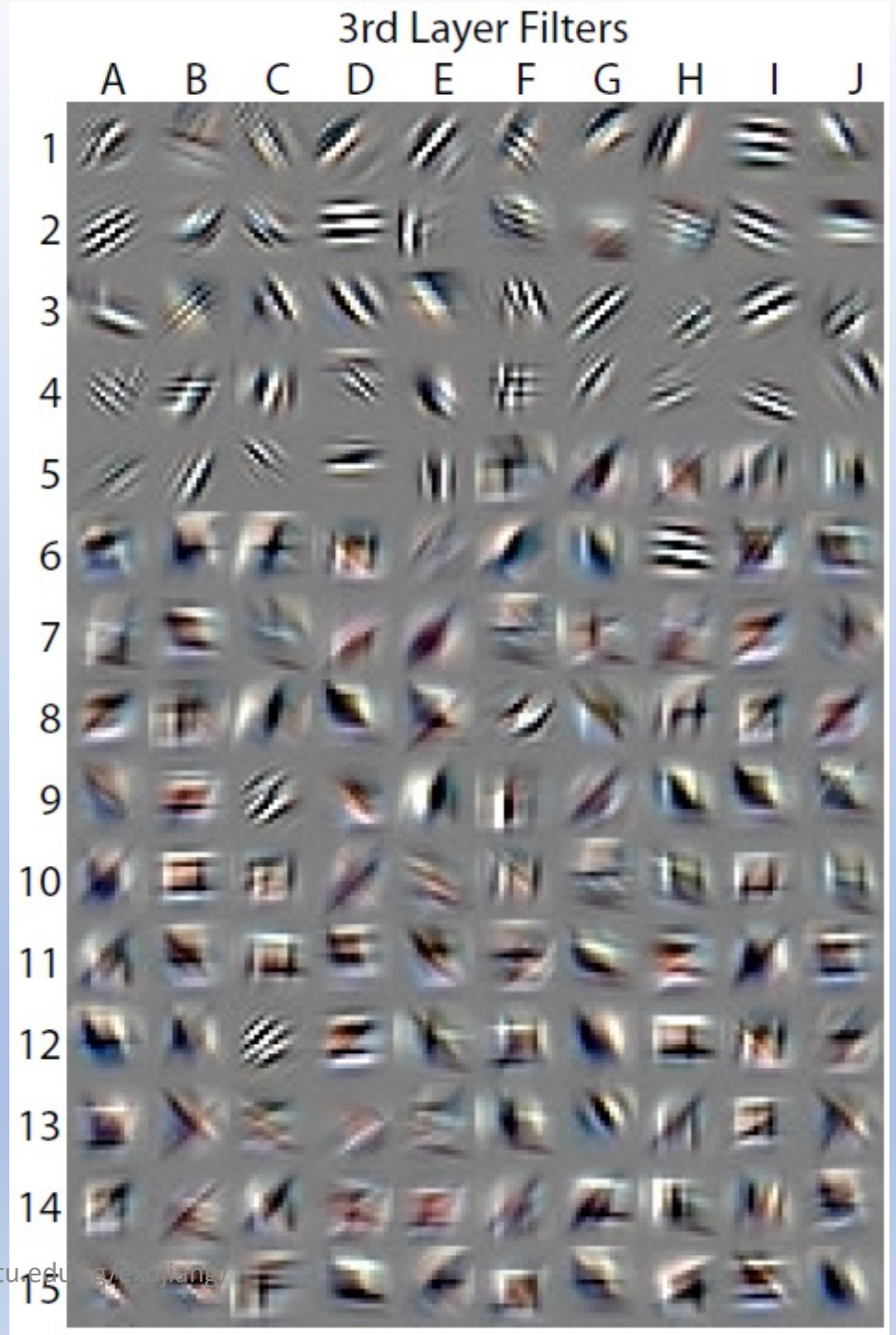


Filters trained on the city dataset. Note the predominance of horizontal and vertical structures.

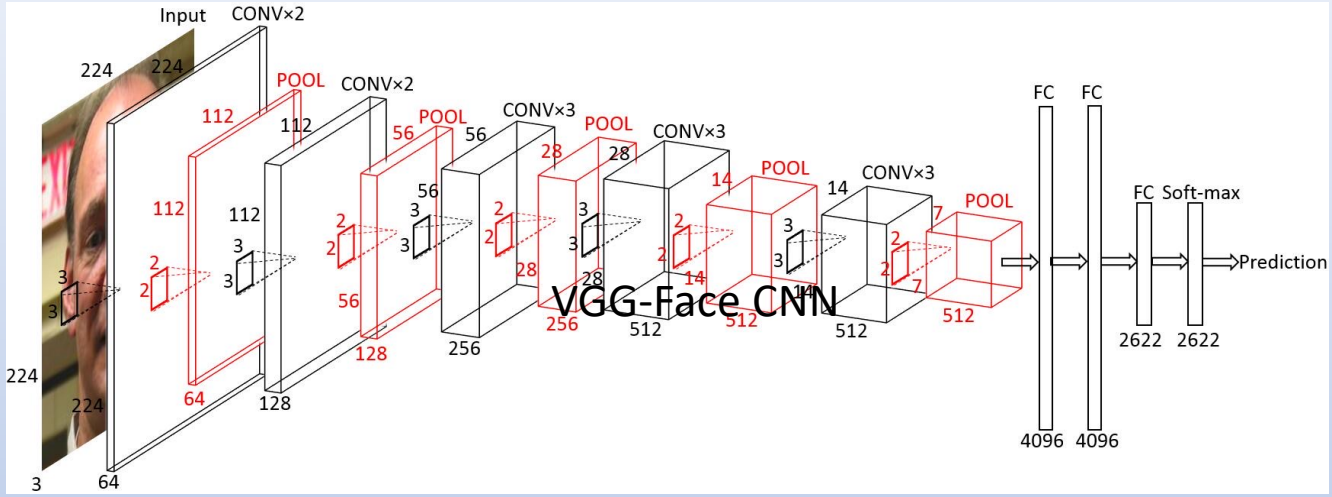
Further Examples:



Filters trained on food scenes. Note the rich diversity of filters and their increasing complexity with each layer. In contrast to the filters shown in previous slide, the filters are evenly distributed over orientation.



Further merits of convolutional network, CNN



$$\mathbf{W}^q = \begin{pmatrix} \mathbf{g}^q & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & \ldots \mathbf{g}^q \ldots & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & \mathbf{g}^q \end{pmatrix}$$

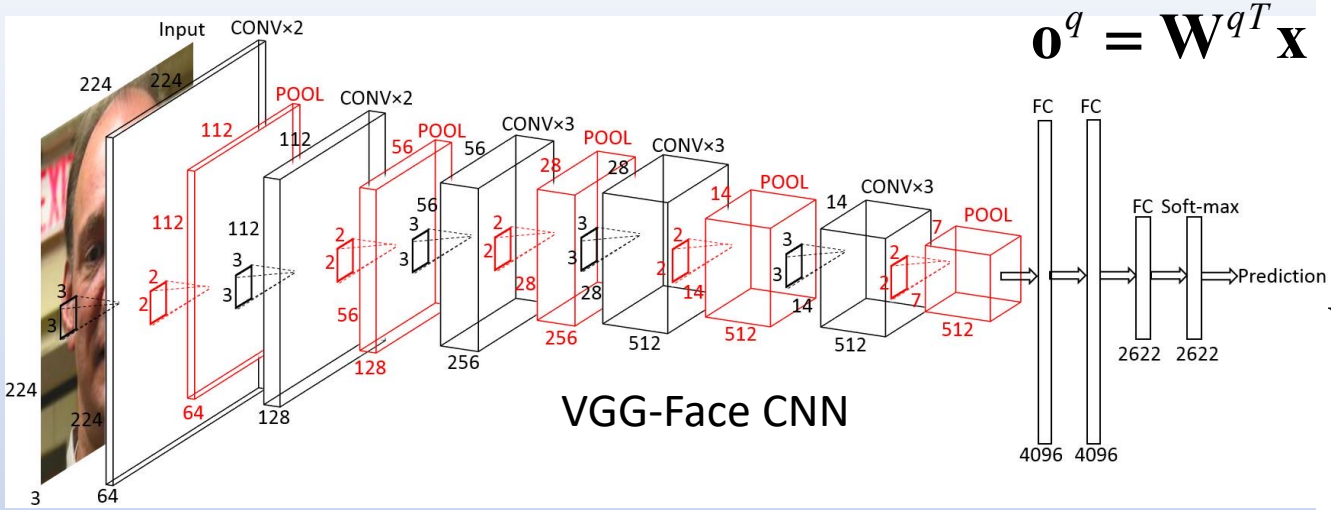
$$\mathbf{o}^q = \mathbf{W}^{qT}\mathbf{x}$$

5. Similar image local structures may appear at many different locations of image. Convolution process of one filter extracts one image local structure at all locations

6. Multiple filters extract multiple image local structures at all locations. Pooling process convert spatial information into assemble information, a set of image local structure.

7. Pooling process also reduces the image size, scale, so that the same filter size at higher layer captures larger scale image local structure. Pooling leads to spatial insensitive features

Further merits of convolutional network, CNN



$$\mathbf{o}^q = \mathbf{W}^{qT}\mathbf{x}$$

$$\mathbf{W}^q = \begin{pmatrix} \mathbf{g}^q & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & \ldots \mathbf{g}^q \ldots & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & \mathbf{g}^q \end{pmatrix}$$

8. Why deep network with many layers? Image structure/pattern has different scales; Cascade connected multiple layers to extract multi-scale features are more efficient than parallel connection. Regularize learned parameter for large receptive filed.

9. Simple nonlinear activation function ReLu leads to efficient and effective training.

10. How deep CNN overcome overfitting problem to make the extracted features robust to novel unseen data? **Effectively convert one image as a training sample to every pixel as a training sample. Convolution/Filter => shared networks. All output pixels share the same network.**