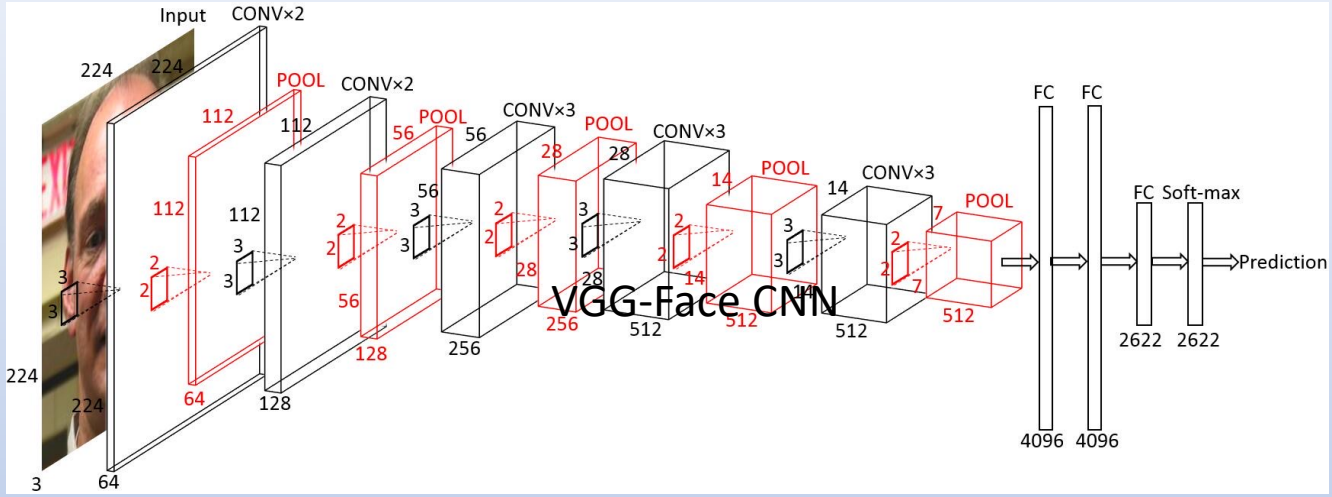
Further merits of convolutional network, CNN



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

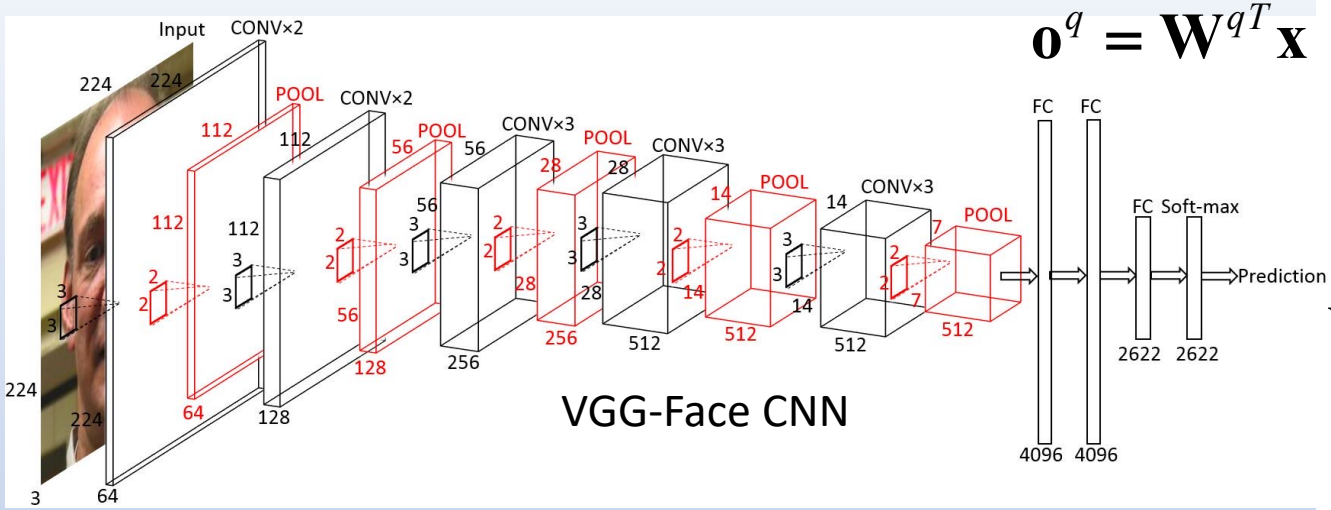
$$\mathbf{o}^q = \mathbf{W}^{qT}\mathbf{x}$$

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.**

Topic 14

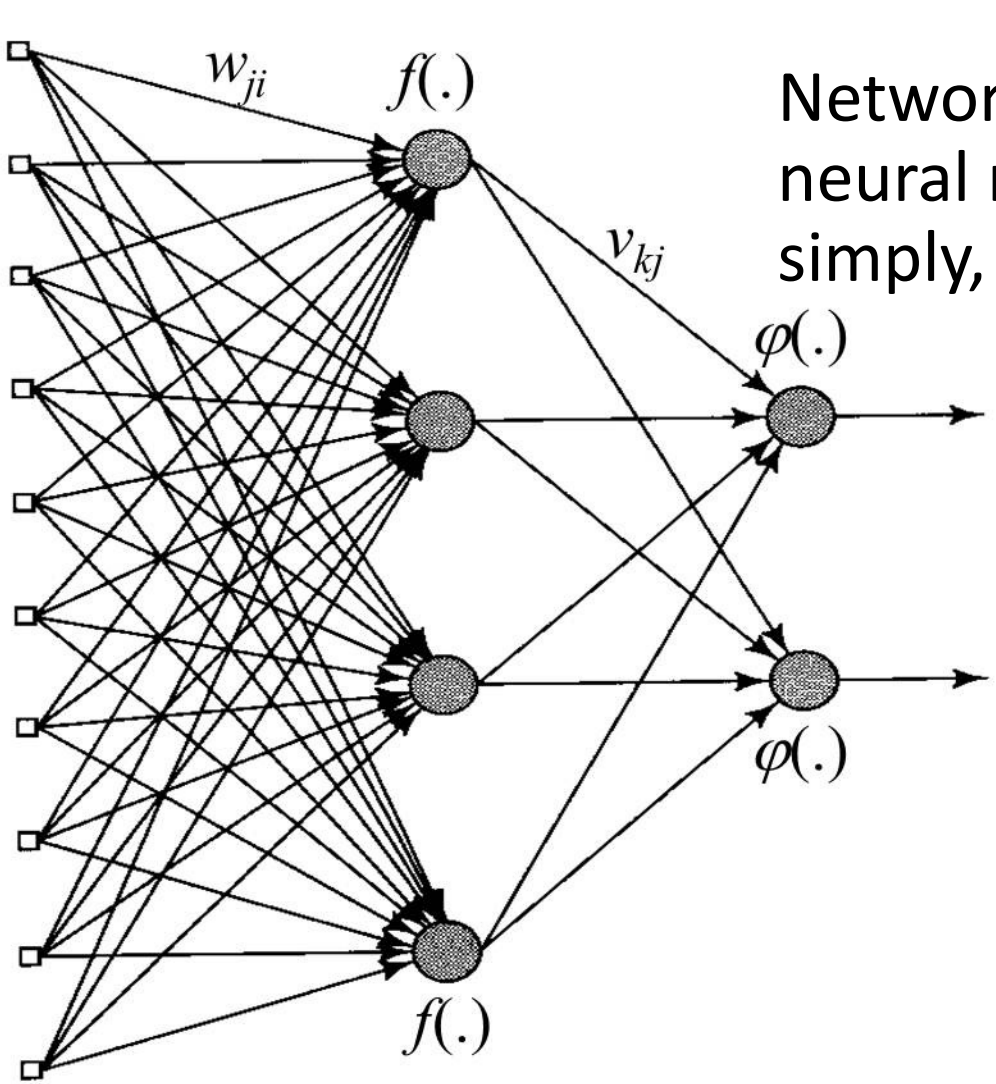
Neural Networks and Deep Machine Learning: from MLP to CNN

Neural Networks and Deep CNN

Outline

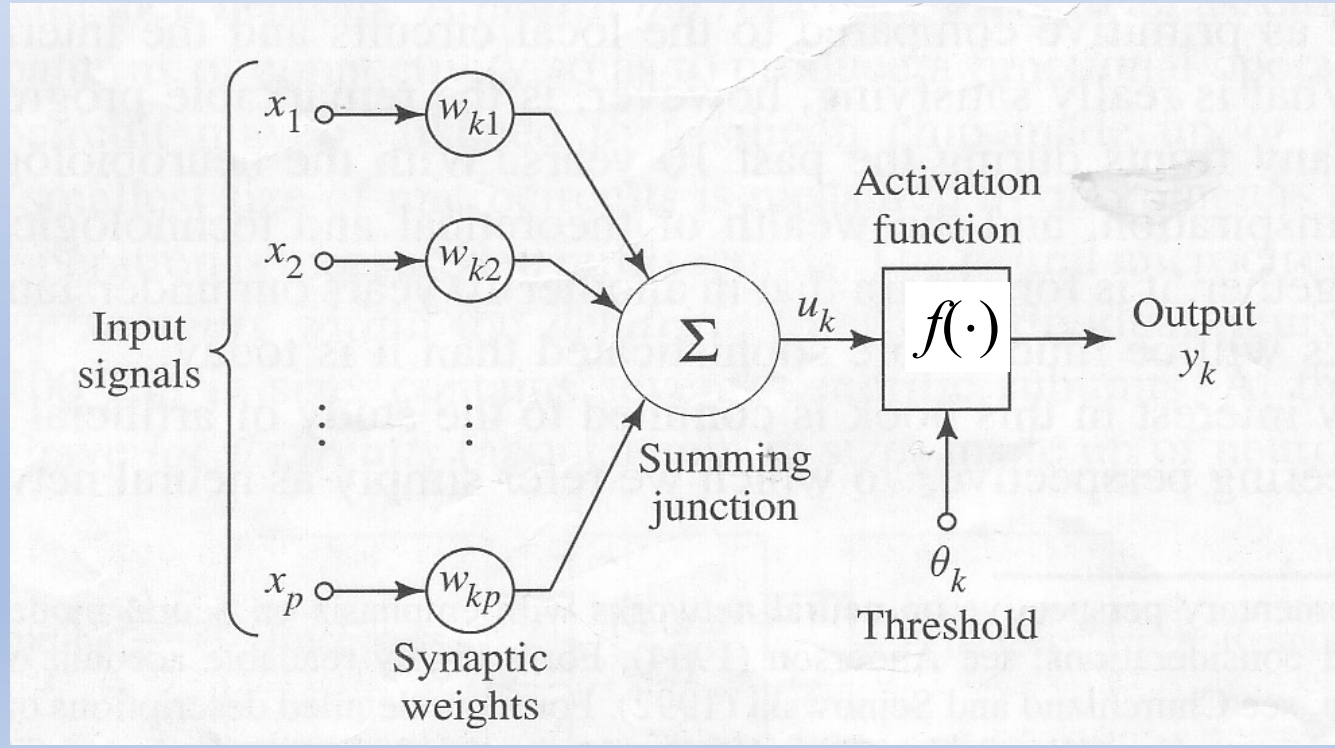
- Network Architecture and Neuron Model
- Optimization and Iterative (Error Correction) Learning
- Multilayer Perceptron (MLP) and Backpropagation Learning Algorithm
- Understanding Deep Learning and Convolutional neural Networks (CNN)
- **Deep Learning: from CNN to Transformer**

Neural Networks and Deep CNN



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

Neuron Model.



Neural Networks and Deep CNN -- Neuron Model

In mathematical terms, we can describe the neuron as:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \qquad y_k = f(u_k - \theta_k)$$

Where $x_1, x_2, \ldots, x_p$ are the input signals, $w_{k1}, w_{k2}, \ldots, w_{kp}$ are the synaptic 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.

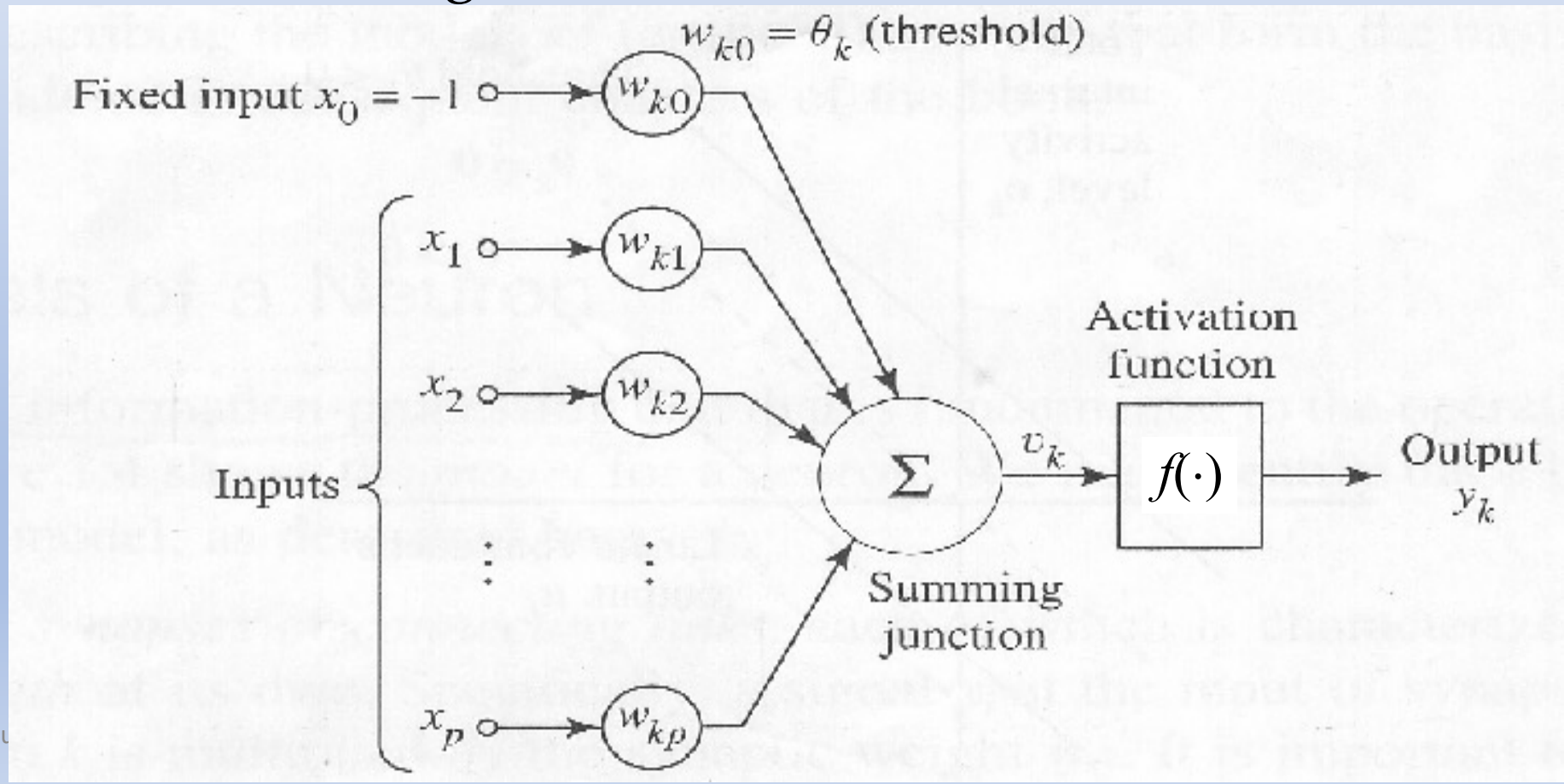
θ_k is an external parameter, we can consider this parameter as an input variable:

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

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

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

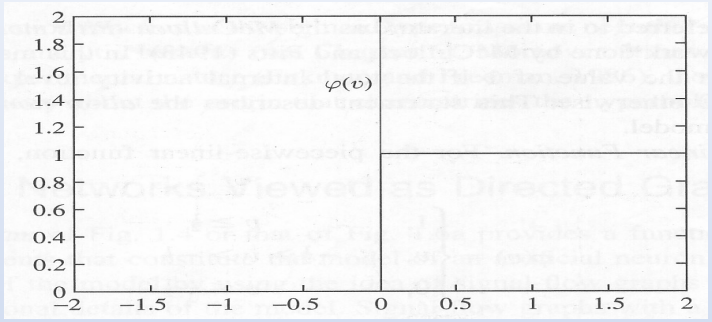
Sigmoid Function

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

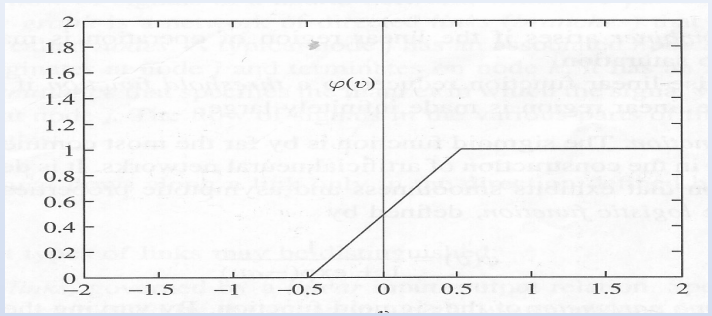
ReLU Function

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

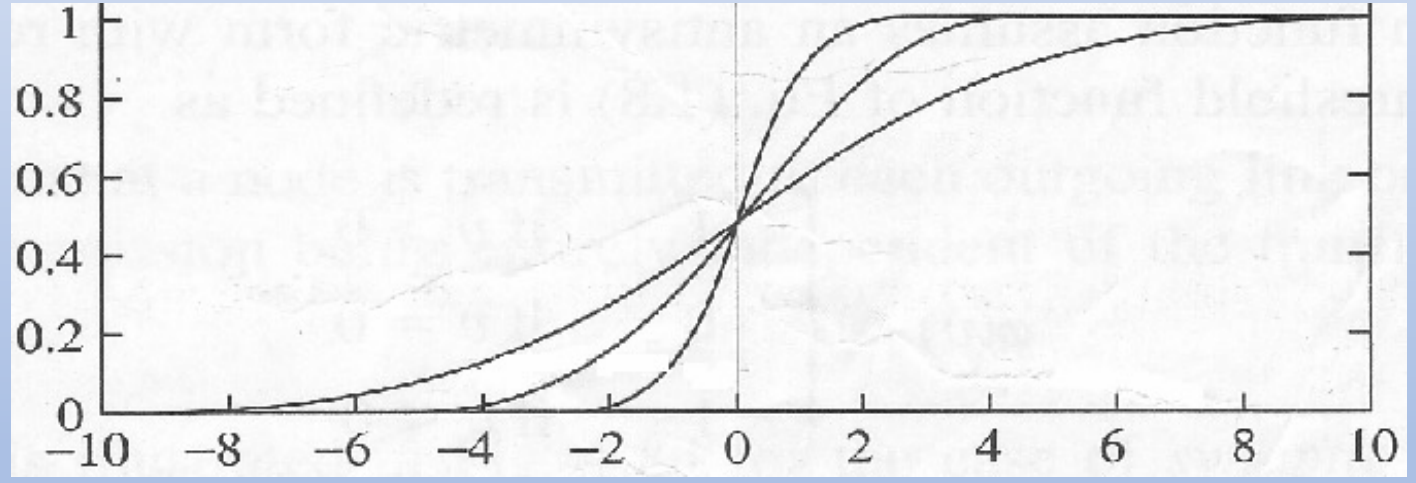
Neural Networks and Deep CNN -- Activation Functions



binary

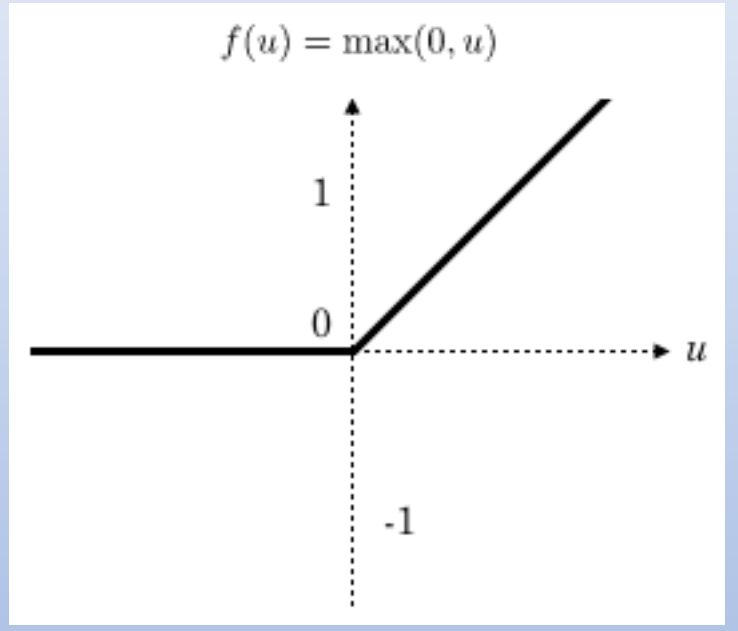


Piecewise linear



sigmoid

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



ReLU

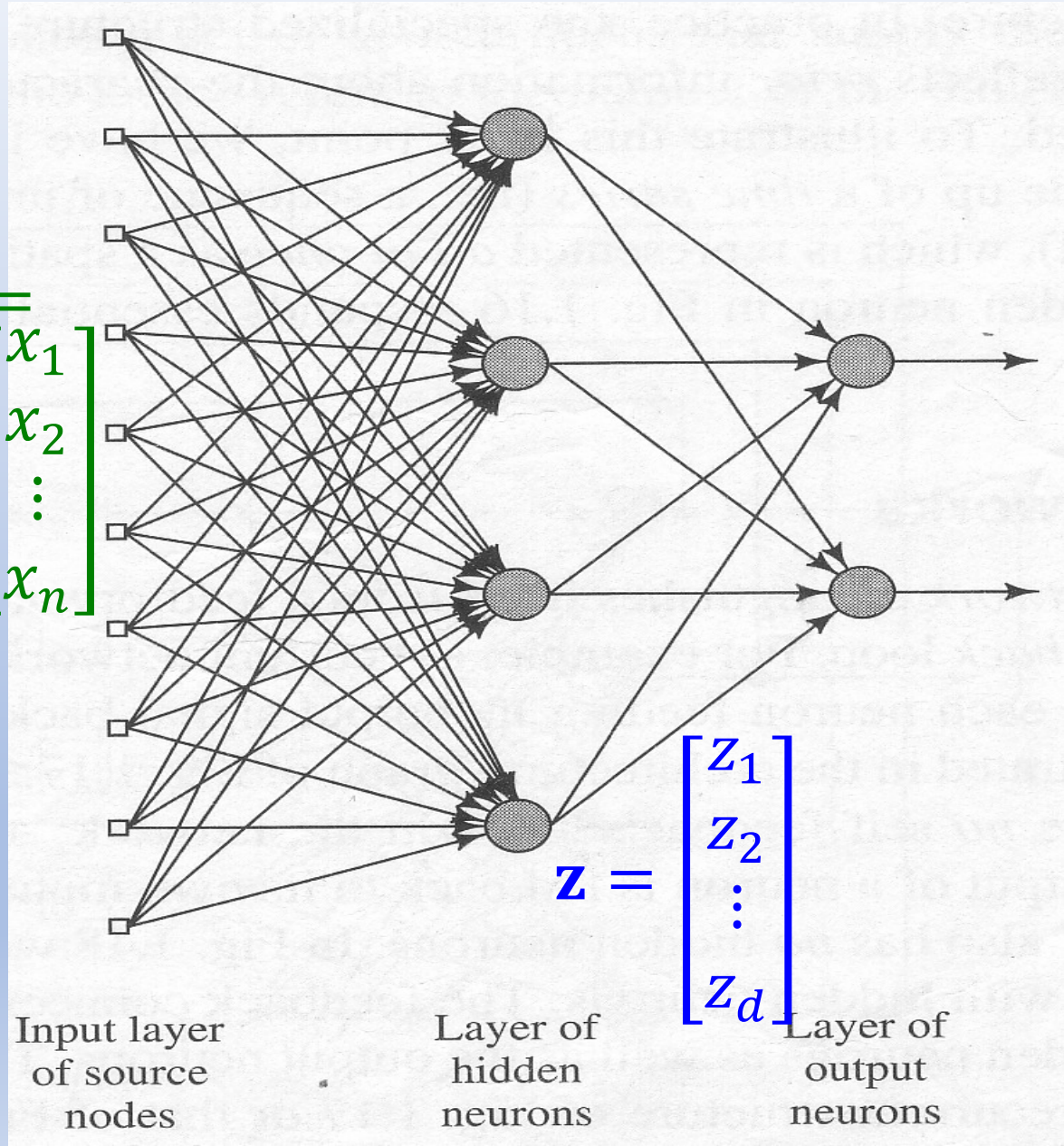
$$f(v) = max(0,v) = \begin{cases} v, & for\ v \geq 0 \\ 0. & for\ v < 0 \end{cases}$$

ANN – Multilayer Perceptron (MLP)

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

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

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



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

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

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

ANN – Multilayer Perceptron (MLP)

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

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

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

$$\Downarrow$$

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

ANN – Multilayer Perceptron

- If the activation faction f is a linear function,

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

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

- 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:

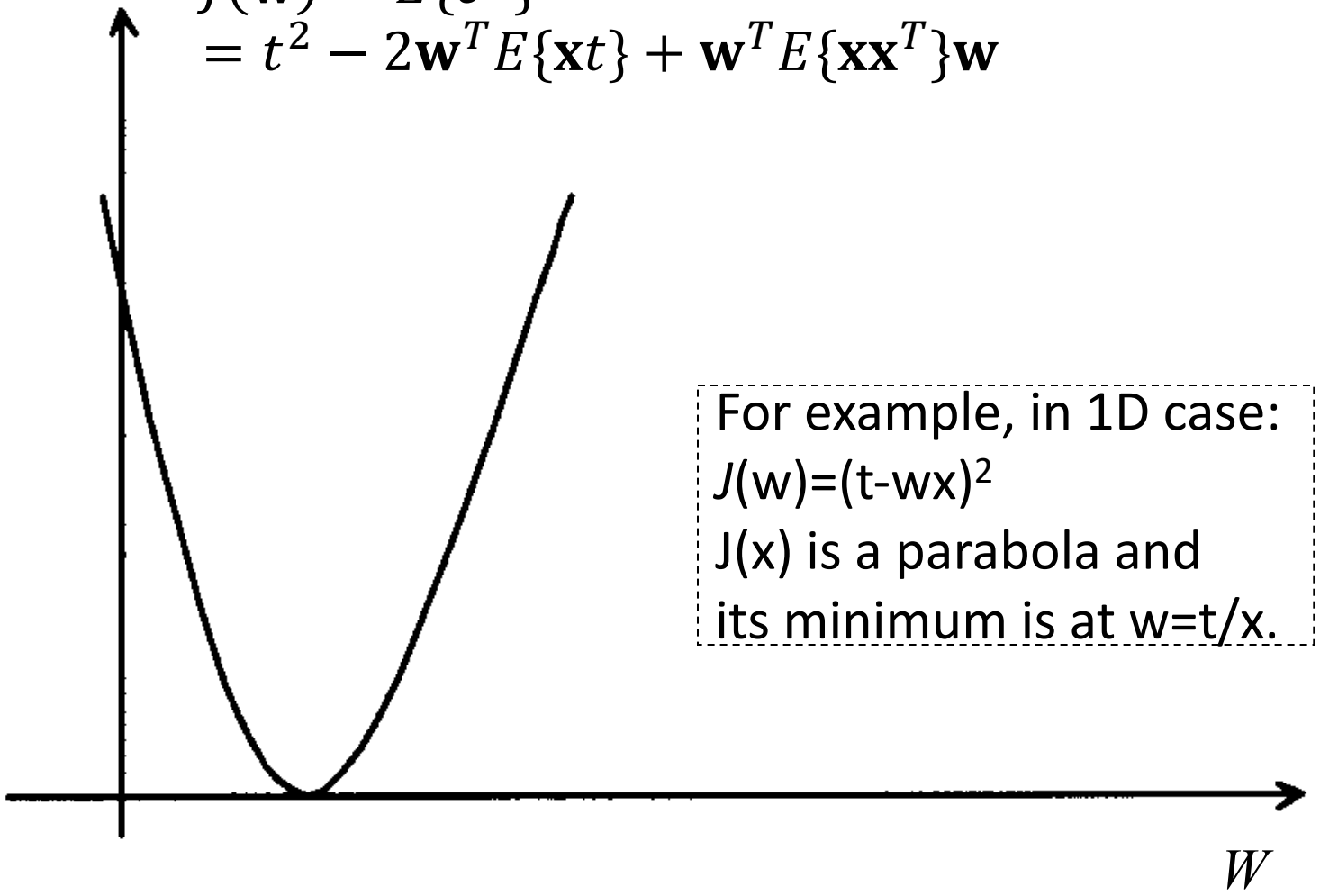
$$J(\mathbf{w}) = E\{e^2\} = E\{(t - y)^2\} = E\{(t - \mathbf{w}^T \mathbf{x})^2\}$$

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

- 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 $\mathbf{w}$:

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

$$\Downarrow$$

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

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

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

ANN – Single Layer Neural Network

- Let $\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_q], \qquad \mathbf{t} = [t_1 \quad t_2 \quad \cdots \quad t_q]$

- Then:

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

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

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

- It is also called least square method.

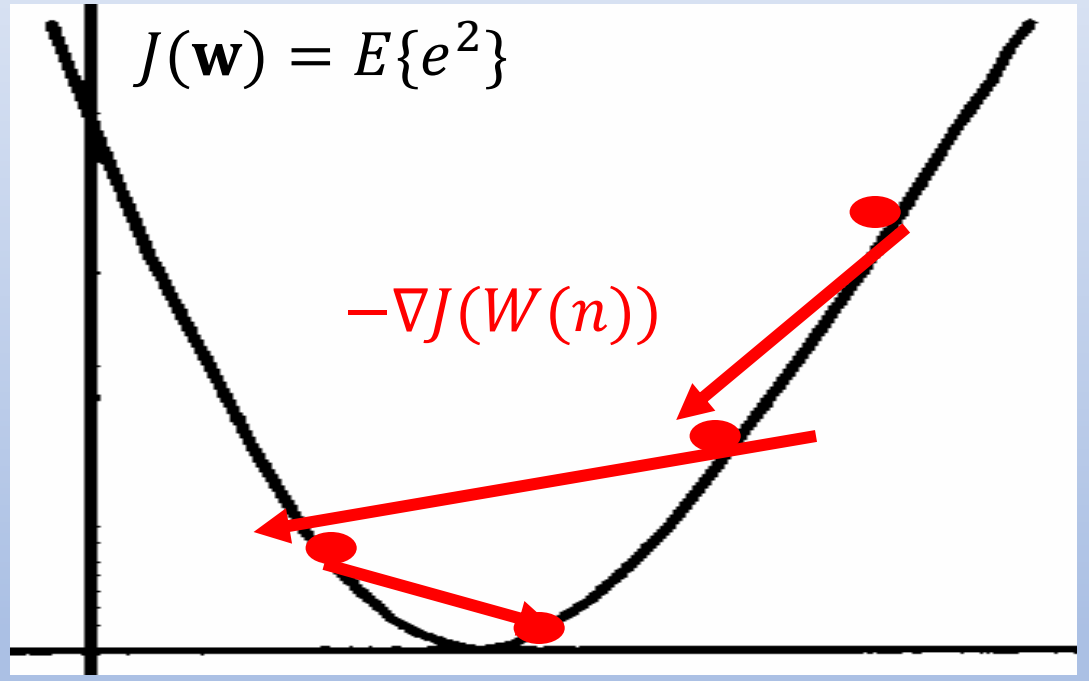
ANN – Single Layer Neural Network

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

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

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

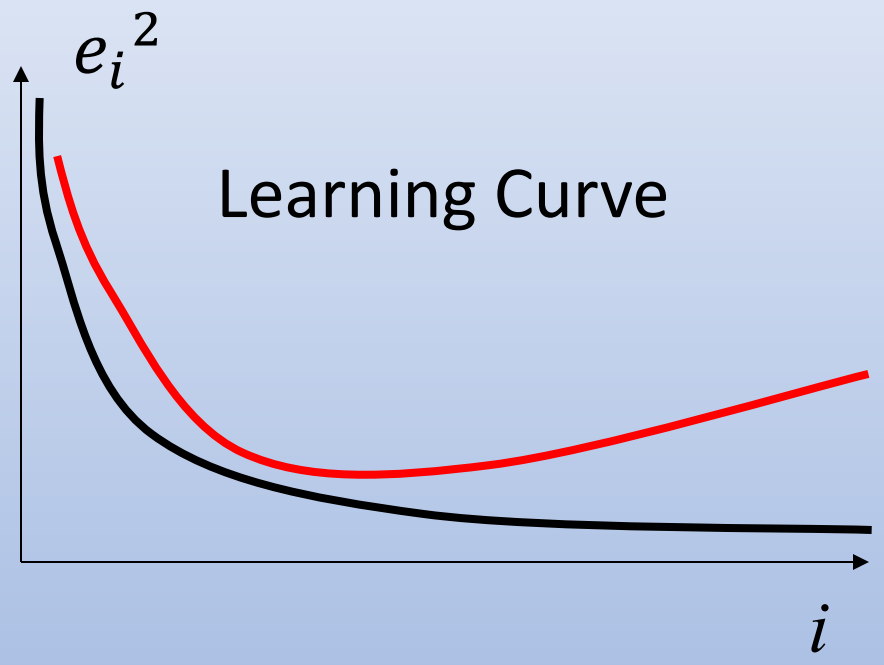
$$= -2E\{(t - \mathbf{w}^T\mathbf{x})\mathbf{x}\}$$
$$= -2E\{e\mathbf{x}\} \cong -2e_i\mathbf{x}_i$$

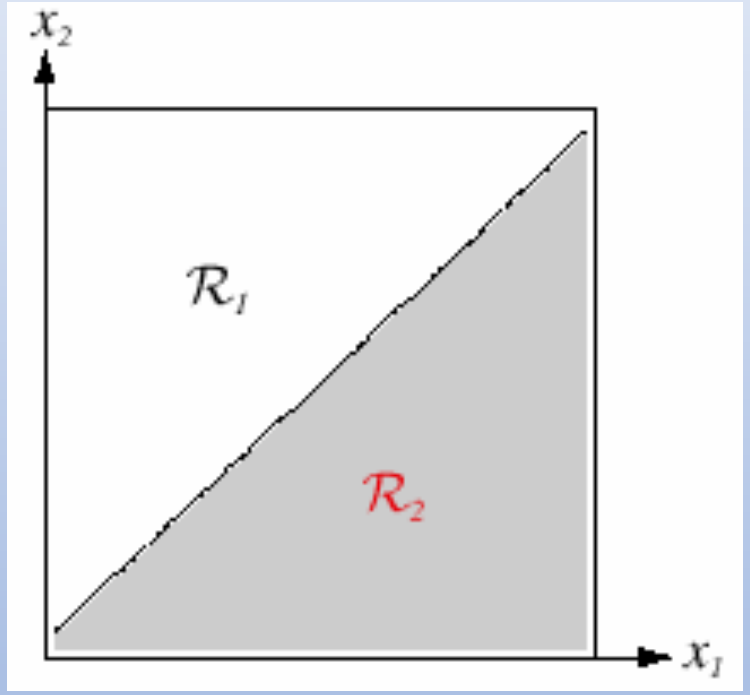


$$\mathbf{w} \Leftarrow \mathbf{w} + \eta e_i\mathbf{x}_i = \mathbf{w} + \eta(t_i - y_i)\mathbf{x}_i$$
$$\rightarrow \mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{t}^T$$

ANN – Learning Curve and Decision Boundary

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





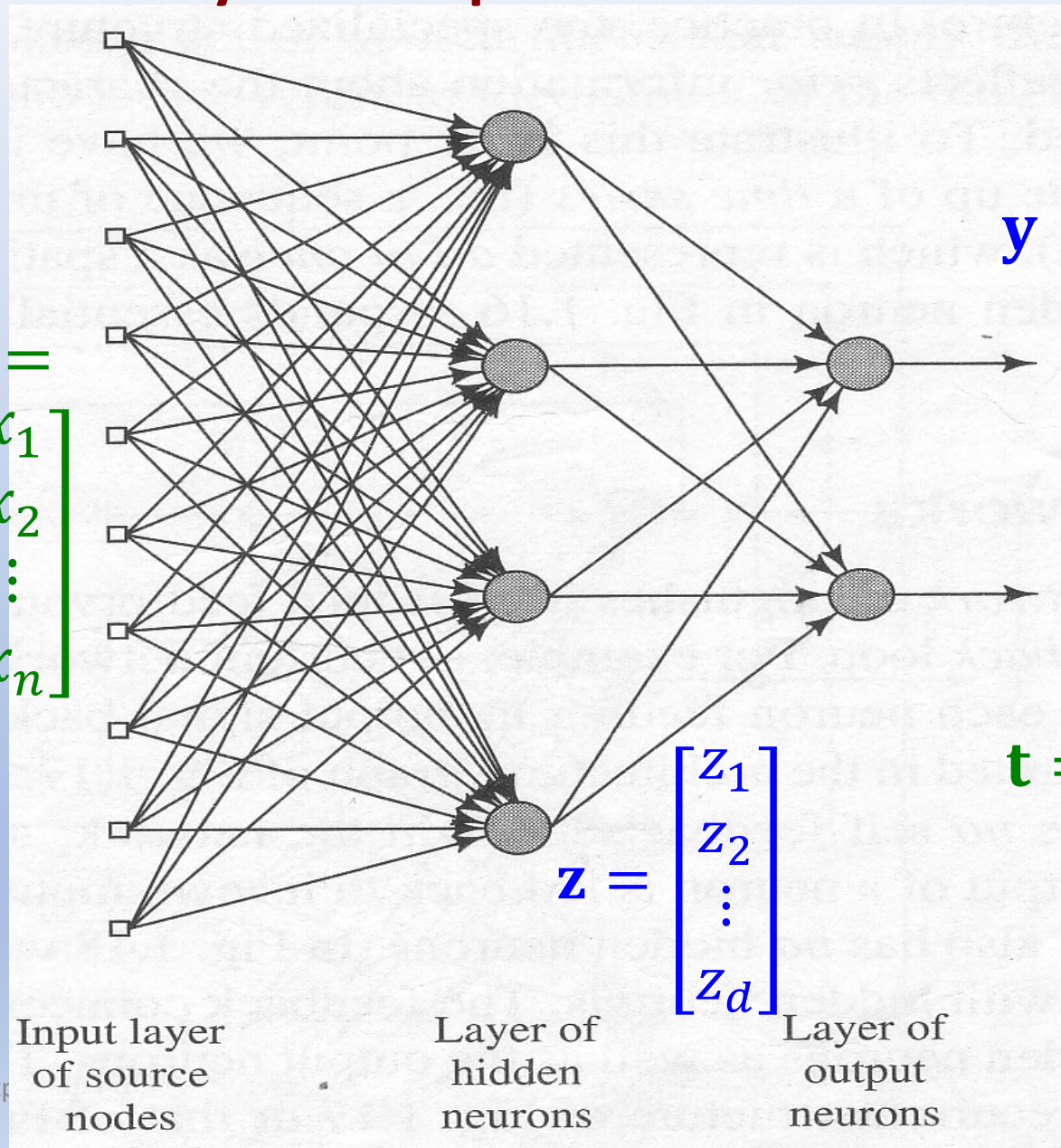
Decision boundary is a strait line or plane or hyper-plane.

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

ANN – Multilayer Perceptron

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

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 set the interconnection weights based on the training patterns and the desired outputs.
- It is a straightforward matter to understand how the output, and thus the error, depend on the hidden-to-output layer weights.
- 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

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

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

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

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.

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

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

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

ANN – Multilayer Perceptron/backpropagation

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

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

$$\frac{\partial J(\mathbf{w},\mathbf{v})}{\partial w_{ji}} = \frac{\partial J}{\partial z_j}\cdot\frac{\partial z_j}{\partial q_j}\cdot\frac{\partial q_j}{\partial w_{ji}} = -\left[\sum_{k=1}^{c}(t_k - y_k) f'(q_k) v_{kj}\right] f'(q_j) x_i$$

ANN – Multilayer Perceptron/backpropagation

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

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

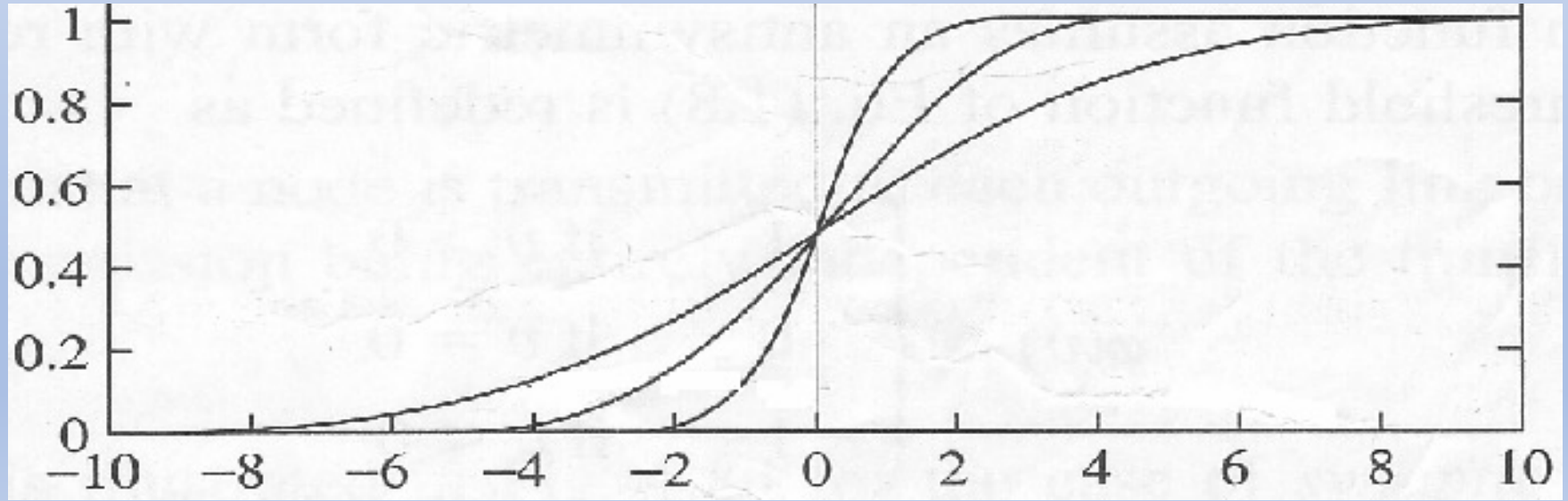
$$v_{kj} \Leftarrow v_{kj} + \eta (t_k - y_k) f'(q_k) z_j$$

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

ANN – Multilayer Perceptron/backpropagation

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

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



ANN – Multilayer Perceptron/backpropagation

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

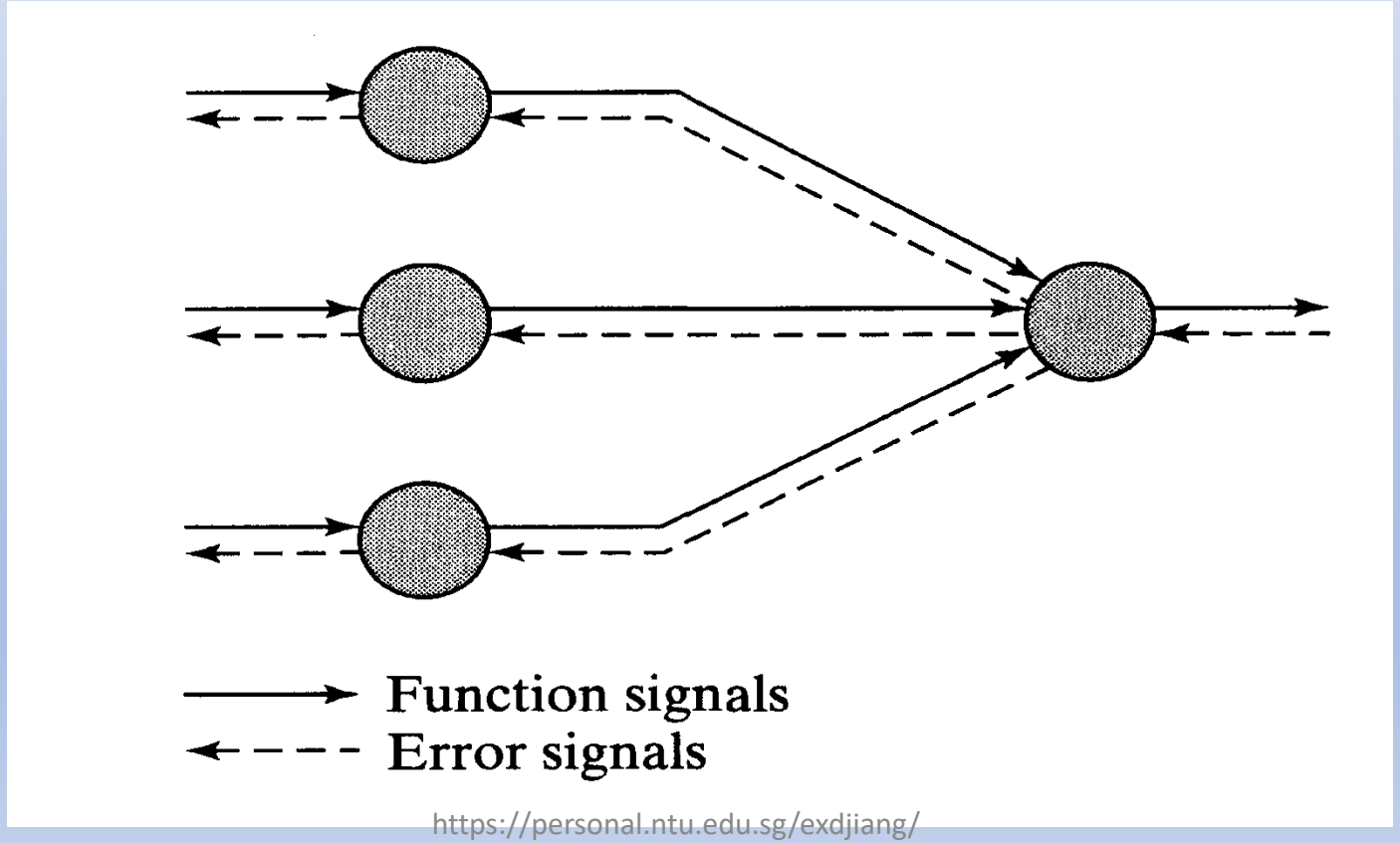
  - Feedforward

    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.

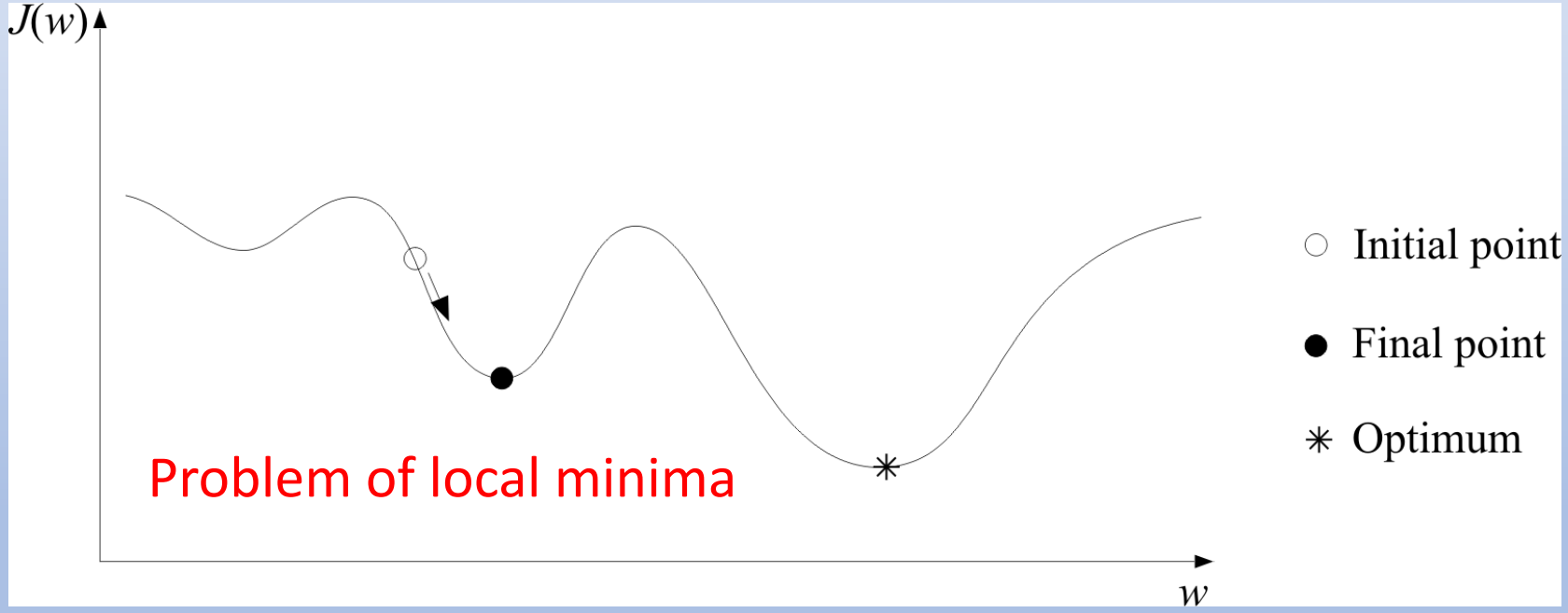
(5) Iteration

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 per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network
- The average 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 in order 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

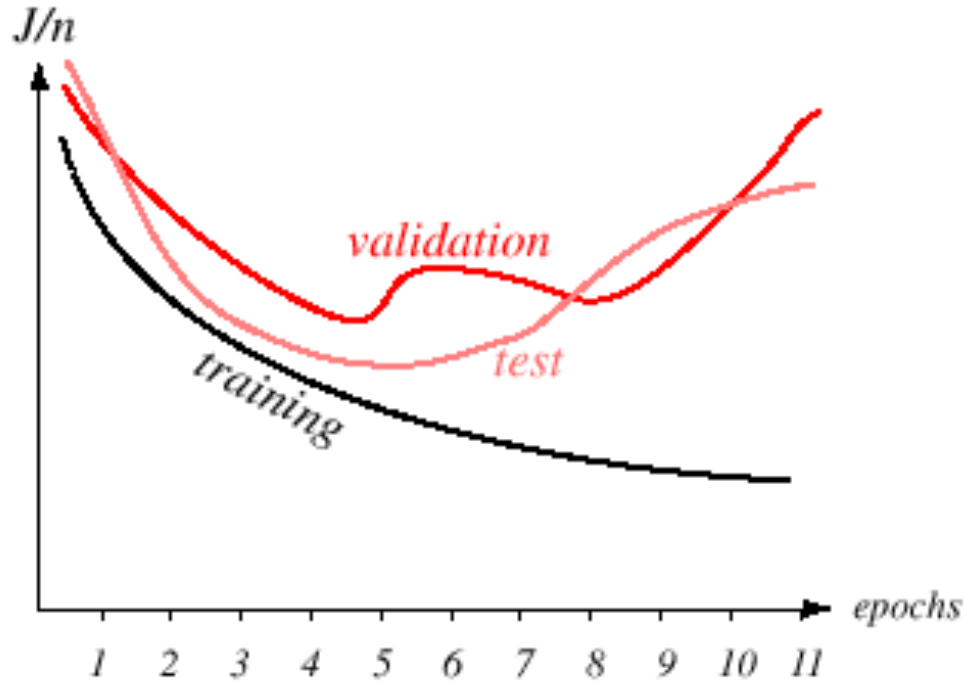
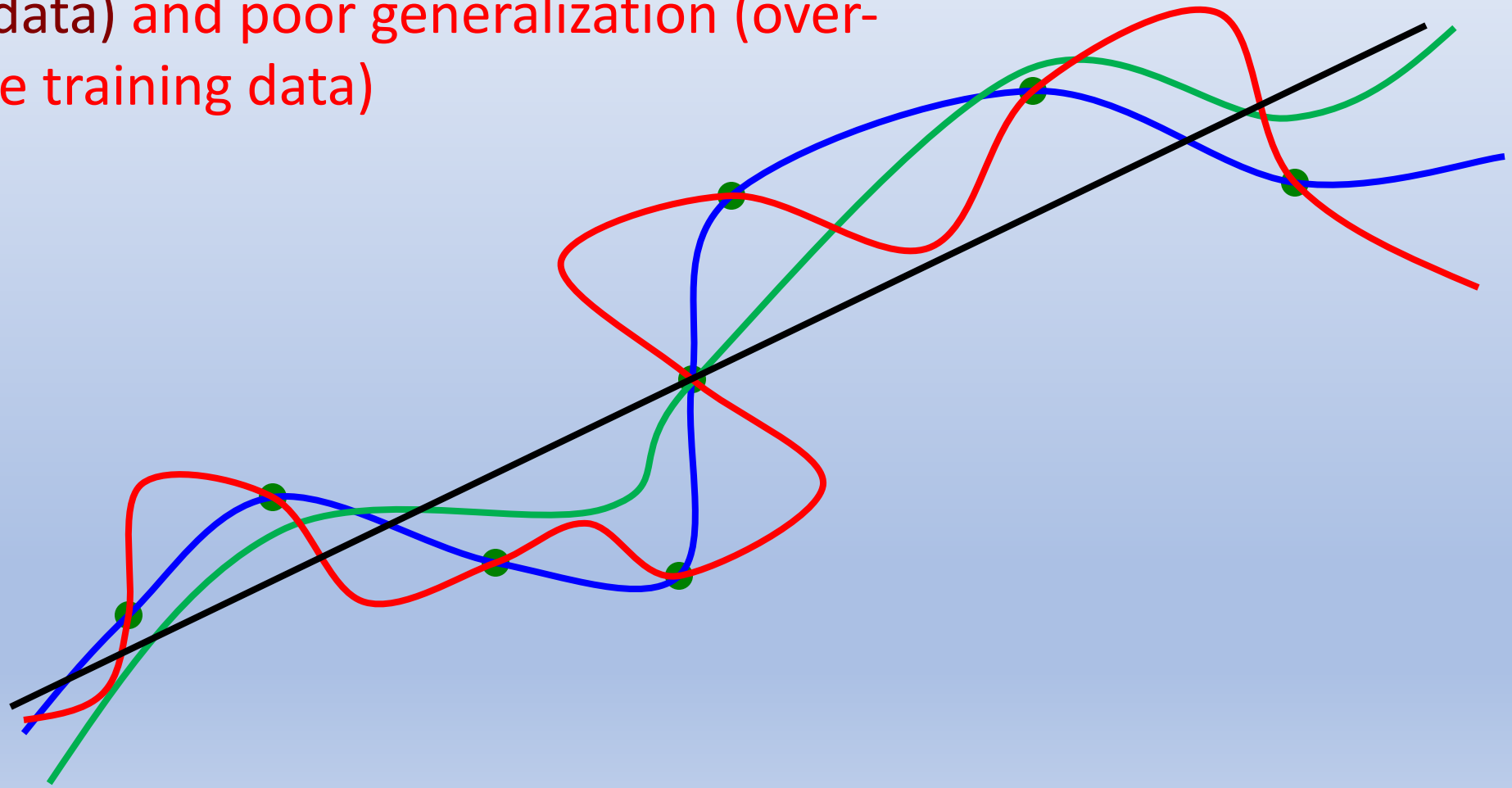


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization 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 – Understand Under- and Over-fitting

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



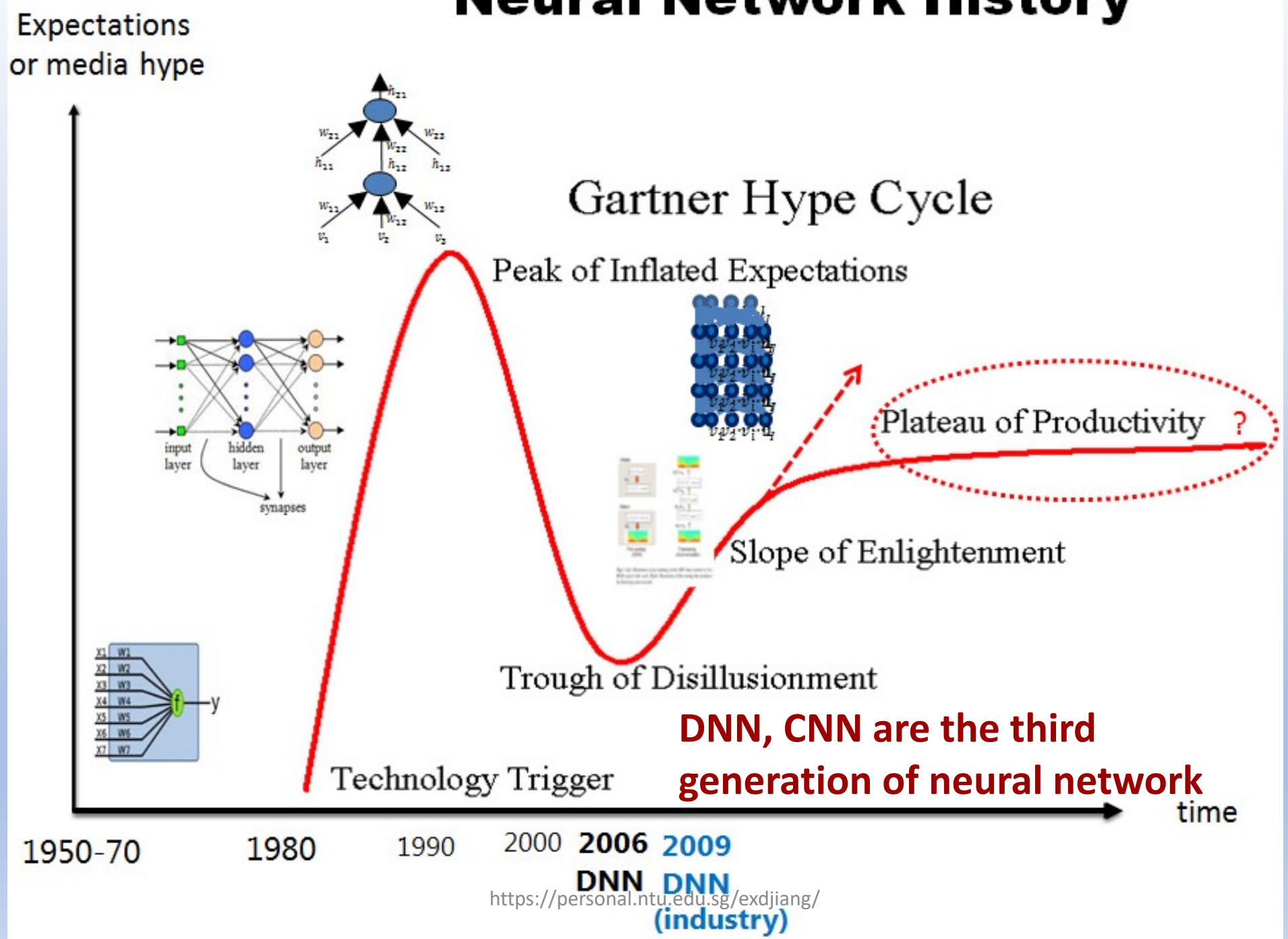
ANN – Conclusions of Neural Networks

- 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.

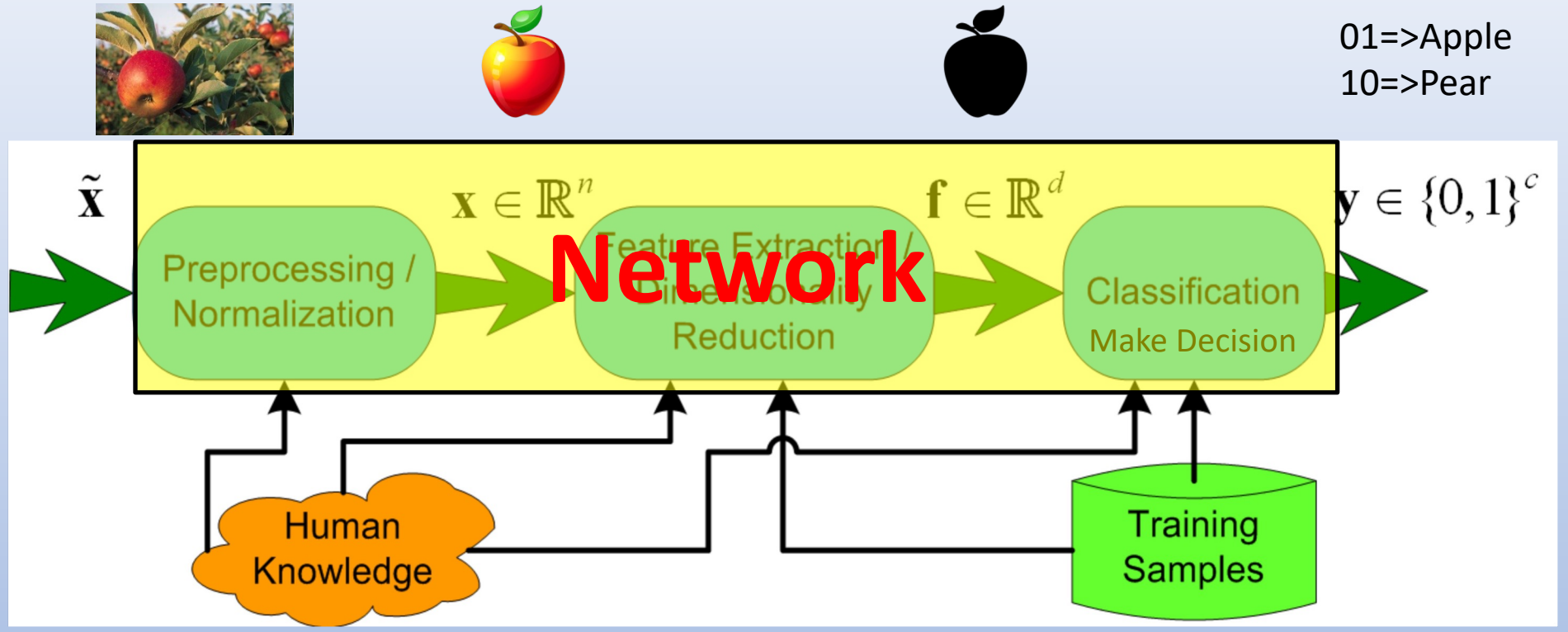
  However, these results are of greater theoretical interest than practical, since the construction of such a network requires a large enough number of nodes and the weight values are to be learnt! It is still an open question how to find the nonlinear functions based on that training data. Problems of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

Neural Network History



DNN, CNN are the third generation of neural network

Functionalities of Image Recognition Modules



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

Supervised learning: find an unknown mapping or continuous function

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

using finite discrete known training samples:

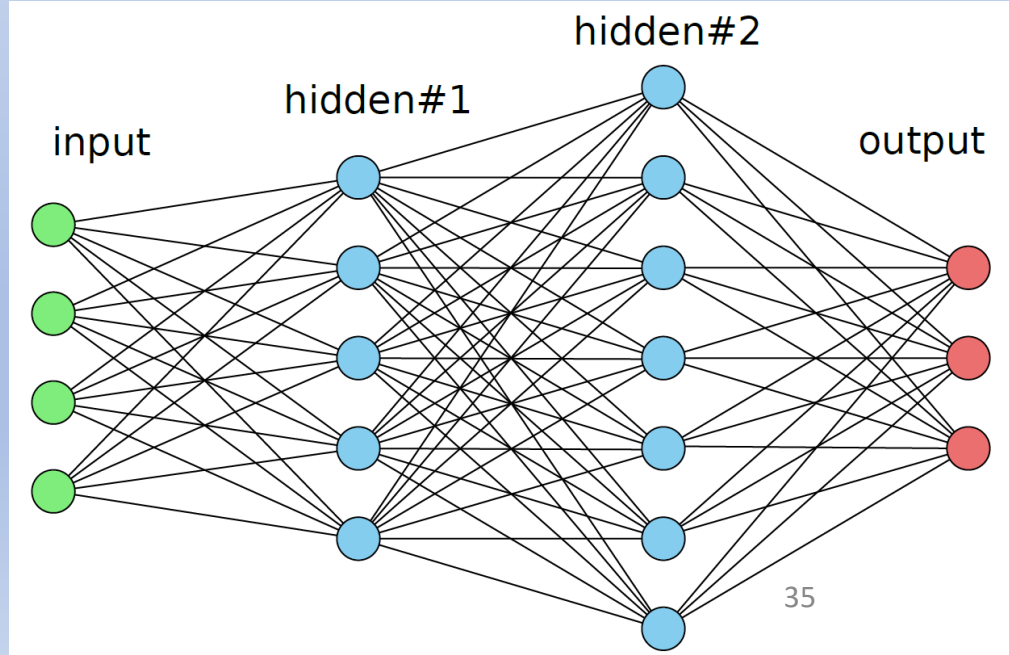
$\{\mathbf{x}_i, \mathbf{y}_i\}, \mathbf{y}_i = f(\mathbf{x}_i)$, for i=1, 2, ..., l.

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:

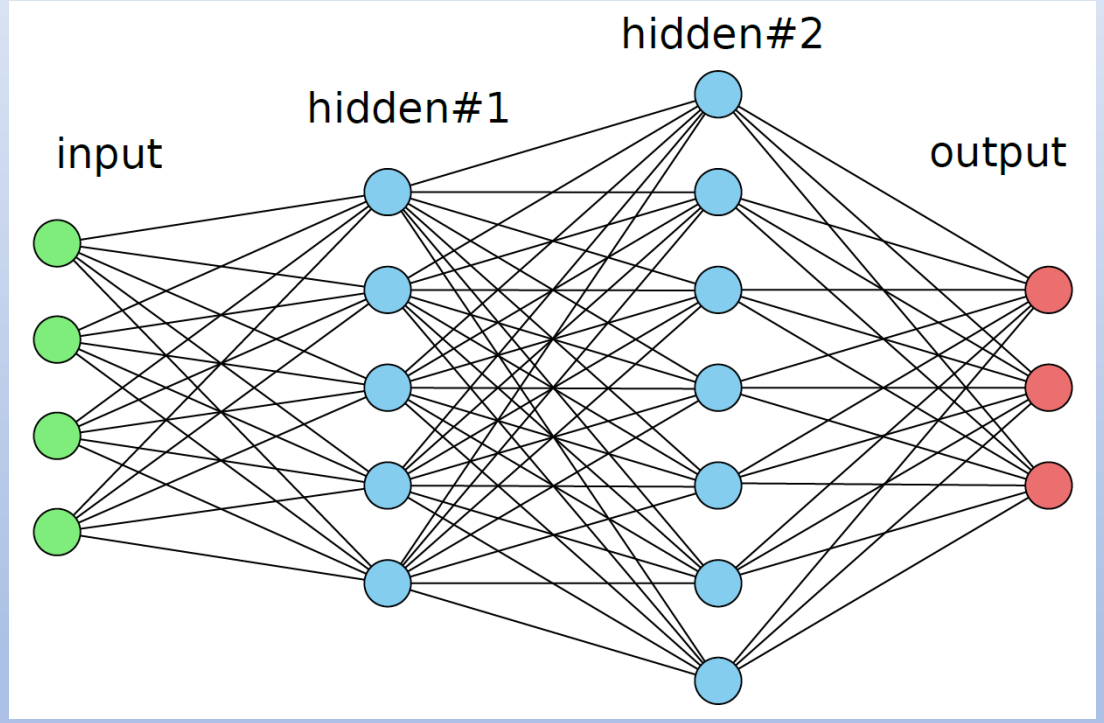
$$\mathbf{y} = h\left(\mathbf{W}_m^T \cdots h\left(\mathbf{W}_2^T h(\mathbf{W}_1^T \mathbf{x})\right)\right)$$

$h(\bullet)$: simple nonlinear function



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

- Theoretically, m=2 is sufficient to approximate any highly nonlinear function, i.e. $e \Rightarrow 0$
- What further can we do?
- Why NN became dead in 2000s?



- Deep Learning

  m>>2.

- Motivation?

  difficult to find solution using m=2?

  may be easier to find solution using m>2?

Problems of machine learning:

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

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 $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{Z}^c$

Problem of overfitting or poor generalization is serious

Compare to the continuous, uncountable infinite different values, a set of finite discrete samples, whatever large in size, is always negligible!

Example of MLP to solve highly nonlinear double spiral problem

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

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

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

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

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

How to regularize $\hat{f}$?

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

$$\Downarrow$$

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

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

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

or all of the above

Understanding Deep Learning and Convolutional Neural Networks, CNN

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

- Theoretically, m=2 is sufficient

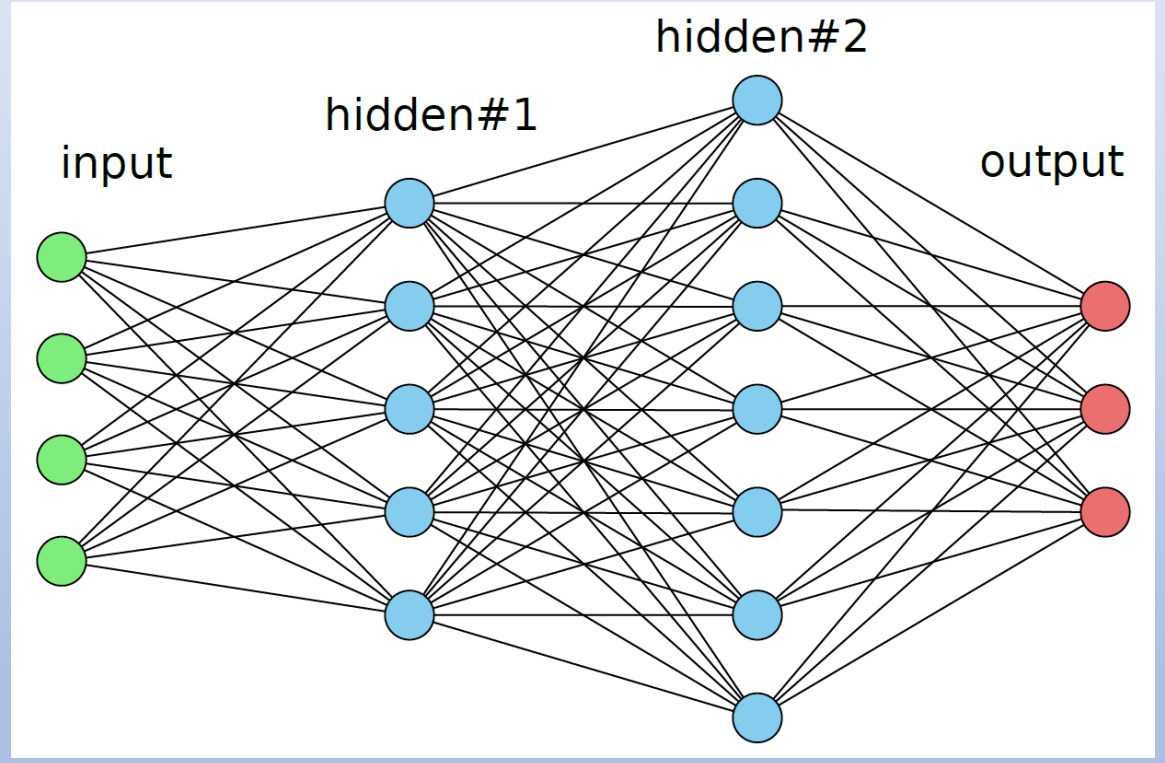
  But difficult to find solution.

- Deep Learning

  m>>2.

- Motivation:

  easier to find solution?

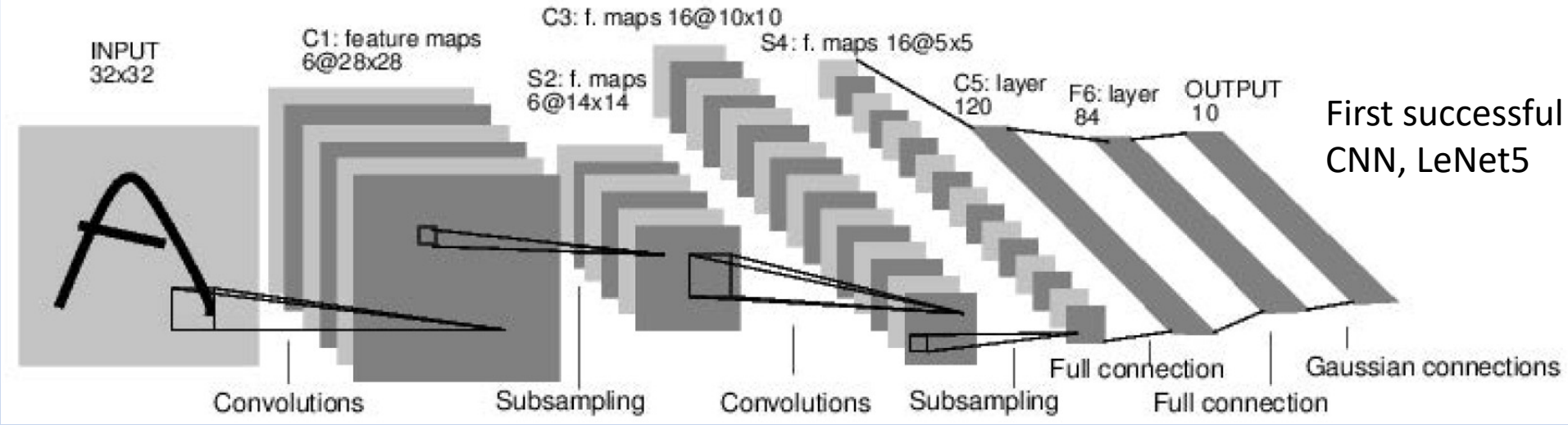


$$\mathbf{x}^{k+1} = h\left(\mathbf{W}_k^T \mathbf{x}^k\right), \qquad k = 1, 2, \ldots, m$$

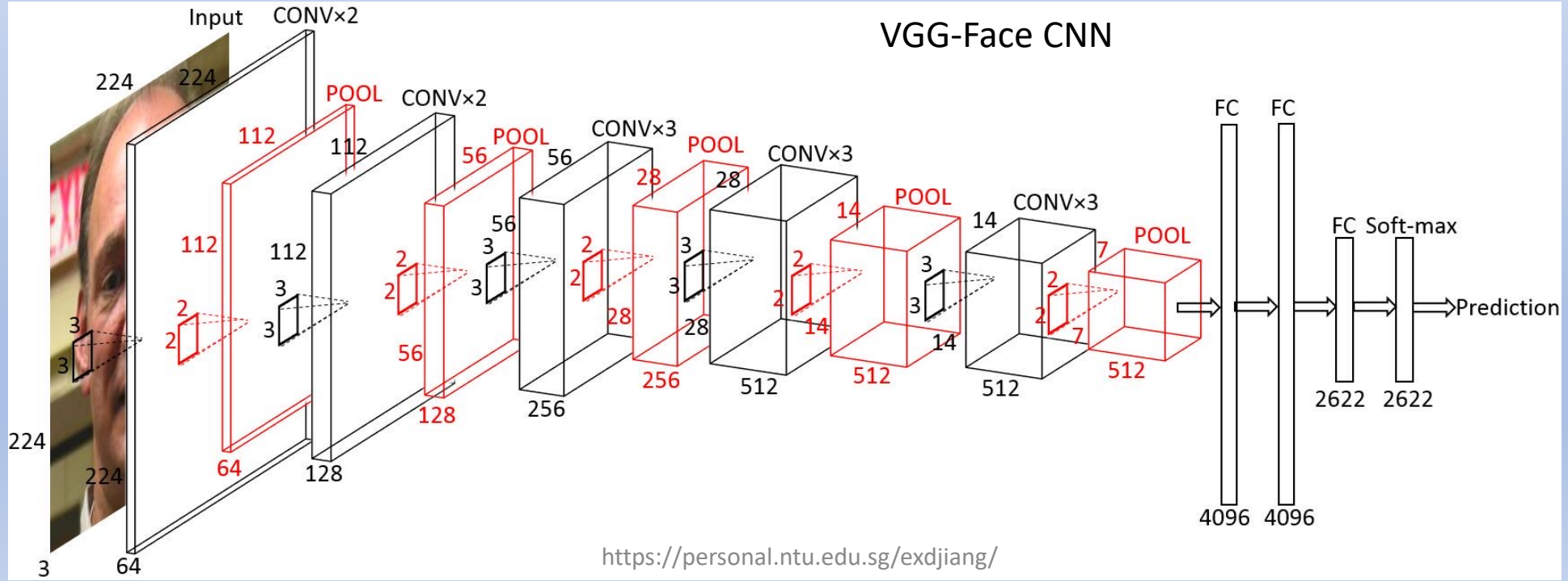
$$\mathbf{x}^1 = \mathbf{x}, \qquad \mathbf{y} = \mathbf{x}^{m+1}, \qquad \mathbf{o}^k = \mathbf{W}_k^T \mathbf{x}^k \Rightarrow \mathbf{o} = \mathbf{W}^T \mathbf{x}$$

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

Convolutional network CNN appears to be quite different from MLP?



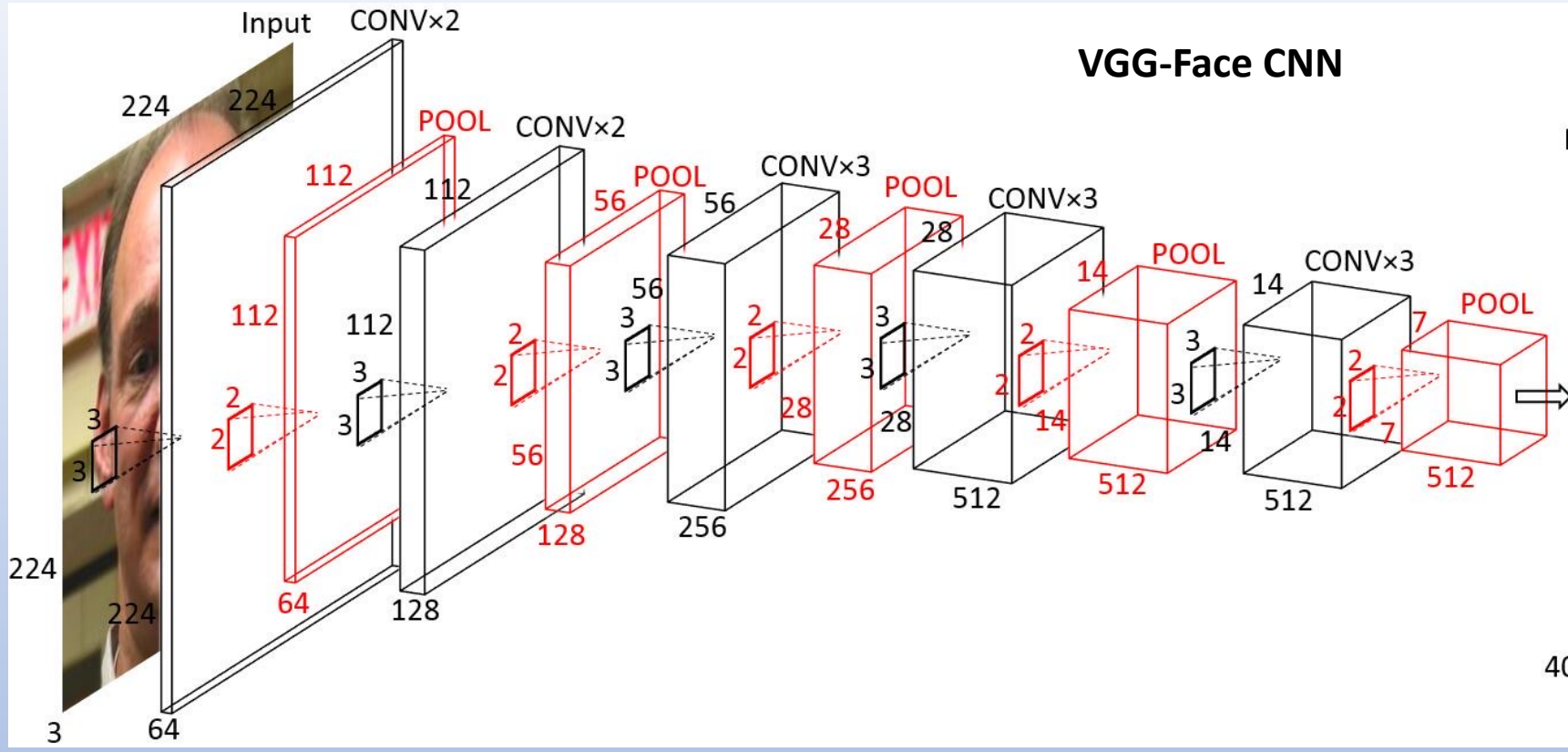
First successful CNN, LeNet5



Characteristics of Convolutional Network CNN

1. Network architecture
2. Convolution

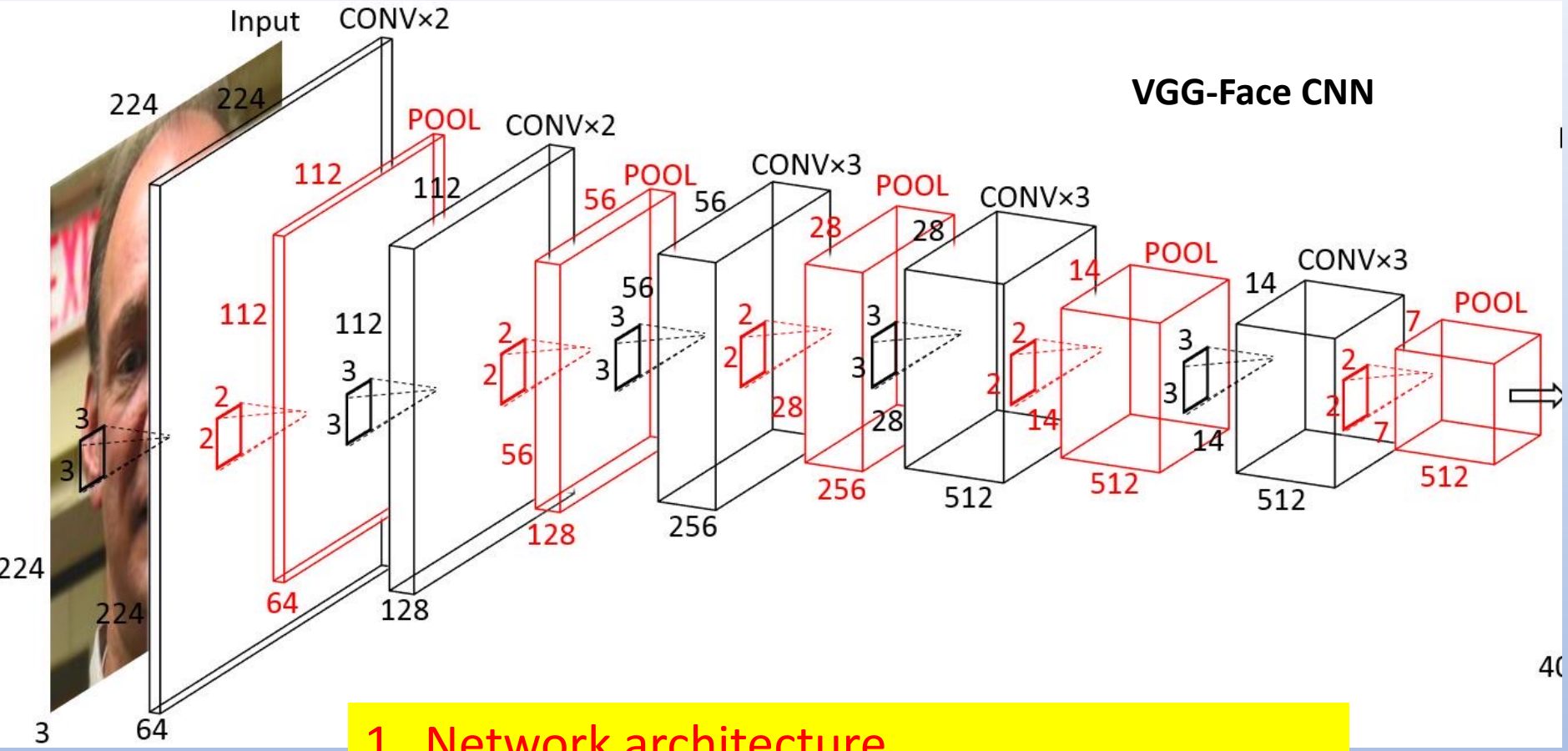
Input feature maps of spatial size $P \times Q$ and C channels and output feature maps of spatial size $P \times Q$ and D channels are expressed



$x_{i,j,k}$, $1 \le i \le \mathrm{P}$, $1 \le j \le \mathrm{Q}$, $1 \le k \le \mathrm{C}$, and $y_{i,j,k}$, $1 \le i \le \mathrm{P}$, $1 \le j \le \mathrm{Q}$, $1 \le k \le \mathrm{D}$.

$$y_{i,j,k} = \sum_{n=1}^{c} \sum_{m=-1}^{1} \sum_{l=-1}^{1} w_{l,m,n,k} x_{i-l,j-m,n} , + b_k, 1 \le i \le \mathrm{P}, 1 \text{?} j \le \mathrm{Q}, 1 \text{?} k \le \mathrm{D}$$

Characteristics of Convolutional Network CNN



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

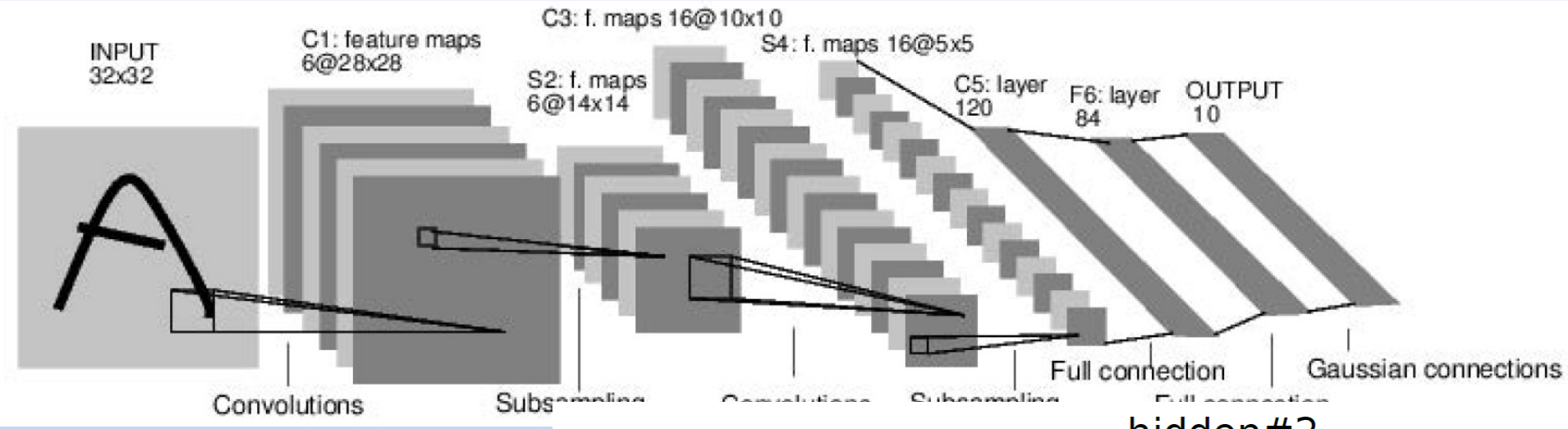
Convolution in CNN includes a bias term.

Along the channel dimension, all inputs of different channels are fully connected, same as MLP.

So 1X1 convolution makes sense in CNN.

What is the key characteristics of convolution?

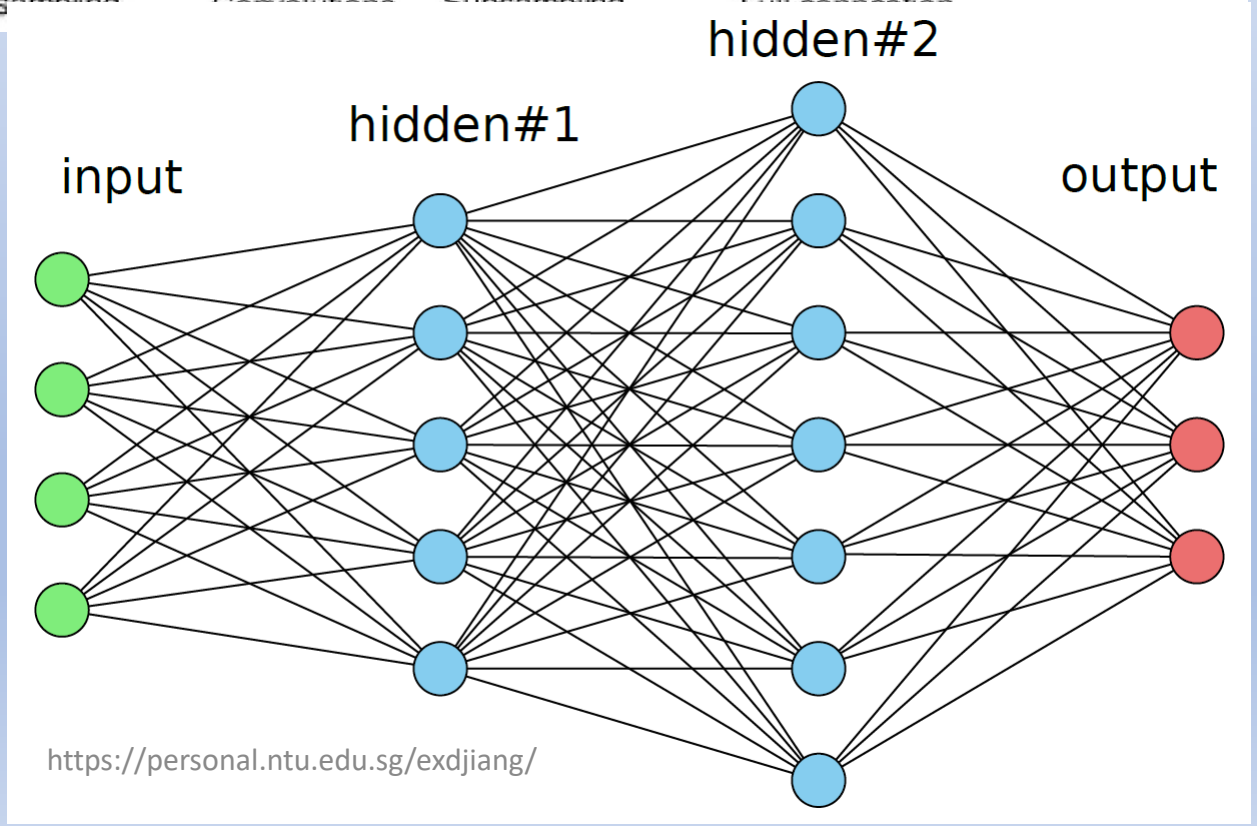
Compare CNN with MLP in 1D: Network architecture



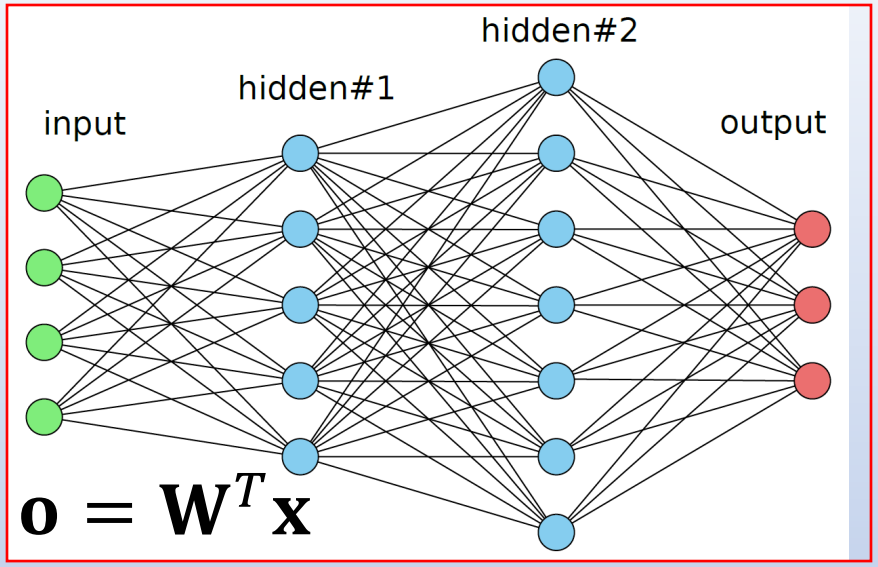
$$n_k = 32\times32 = 1024$$
$$n_{k+1} = 6\times28\times28 = 4704$$

$$\mathbf{x} = \{x_j\} \in \mathbb{R}^{n_k}$$
$$\mathbf{o} = \{o_j\} \in \mathbb{R}^{n_{k+1}},$$
$$\mathbf{W} \in \mathbb{R}^{n_k \times n_{k+1}}$$

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



Compare CNN with MLP in 1D: Network architecture



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

$$o_j^q = g_j^q * x_j, \qquad g_j^q = 0, \ \mathrm{j} < -a, j > a$$

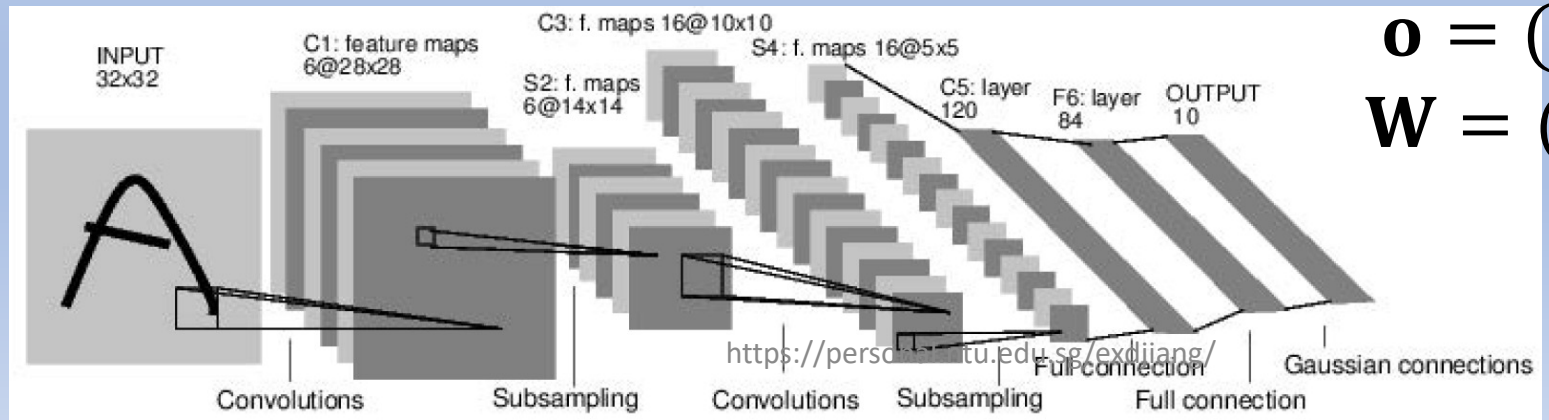
$$= \sum_{i=1}^{n_k} g_{j-i}^q x_i = \mathbf{w}_j^{qT}\mathbf{x}$$

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

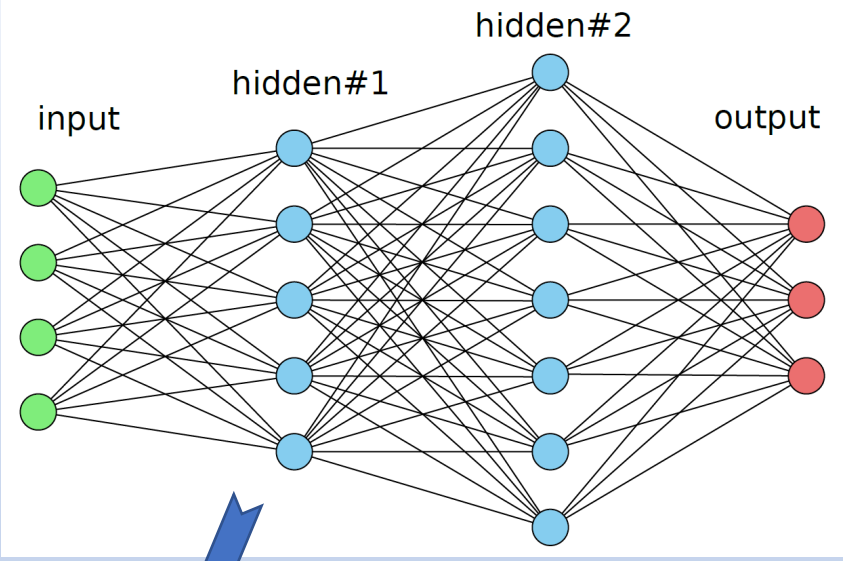
$$\mathbf{o} = (\mathbf{o}^1, \ldots, \mathbf{o}^q \ldots, \mathbf{o}^p)^T$$

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

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



Compare CNN with MLP in 1D: Network architecture



在MLP中，通常每个输入单元会与下一层的每个单元相连，形成一个完全连接的网络。这导致参数数量庞大，尤其是当处理大型图像数据时。
CNN通过使用卷积层减少了参数的数量。在卷积层中，网络学习过滤器（或称为卷积核），这些过滤器在输入数据上滑动以提取特征。每个过滤器只关注输入数据的一小部分，并且同一个过滤器在整个数据上共享，这大大减少了模型的参数数量。

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

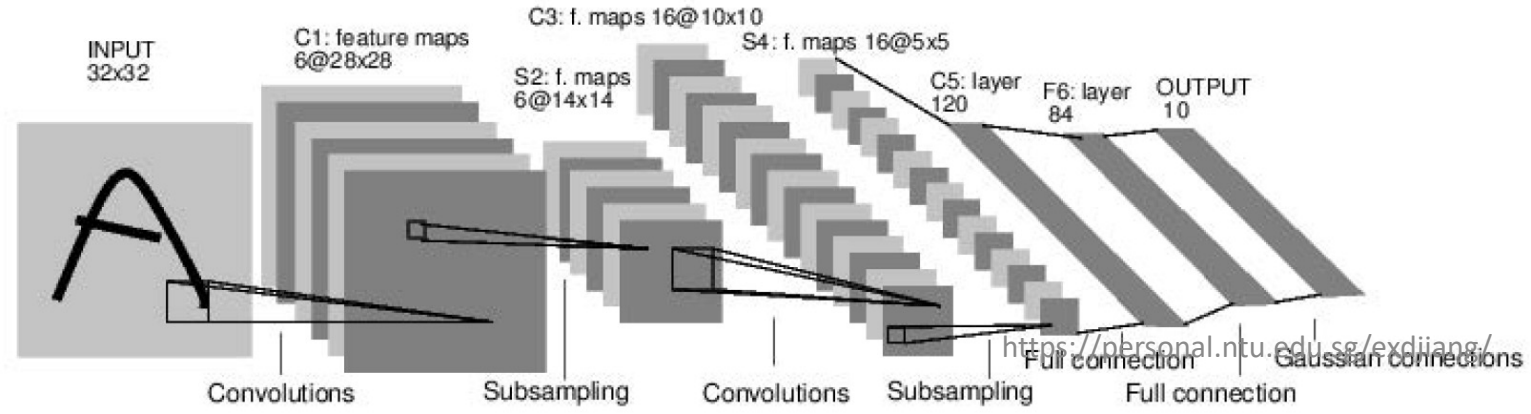
$$= \begin{pmatrix} \mathbf{g}^1 & 0 & 0 & \mathbf{g}^p & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 \ldots & \mathbf{g}^1 \ldots & 0 \ldots\ldots & 0 \ldots & \mathbf{g}^p \ldots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots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

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

CNN is a simplified MLP

CNN is a regularized MLP

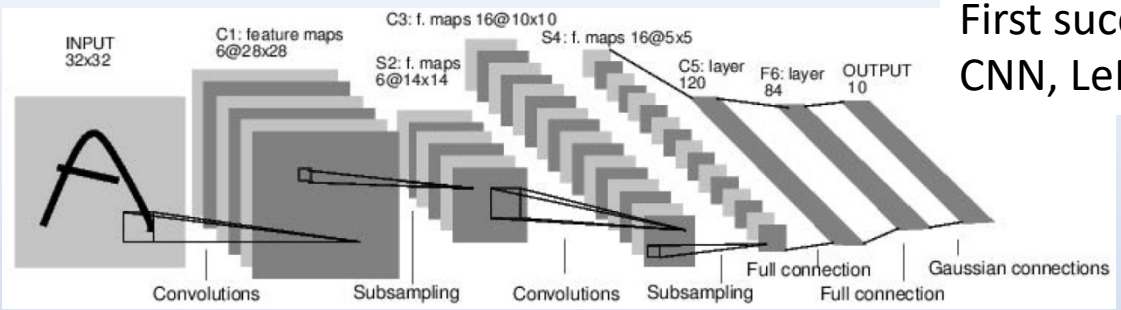
MLP由于其完全连接的性质，容易出现过拟合，即模型过于复杂，学到了训练数据中的噪声而非潜在的数据分布。
CNN的结构通过局部连接和权重共享来引入正则化效果。这意味着网络被迫学习更具有代表性的特征，因为每个权重必须在多个位置的数据中工作，从而减少了模型在训练数据上的过拟合风险。



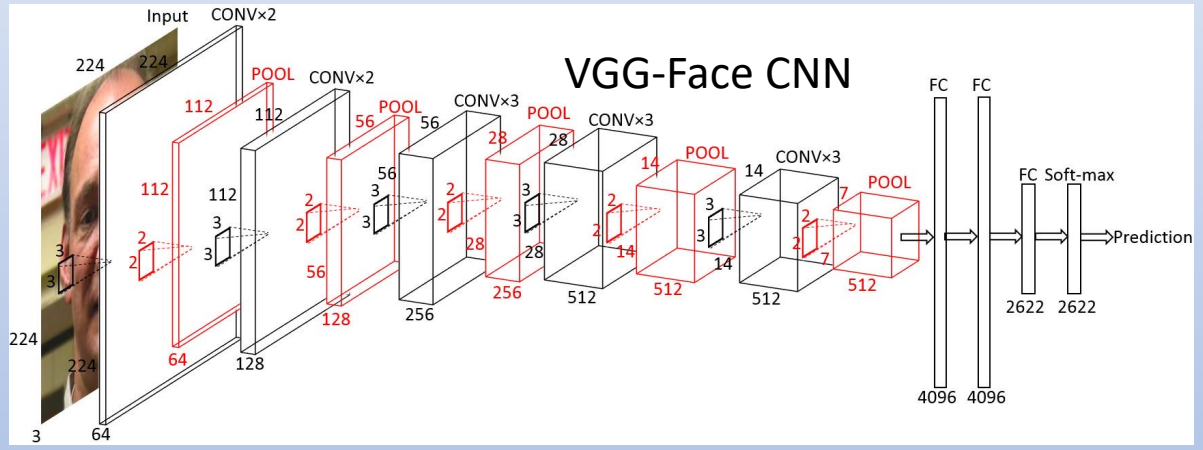
Simplification/regularization extracts less amount of discriminative information, Solve over-fitting problem.

Not any arbitrary implification can solve over-fitting problem.

Merits of convolutional network, CNN



First successful CNN, LeNet5



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

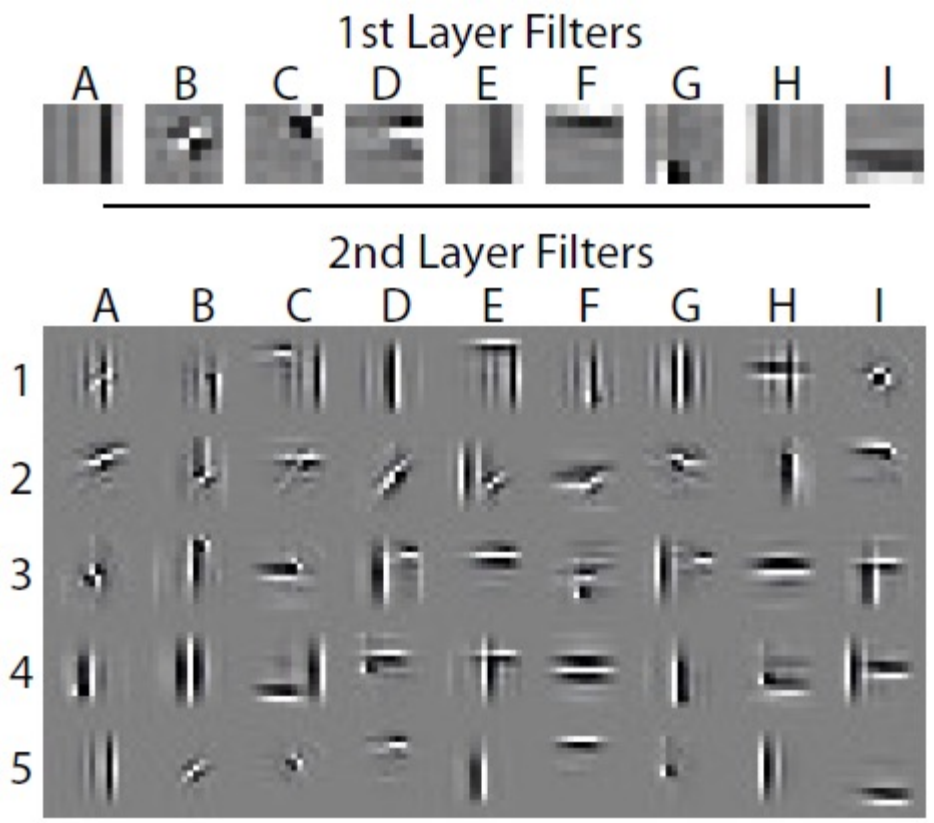
$$o_j^q = g_j^q * x_j$$

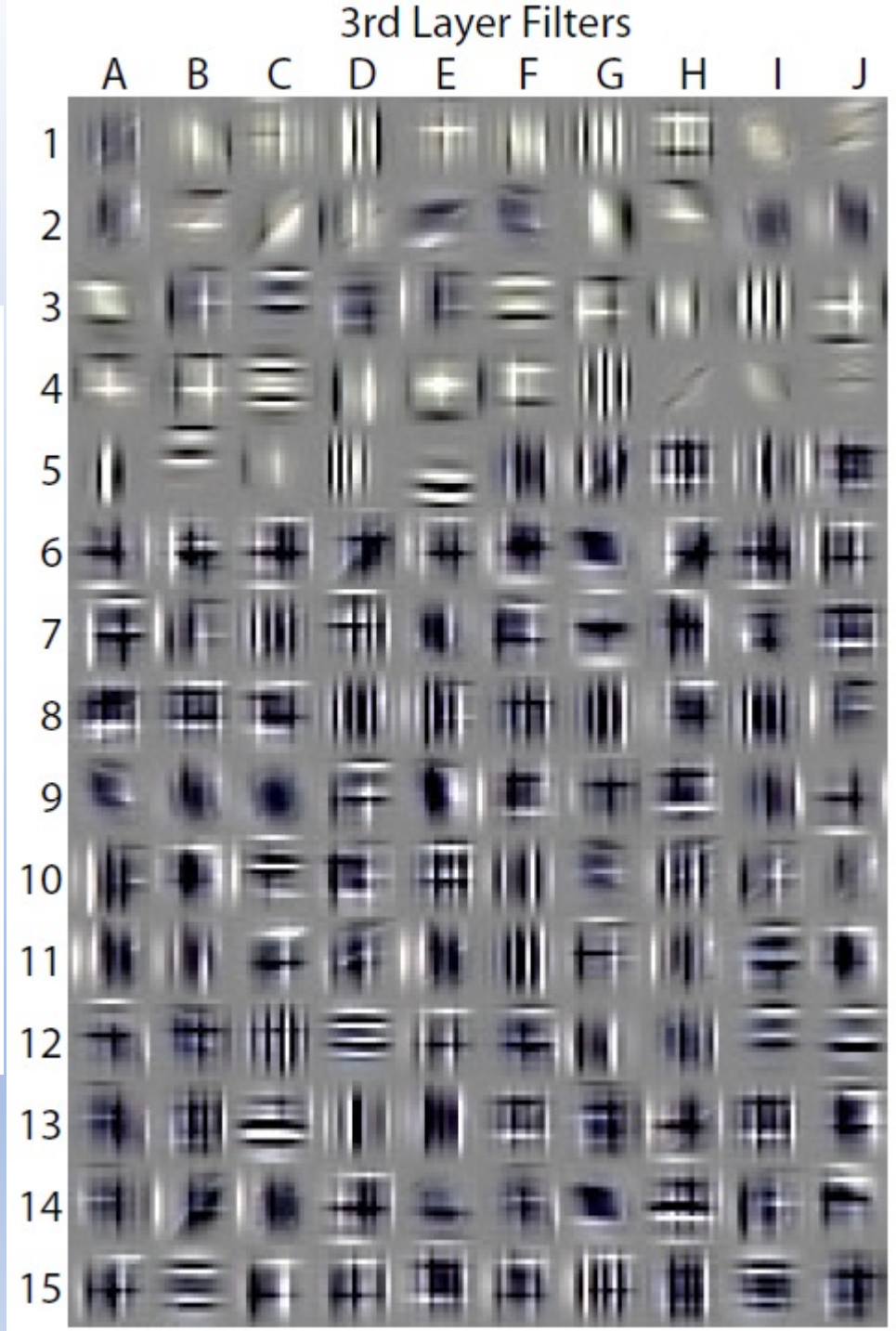
$$= \sum_{i=1}^{n_k} g_{j-i}^q x_i$$

$$= \mathbf{w}_j^{qT} \mathbf{x}$$

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.

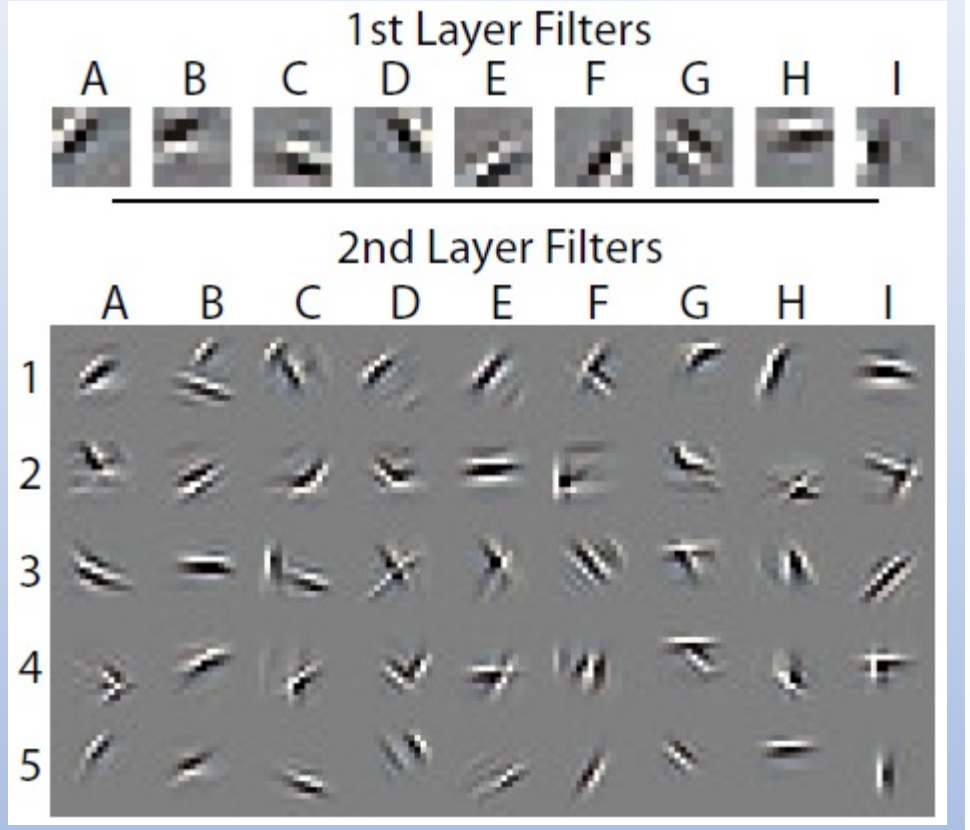
Examples:

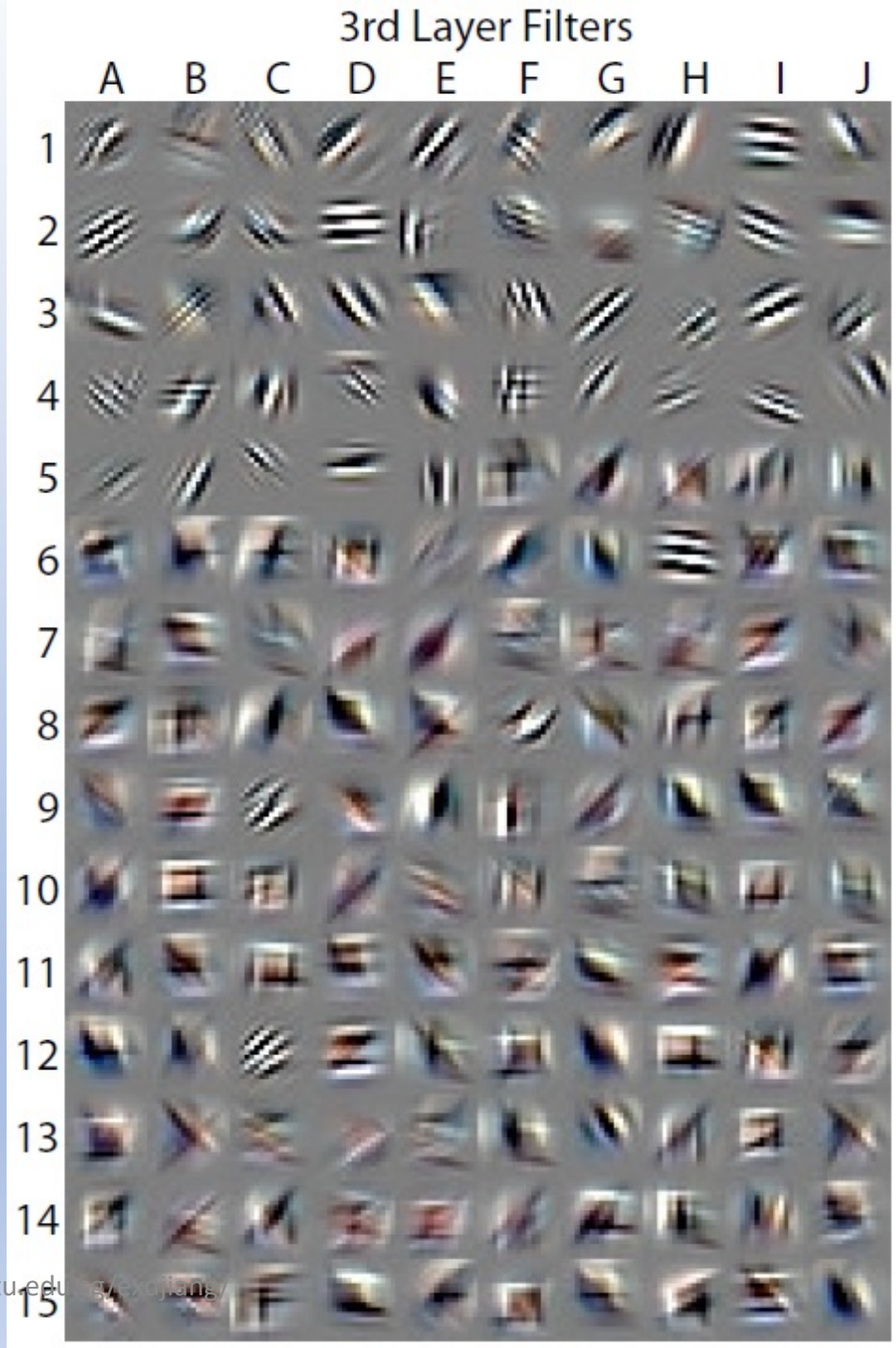




Filters trained on the city dataset. Note the predominance of horizontal and vertical structures.

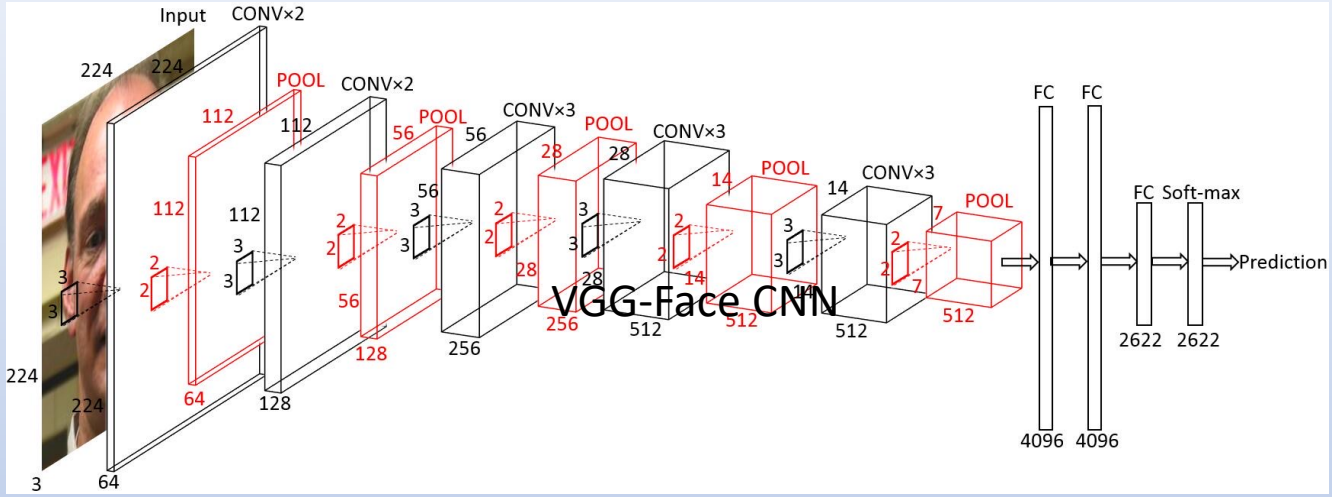
Further Examples:





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



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

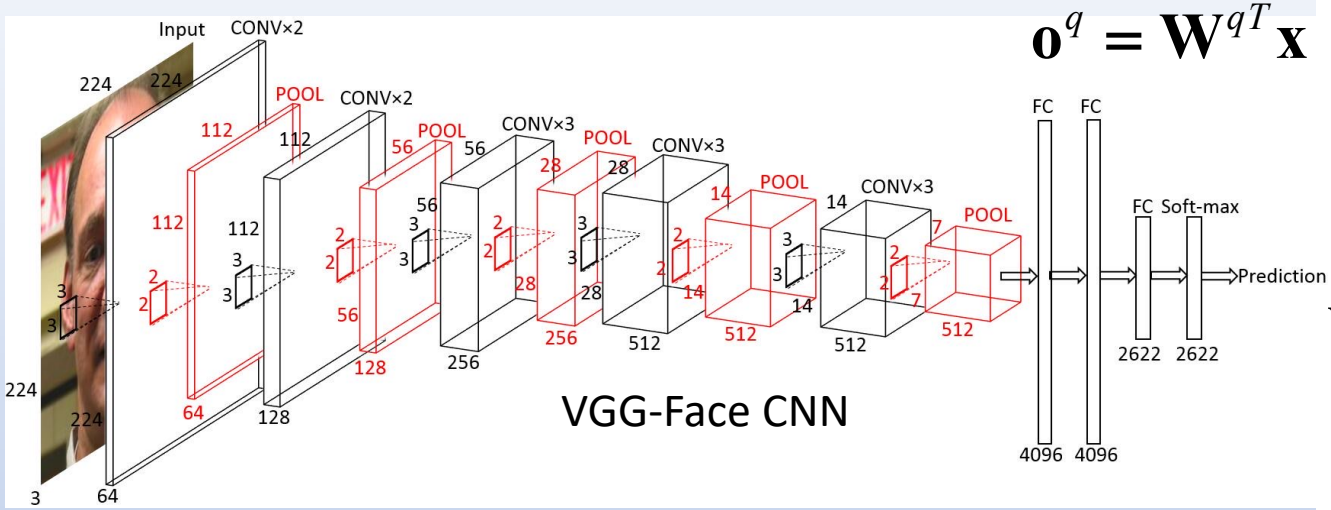
$$\mathbf{o}^q = \mathbf{W}^{qT}\mathbf{x}$$

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.

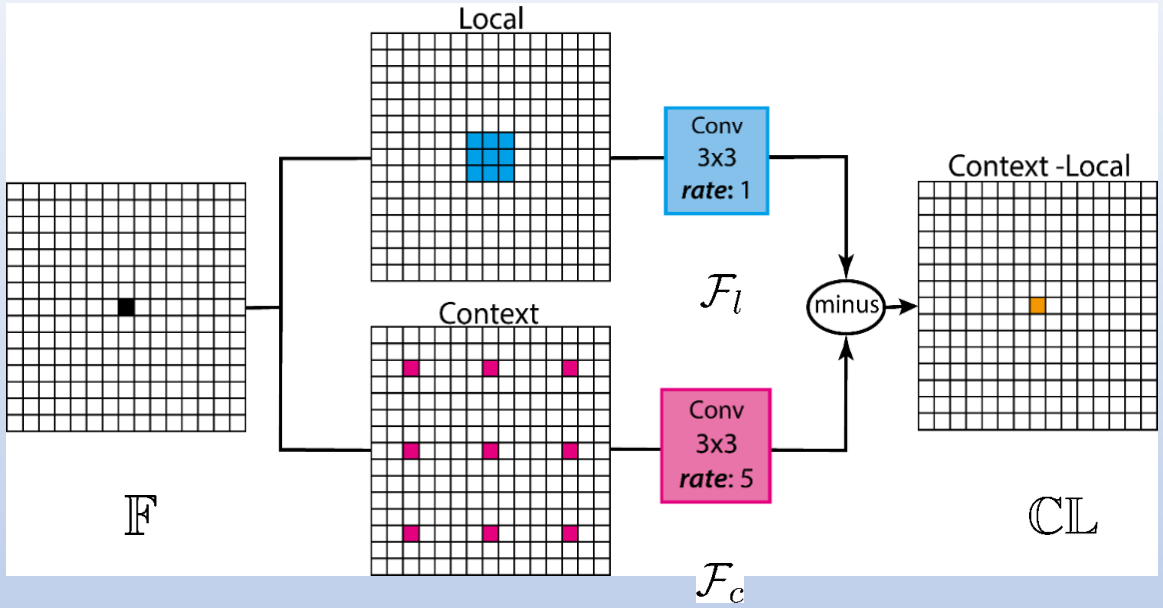
9. Simple nonlinear activation function ReLu leads to efficient and effective training.

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.**

Topic 15

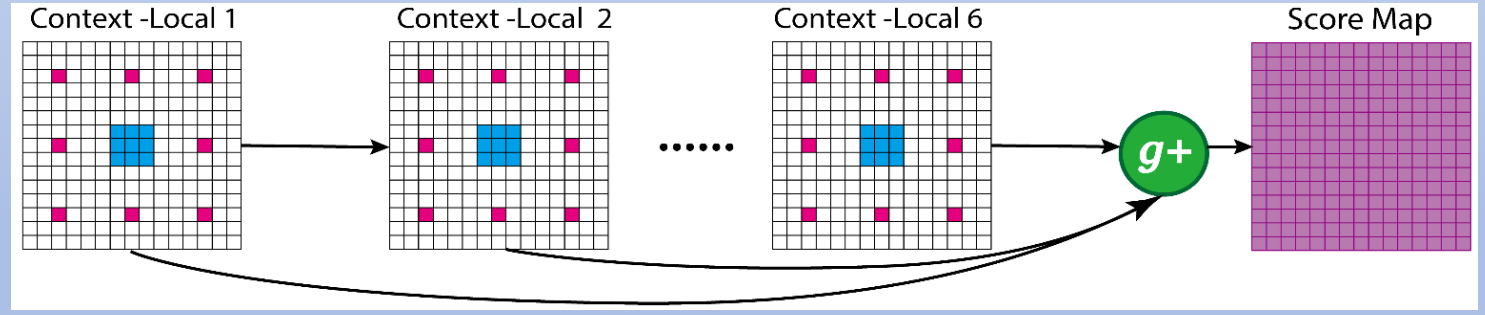
Deep Learning: from CNN to Transformer

Context Contrasted Local Features



Context Contrasted Local Features:
The context contrasted local features are obtained via making a contrast between the local information and its context (surroundings).

$$\mathbb{CL} = \mathcal{F}_l(\mathbb{F}, \Theta_l) - \mathcal{F}_c(\mathbb{F}, \Theta_c)$$



Context Contrasted Local (CCL) Model:
Several blocks are chained to make multi-level context contrasted local features.

Understand Transformer

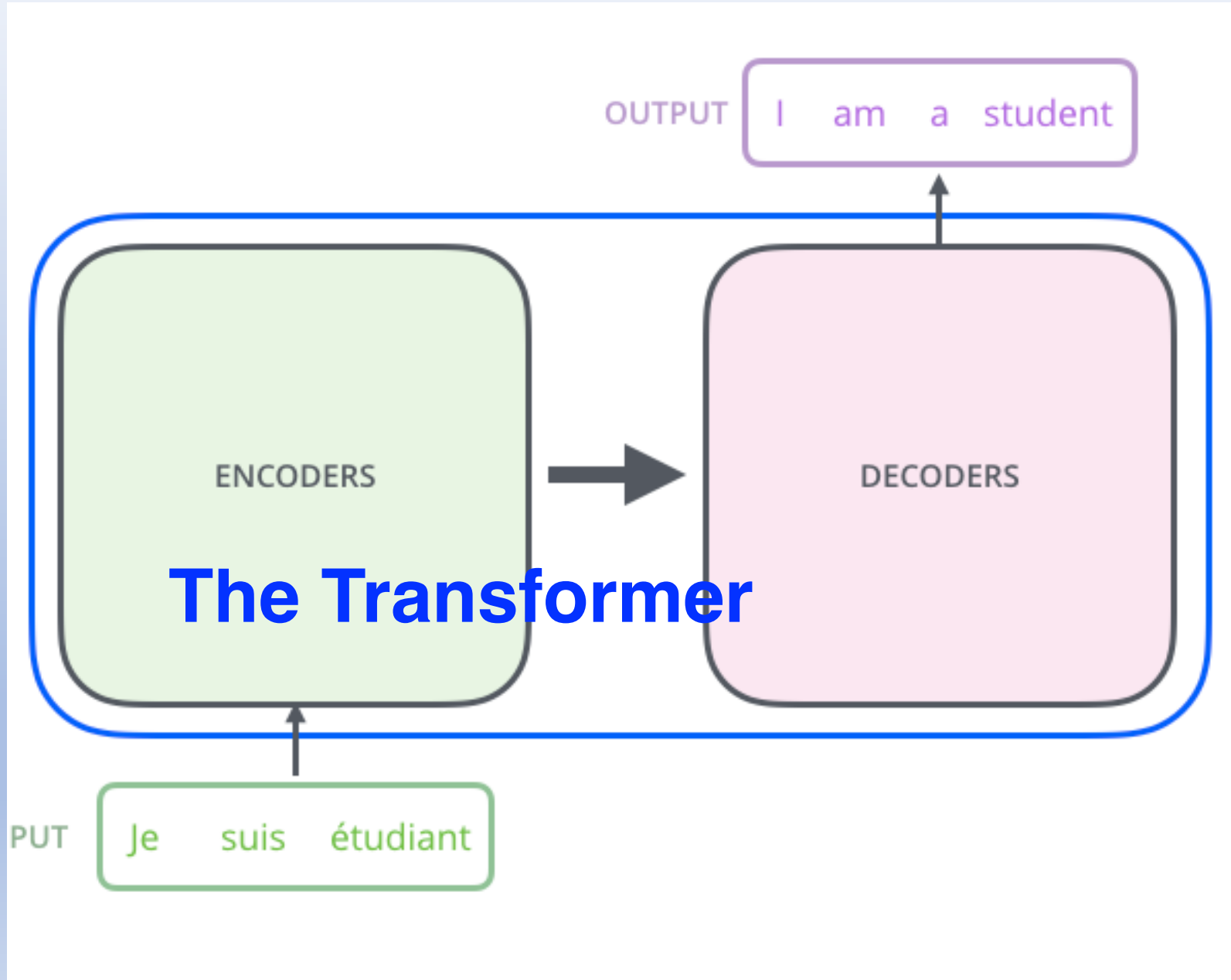
- The Transformer has a neural network architecture that transforms an input sequence to an output sequence such as speech, text and time series.
- Recurrent neural networks (RNNs) and its further development, Long-Short Term Memory (LSTM), are clearly related to sequences and lists. Transformer outperforms them by parallelization of their sequential operations.
- It is developed originally for natural language processing (NLP). Now it becomes a powerful neural network architecture also for computer vision, showing more powerful than CNN.
- The Transformer gets its powers thanks to its Attention module. It captures the relationships between each word/token in a sequence with every other word/token. The original paper of transformer: ”Attention Is All You Need”.
- how does the transformer work? Is attention really everything? Any relation to CNN?

Transformer consists of Encoders and Decoders

A machine translation application, it takes a sentence in one language, and outputs its translation in another.

All words/tokens of a sentence/image are parallelly inputted into the transformer.

Output is a linear classification of the decoder output.

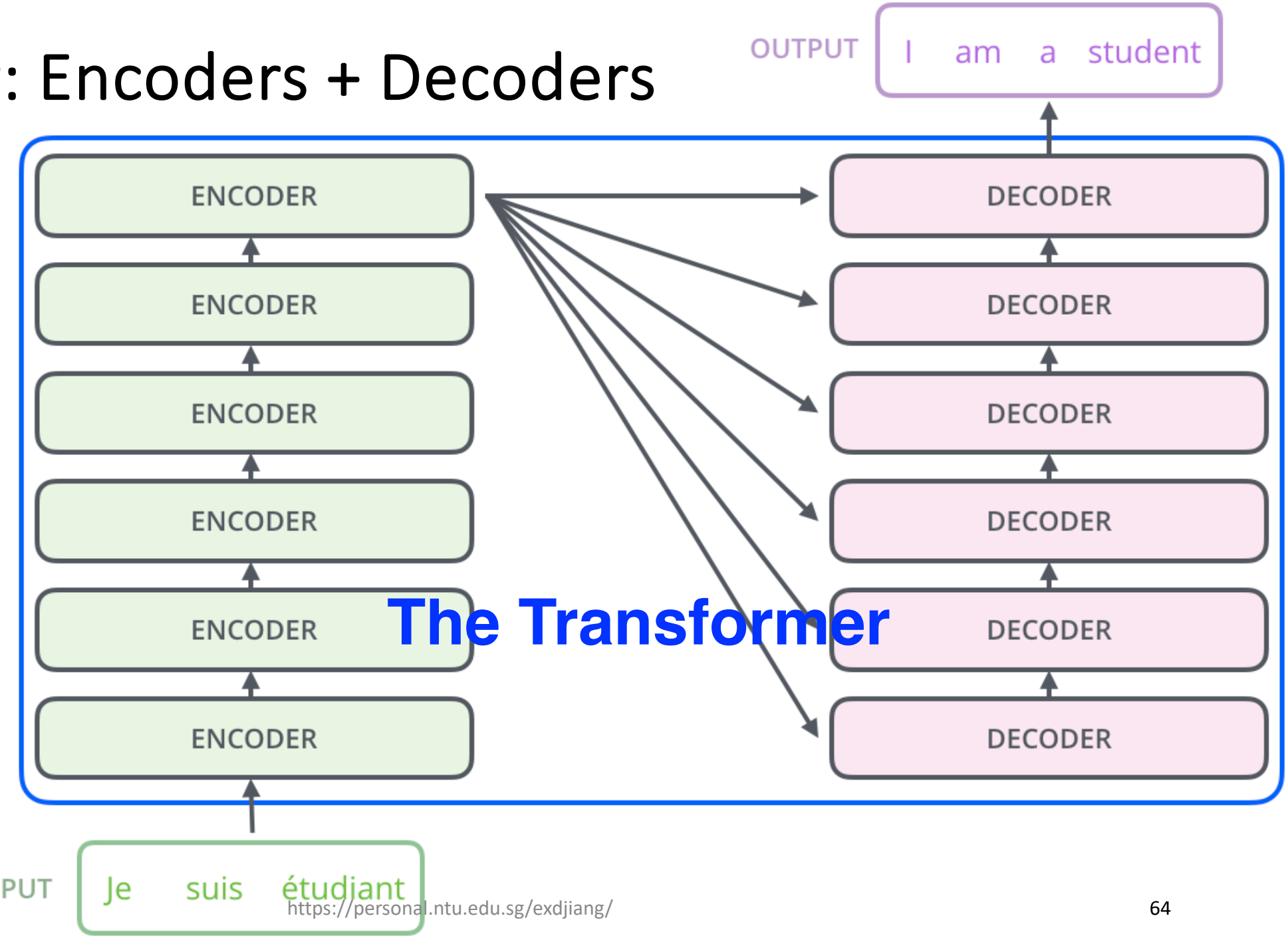


Transformer: Encoders + Decoders

A stack of encoders of the same structure

and

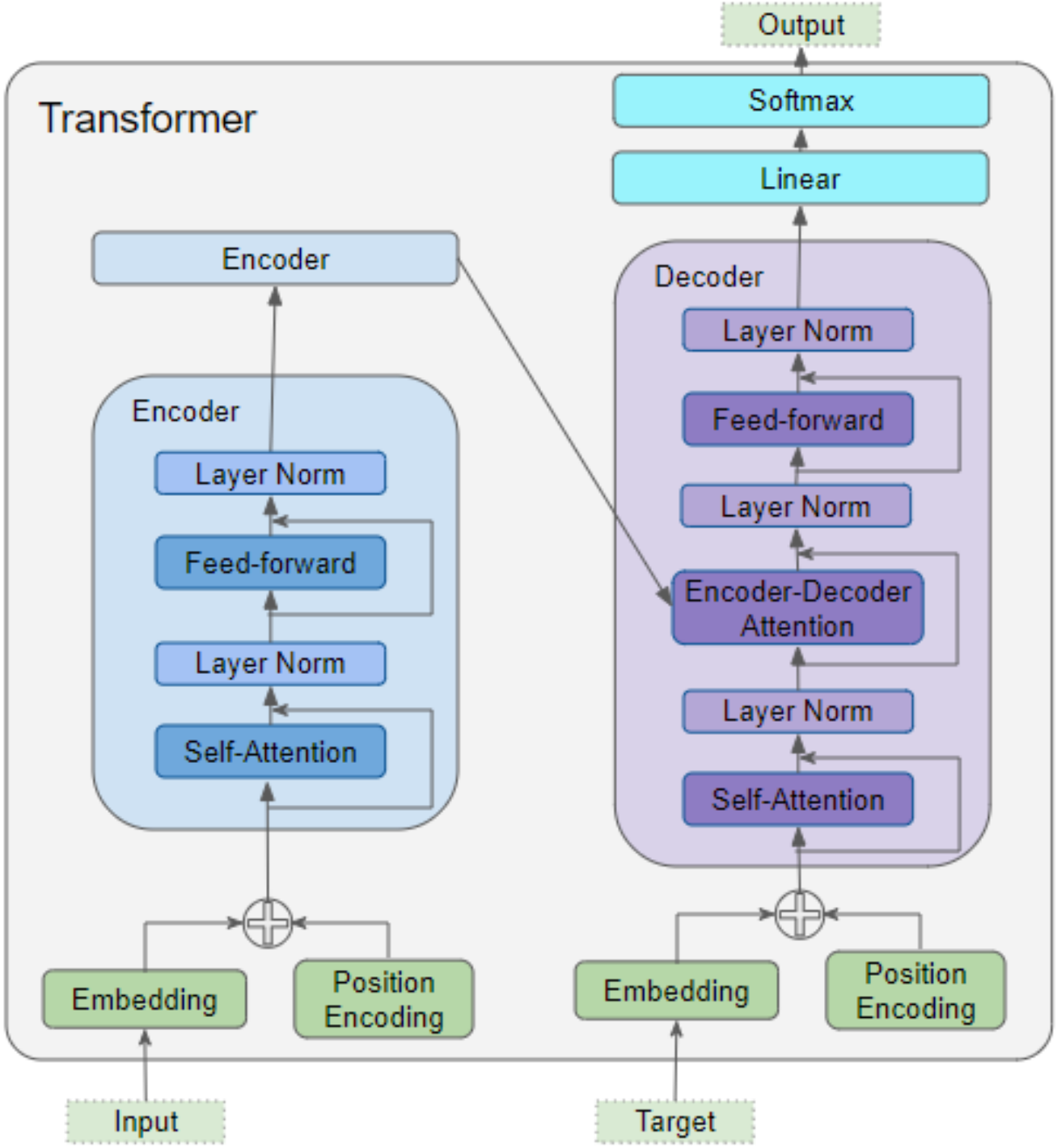
a stack of decoders of the same structure



Transformer:
Encoders + Decoders

Structures of an Encoder and a Decoder

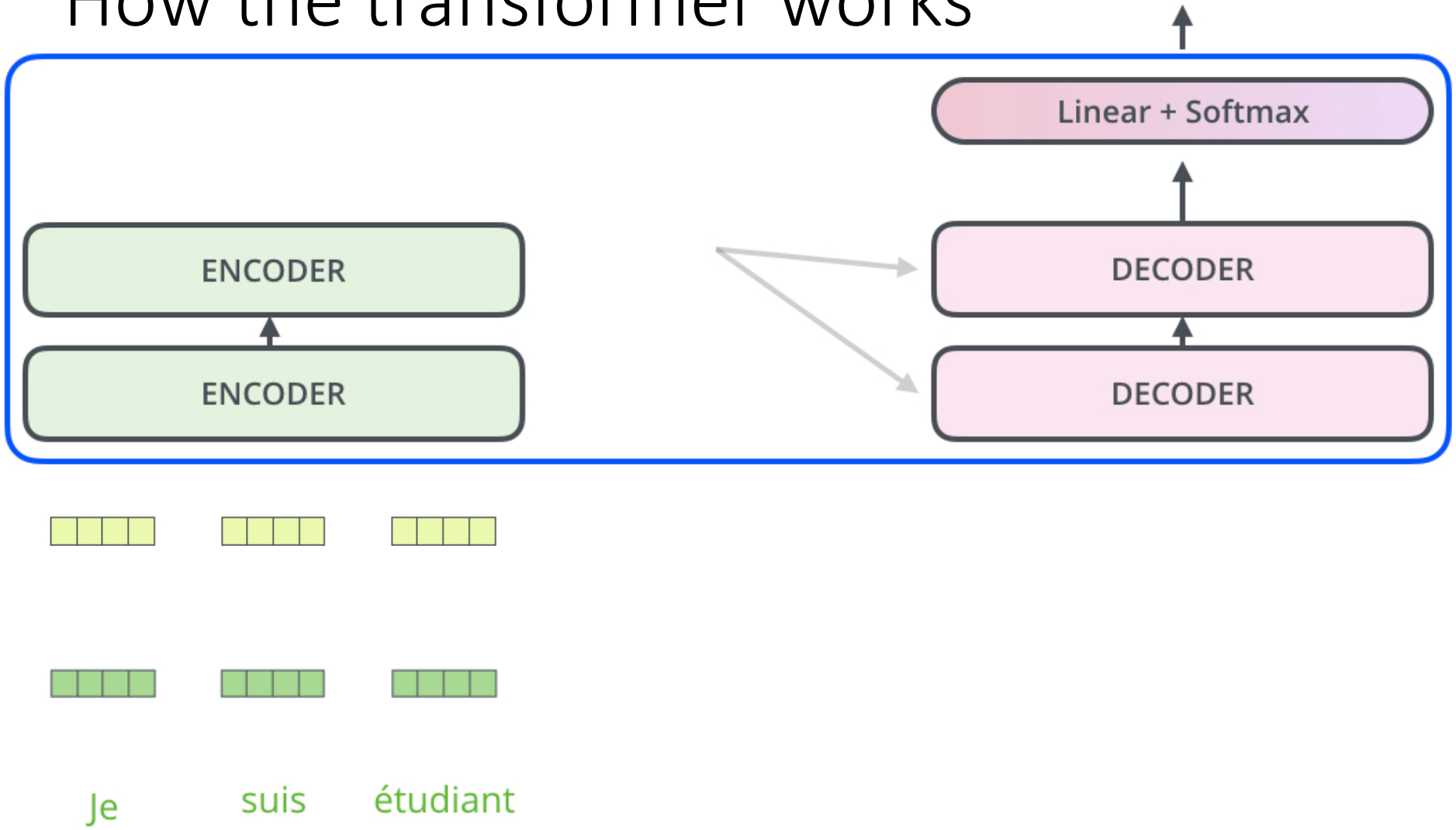
Commonality and Difference of encoder and decoder

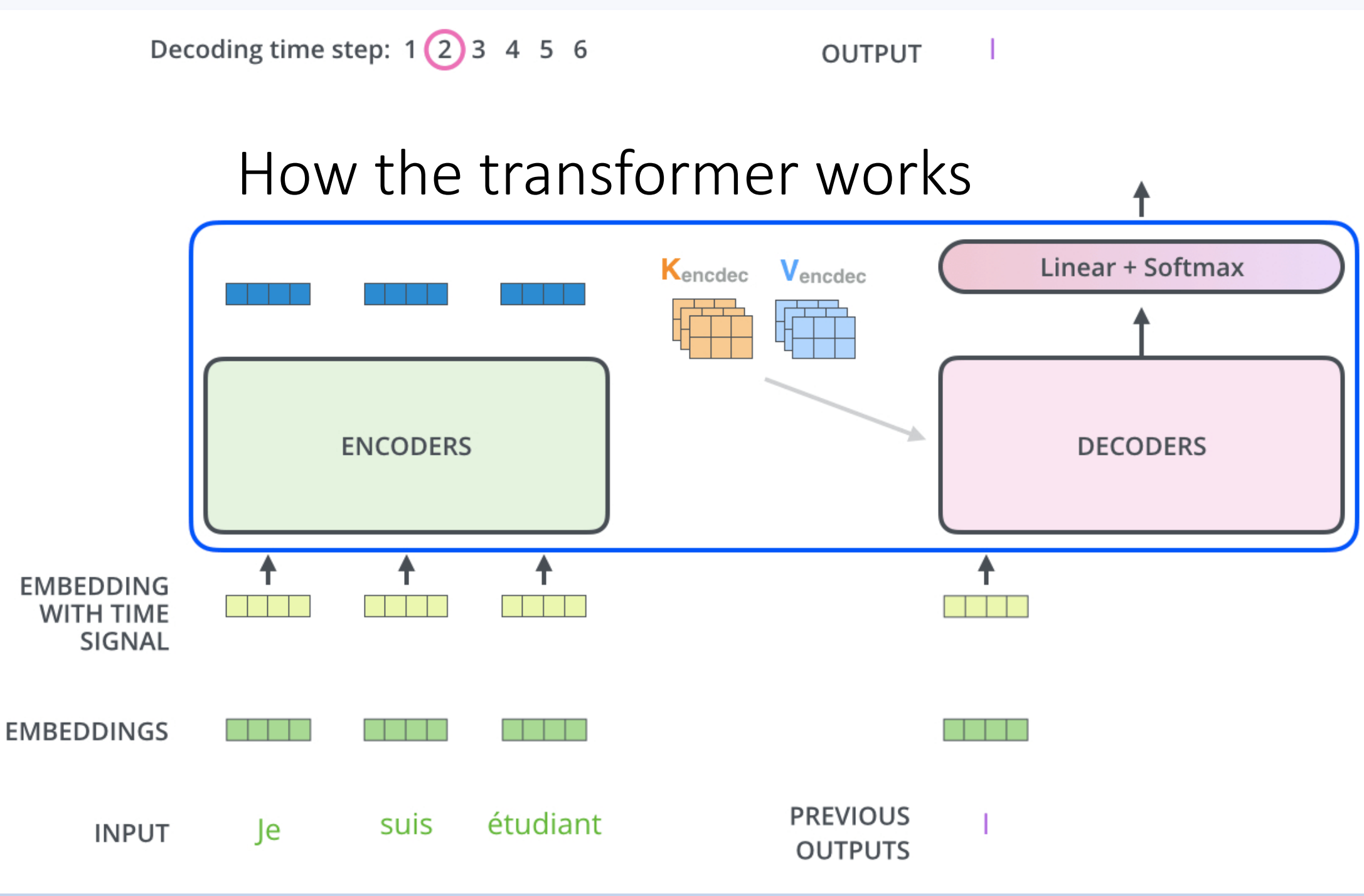


Decoding time step: 1 2 3 4 5 6

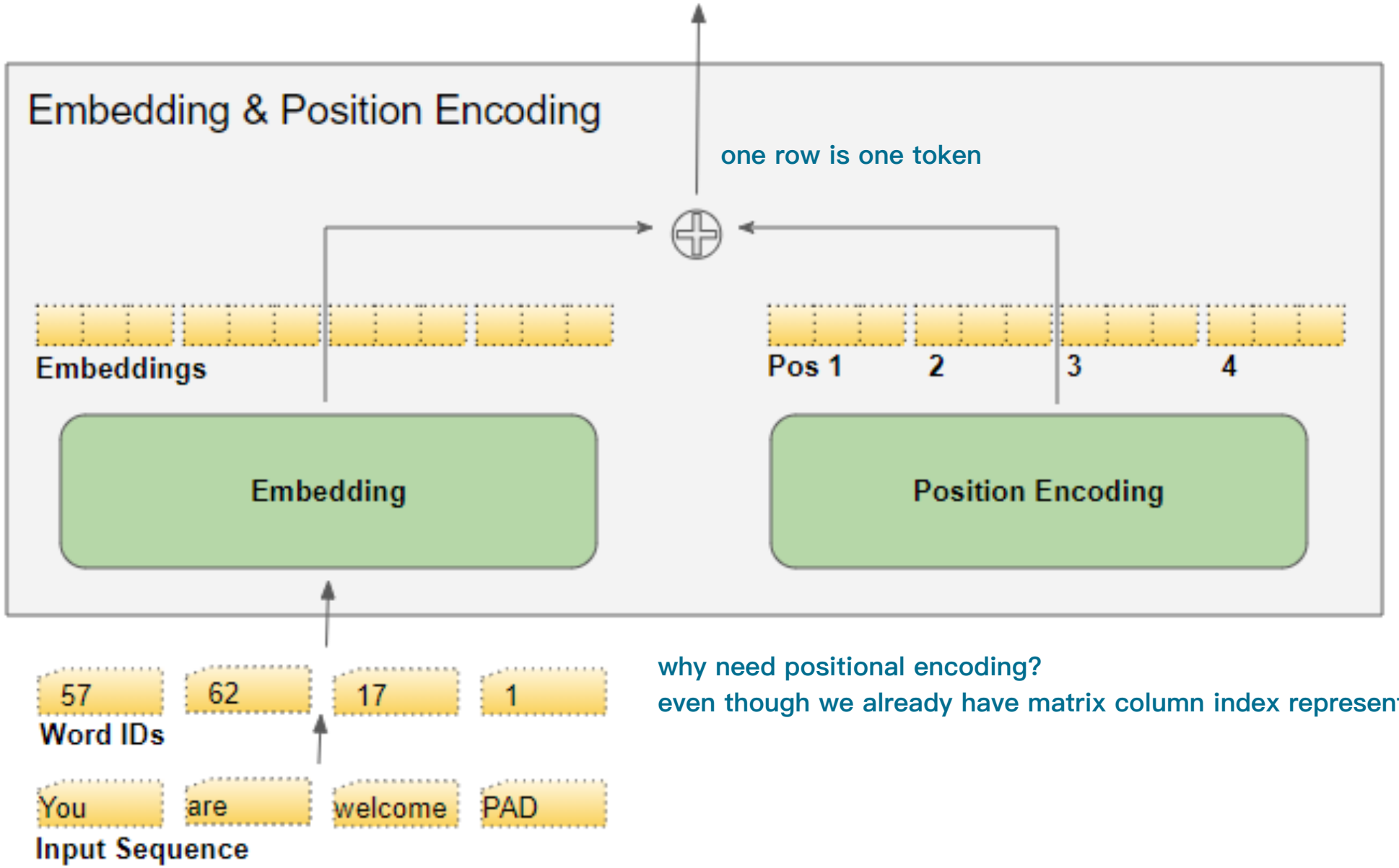
OUTPUT

How the transformer works




Decoding time step: 1 2 3 4 5 6
OUTPUT
I
How the transformer works
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Embedding each input word/token into a feature vector

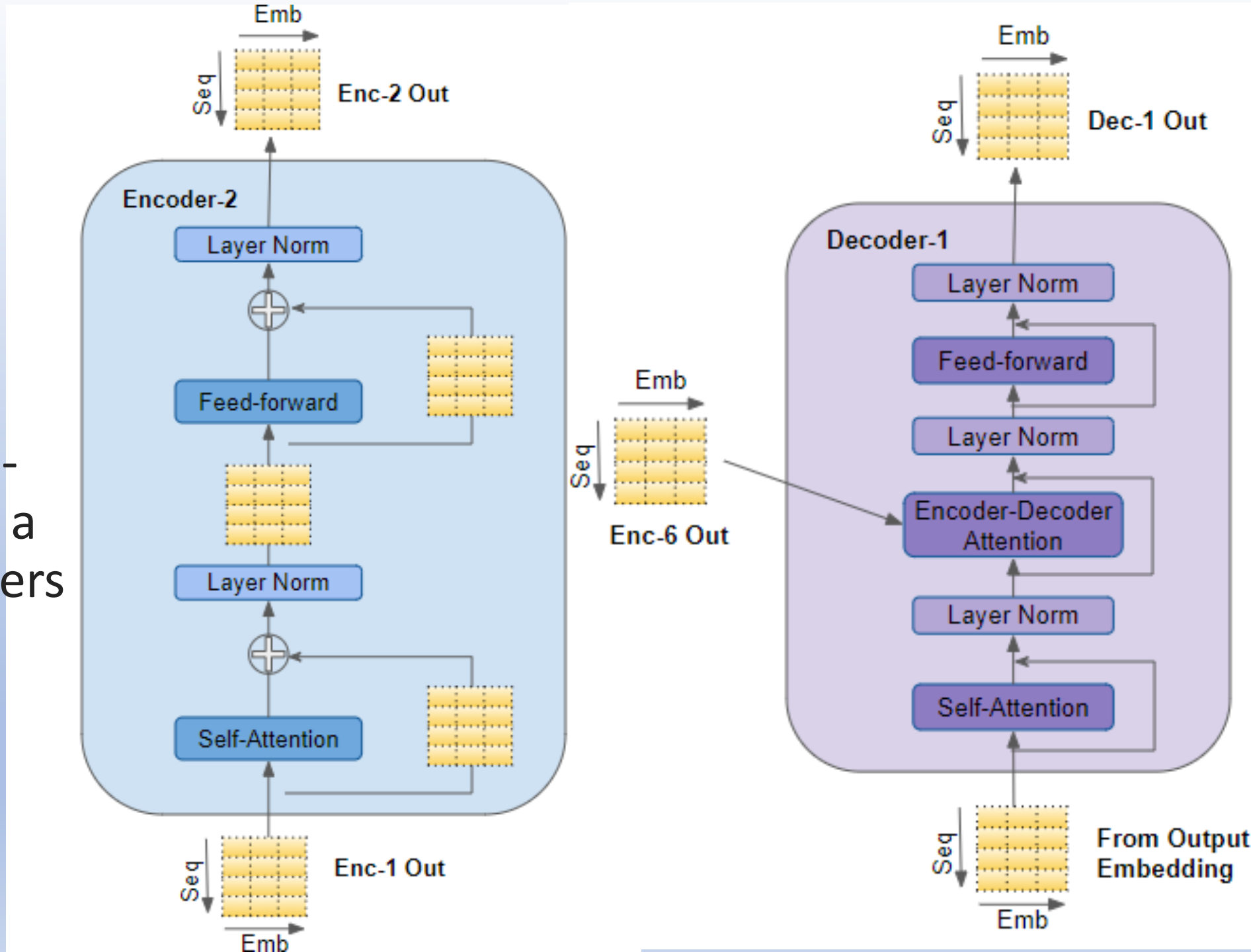


why need positional encoding?
even though we already have matrix column index representing positions

Both the Attention and Feed-forward layers, have a residual skip-connection.

The pointwise feed-forward network is a couple of linear layers with a ReLU activation in between.

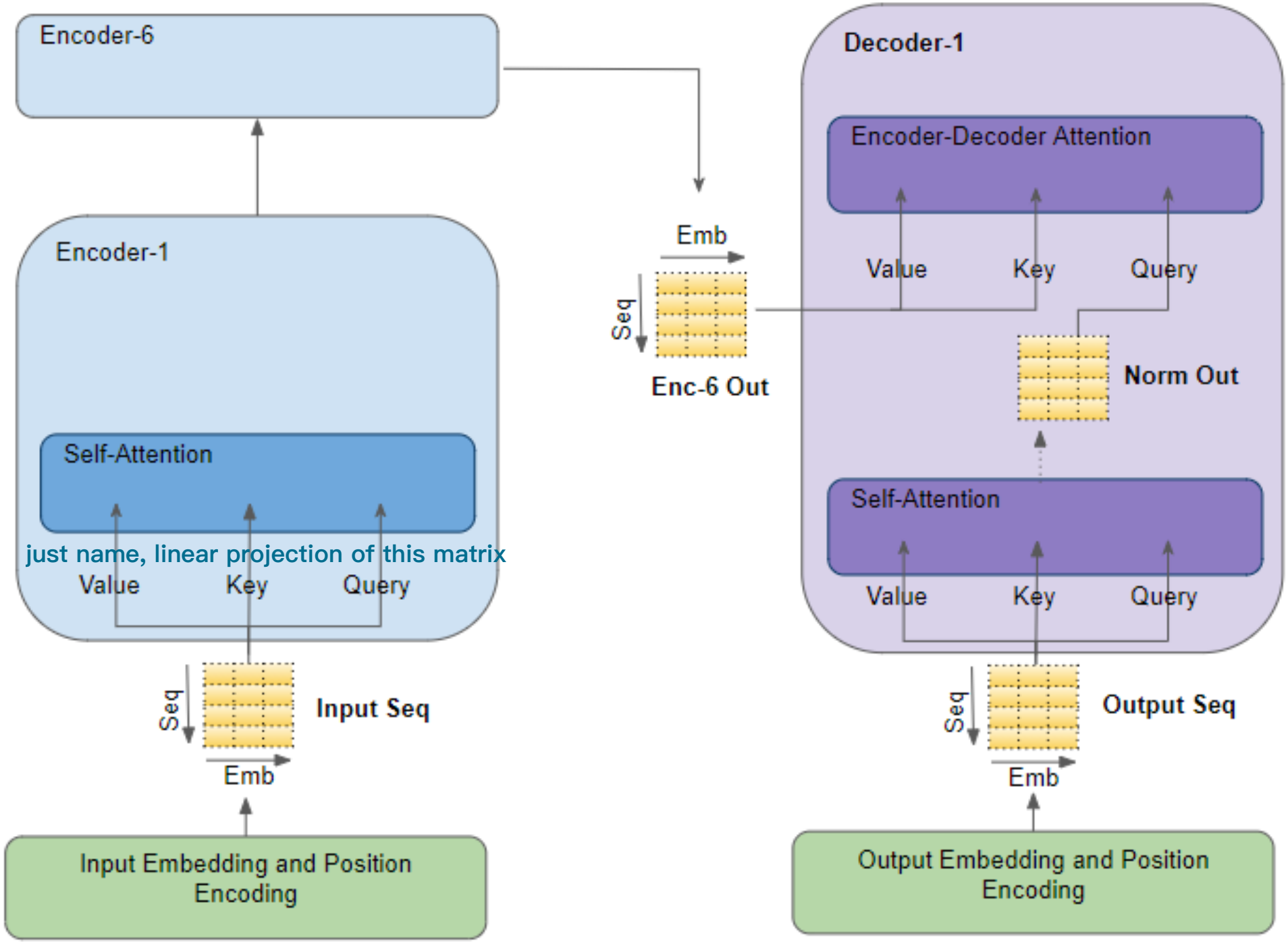
Pointwise?



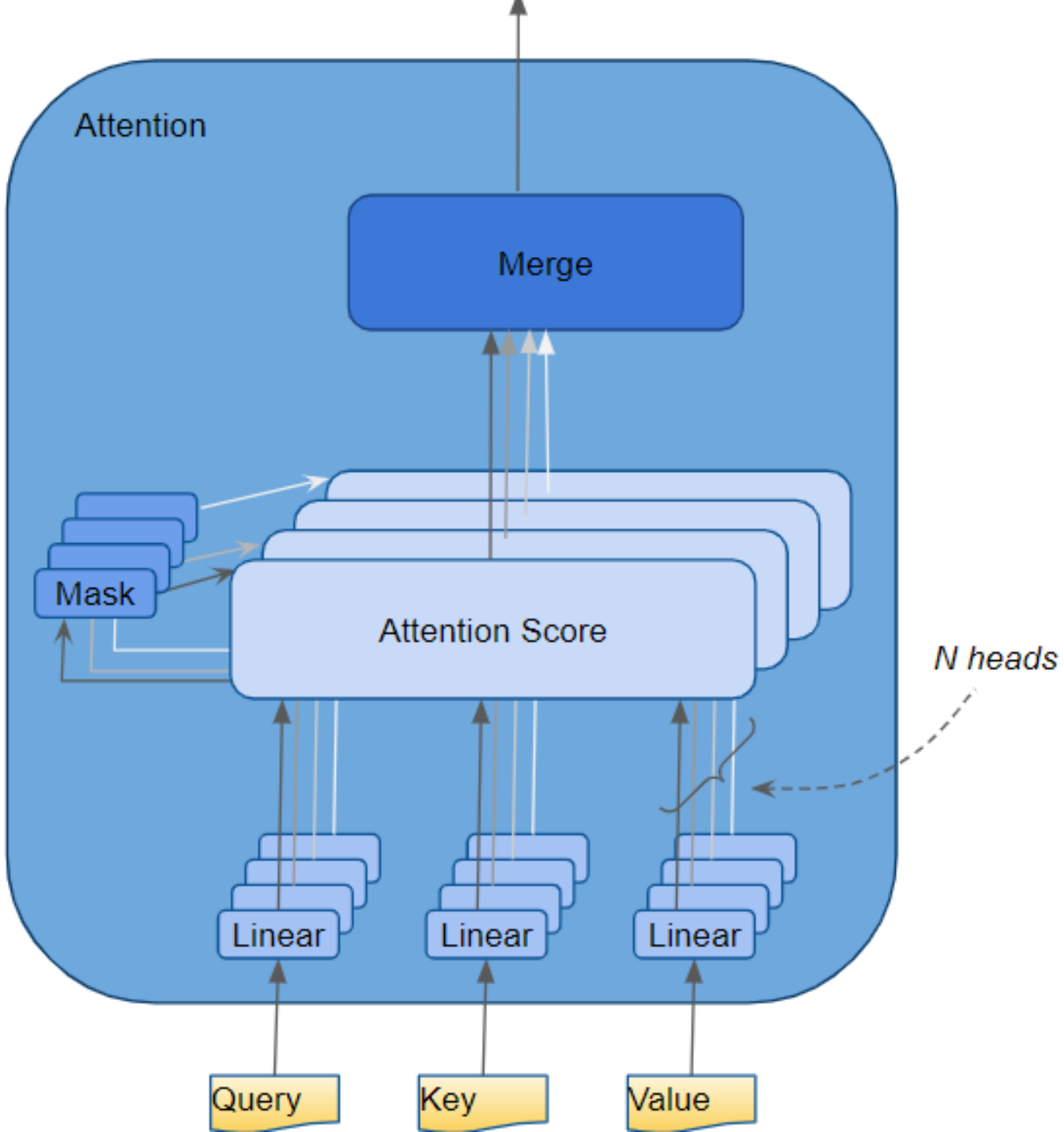
Self-attention

and

Encoder-decoder attention



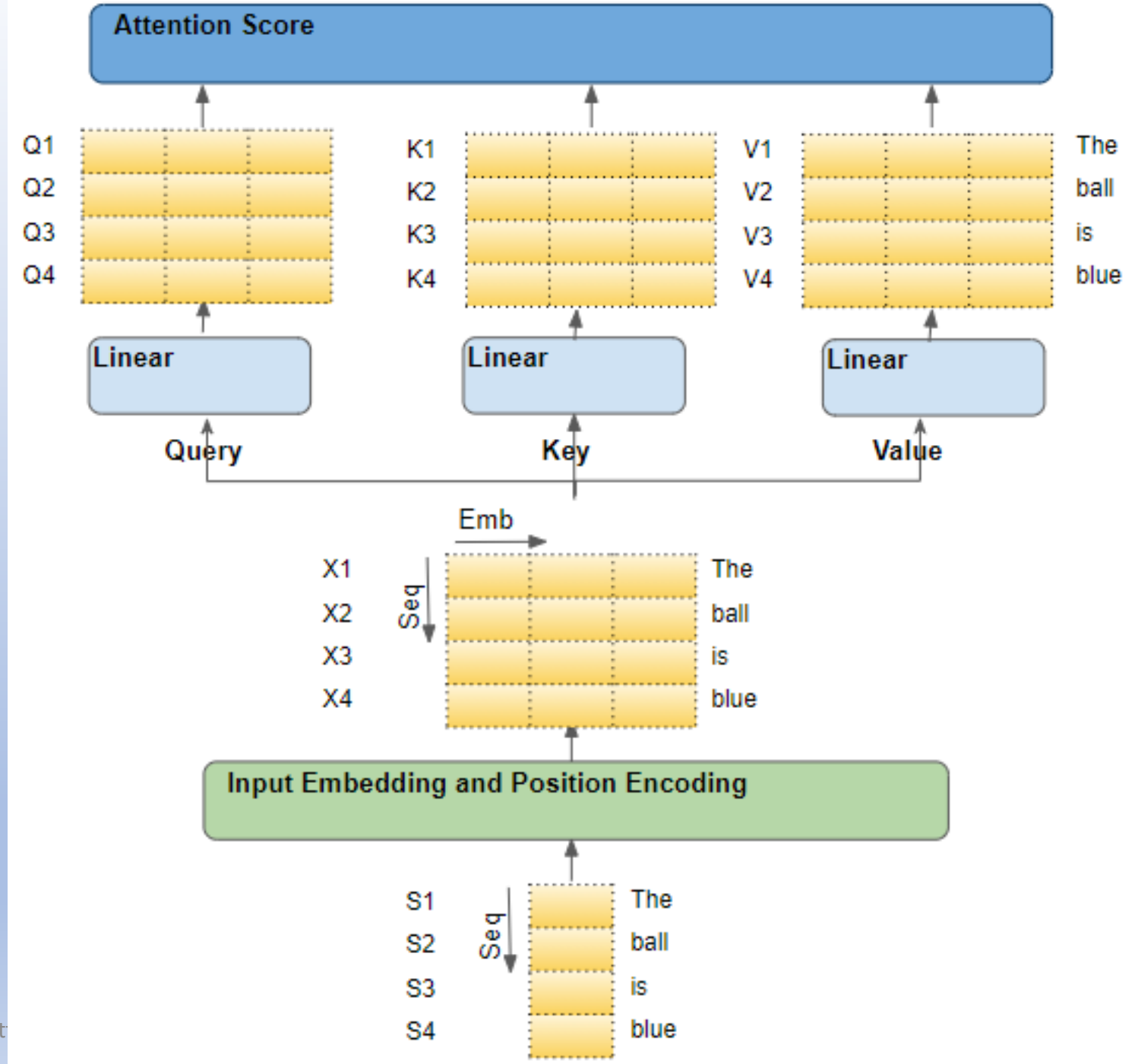
Multiple attention heads in each encoder and decoder.



Prepare for Attention

An example of English-to-Spanish translation problem, where one sample source sequence is “The ball is blue”. The target sequence is “La bola es azul”.

Three copies of each word/token are generated for self-attention by linear projection.



Learnable linear Projections:

The input sequence (a matrix) is passed through three trainable linear layers which produce three separate matrices — known as the Query, Key, and Value.

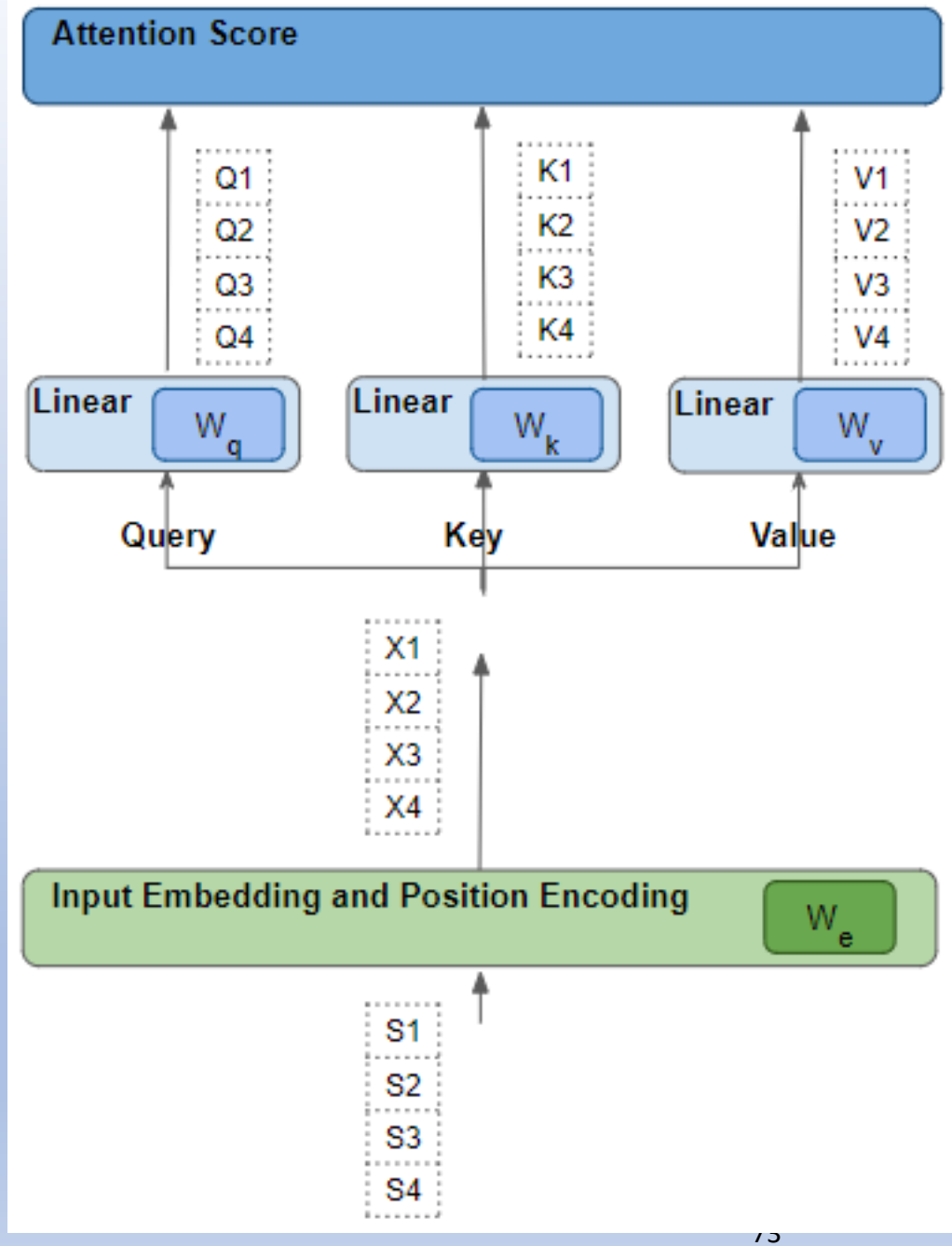
Q1 only related to X1, have nothing to do othersX

$$\text{Let } \mathbf{X} = \begin{pmatrix} X1 \\ X2 \\ X3 \\ X4 \end{pmatrix}, \mathbf{Q} = \begin{pmatrix} Q1 \\ Q2 \\ Q3 \\ Q4 \end{pmatrix}, \mathbf{K} = \begin{pmatrix} K1 \\ K2 \\ K3 \\ K4 \end{pmatrix}, \mathbf{V} = \begin{pmatrix} V1 \\ V2 \\ V3 \\ V4 \end{pmatrix}$$

$$\text{Then: } \mathbf{Q} = \mathbf{X}\boldsymbol{W}_q, \ \mathbf{K} = \mathbf{X}\boldsymbol{W}_k, \ \mathbf{V} = \mathbf{X}\boldsymbol{W}_v$$

treat Wq as a convolutional filter, looping all X

The important thing to keep in mind is that each 'row' of these matrices corresponds to one word (token) in the source sequence.



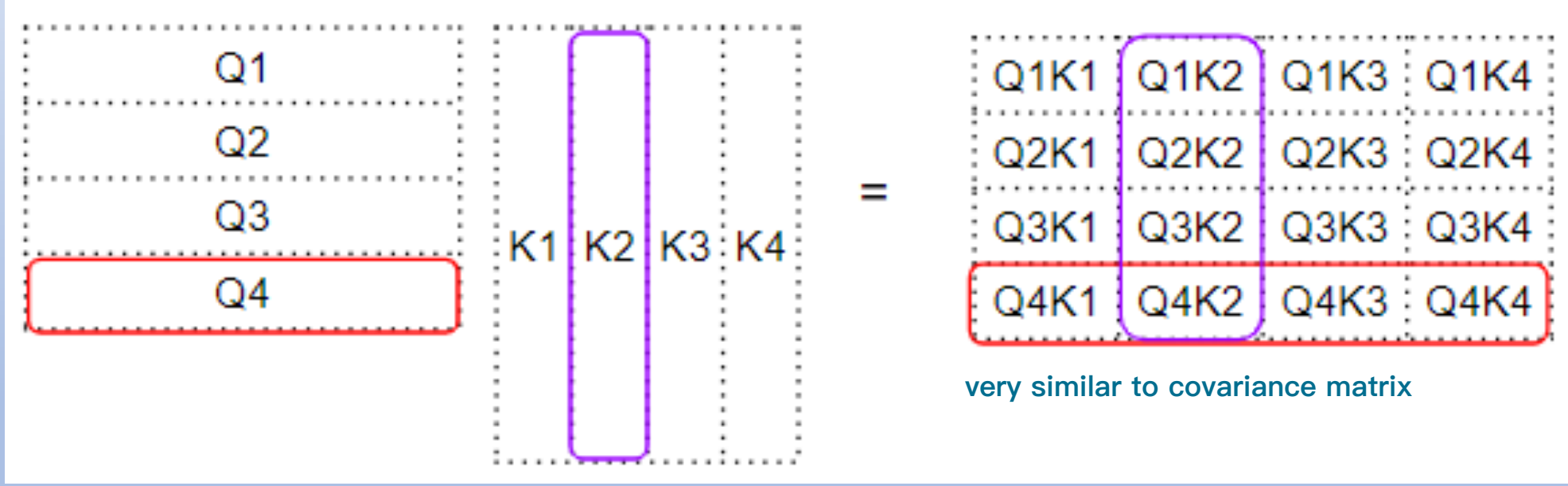
Attention Score — Dot Product between Q and K words

- The first step of Attention is to do a matrix multiply (ie. dot product) between the Query (Q) matrix and a transpose of the Key (K) matrix.

$$R = QK^T$$

- Watch what happens to each word. **Dot product generates similarity between words**

$R = QK^T$



- We produce an intermediate matrix (let's call it a 'factor' matrix) where each cell is a matrix multiplication between two words. **Covariance Matrix!!!**

Attention: Attended (weighted) combination of Values

The dot product between the Query and Key computes the relevance between each pair of words. This relevance is then used as a “factor” to compute a weighted sum of all the Value words. That weighted sum is output as the Attention module.

Q1K1	Q1K2	Q1K3	Q1K4
Q2K1	Q2K2	Q2K3	Q2K4
Q3K1	Q3K2	Q3K3	Q3K4
Q4K1	Q4K2	Q4K3	Q4K4

V1
V2
V3
V4

=

Q1K1V1 + Q1K2V2 + Q1K3V3 + Q1K4V4
Q2K1V1 + Q2K2V2 + Q2K3V3 + Q2K4V4
Q3K1V1 + Q3K2V2 + Q3K3V3 + Q3K4V4
Q4K1V1 + Q4K2V2 + Q4K3V3 + Q4K4V4

$$Z = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$ =

Z1
Z2
Z3
Z4

Fourth word Score

*Fourth Query word * first Key word*

Q4K: how the 4th token related to all tokens

$$Z_4 = (Q_4K_1)\,V_1 + (Q_4K_2)\,V_2 + (Q_4K_3)\,V_3 + (Q_4K_4)\,V_4$$

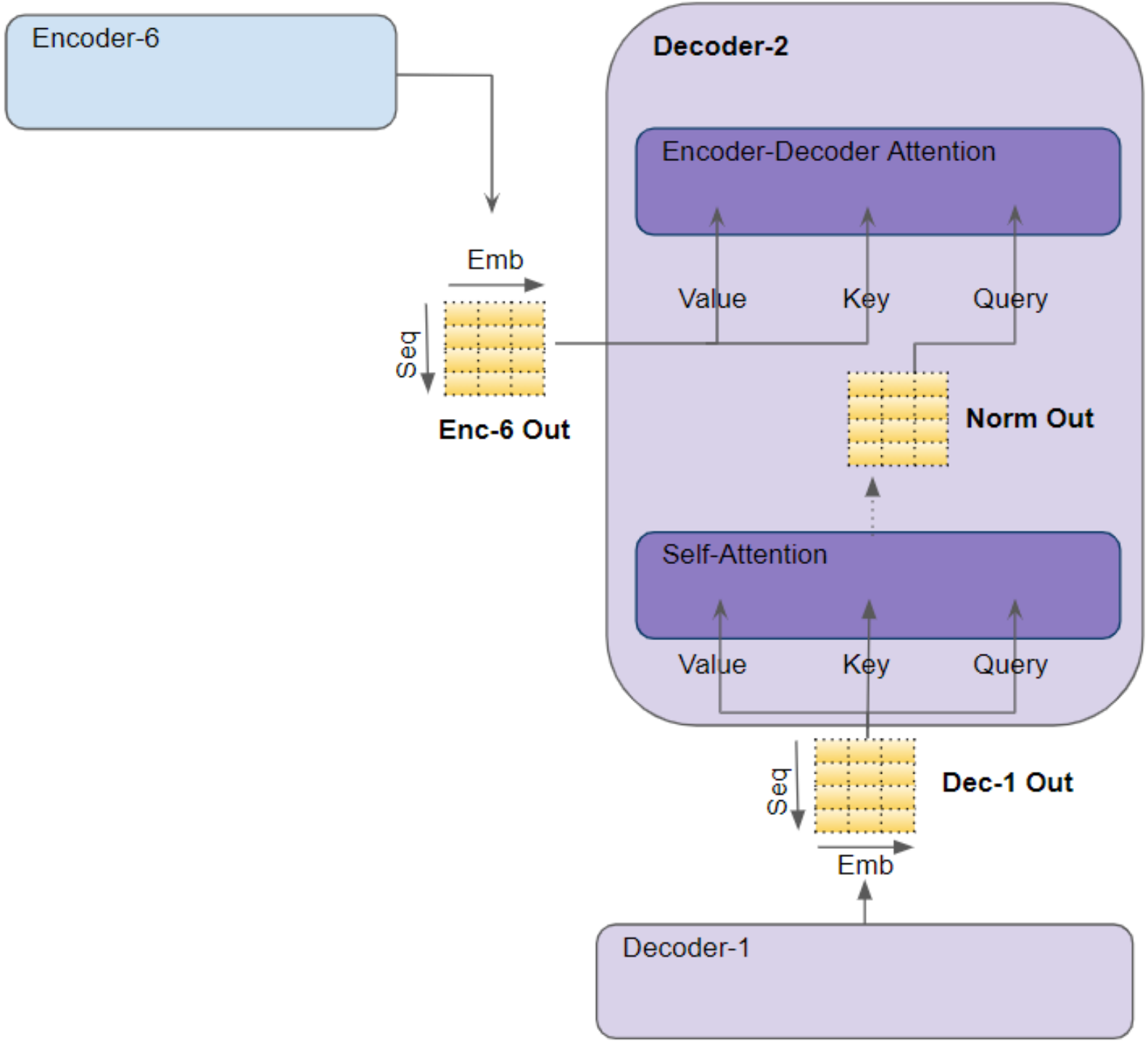
Q4K4 will mostly have the largest weight

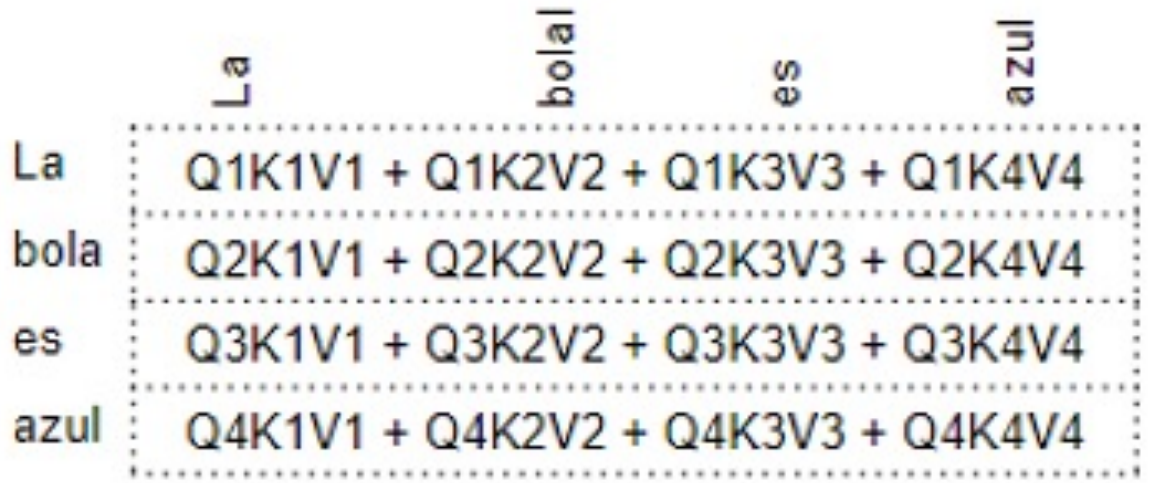
Self-attention in Encoder — the source sequence pays attention to itself.

*Fourth Query word * second Key word*

Encoder-Decoder attention

In the Decoder's Encoder-Decoder attention, the output of the final Encoder in the stack is passed to the Value and Key parameters. The output of the Self-attention module below it is passed to the Query parameter.

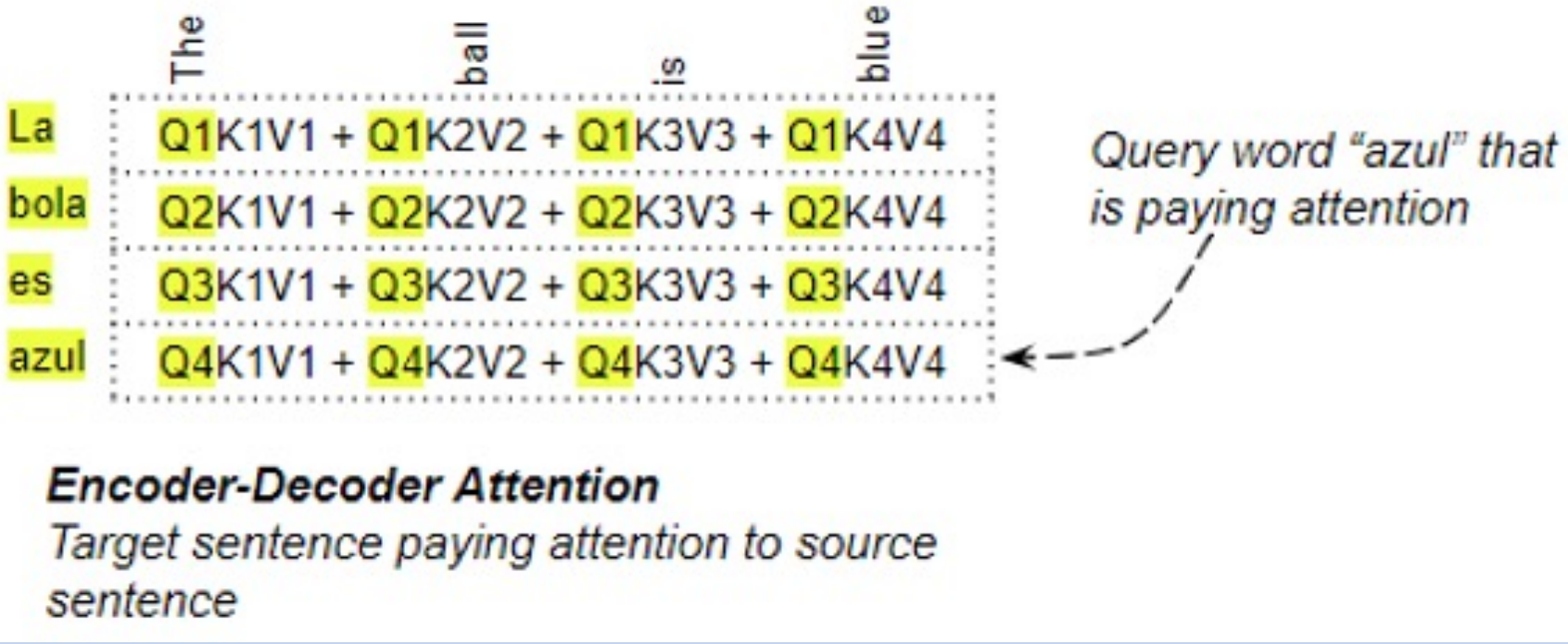




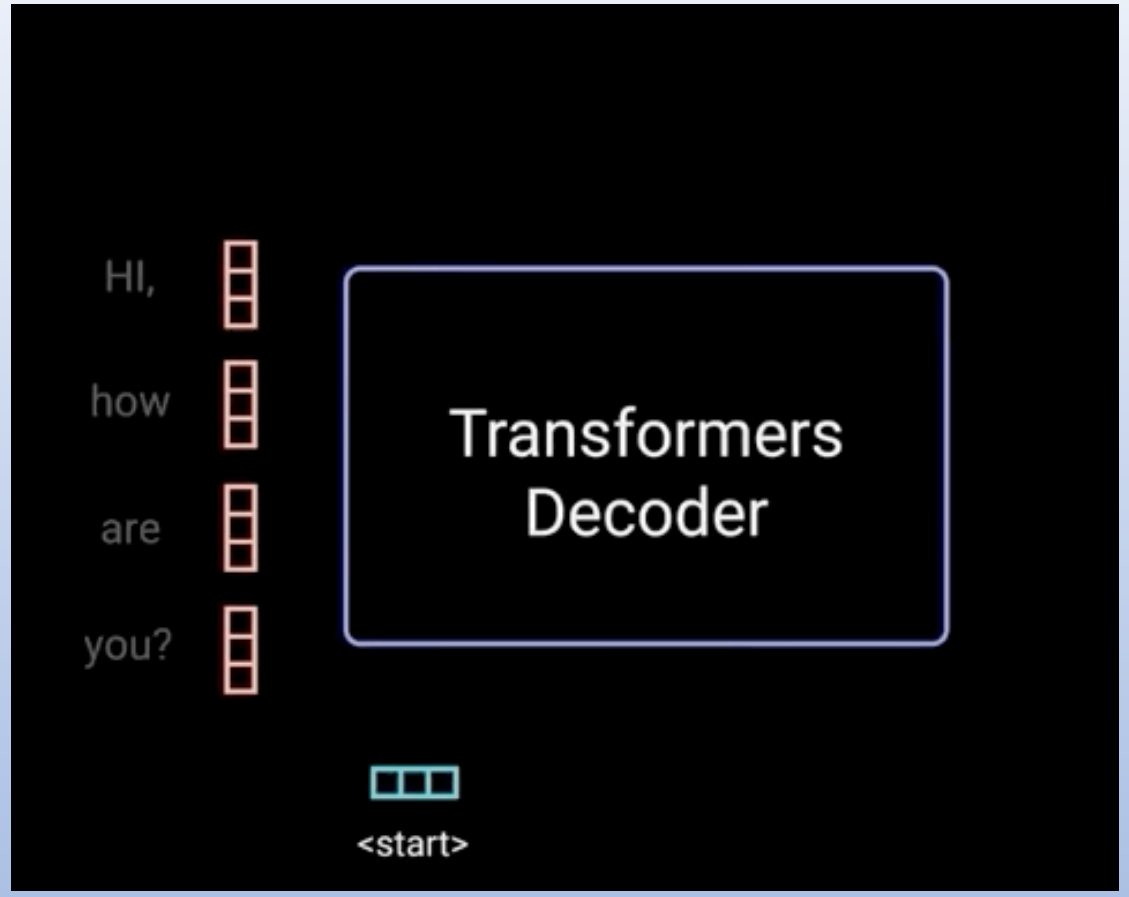
Self-attention in the Decoder — the target sequence pays attention to itself.

Encoder-Decoder-attention in the Decoder — the target sequence pays attention to the source sequence

Mask is used in decoder to exclude words/tokens that haven't appeared in the target.



Inputs: Hi, How are you? Output: I am fine. Transformer is a generative model



Attention is the core component.

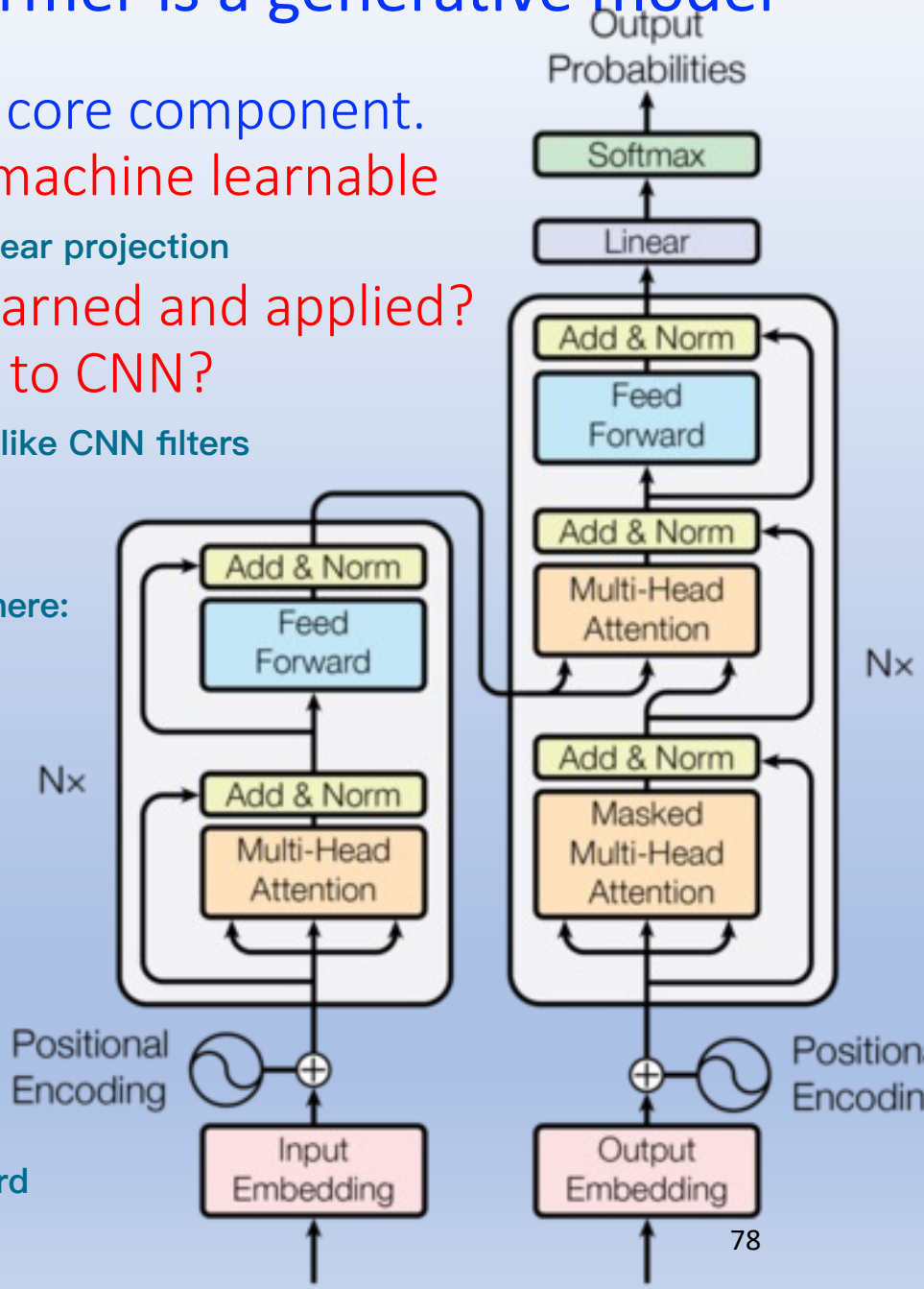
Where are the machine learnable parameters? linear projection

How are they learned and applied?

Any connection to CNN?

Wq Wk Wv are just like CNN filters

Feed Forward here: is 1 by 1 conv

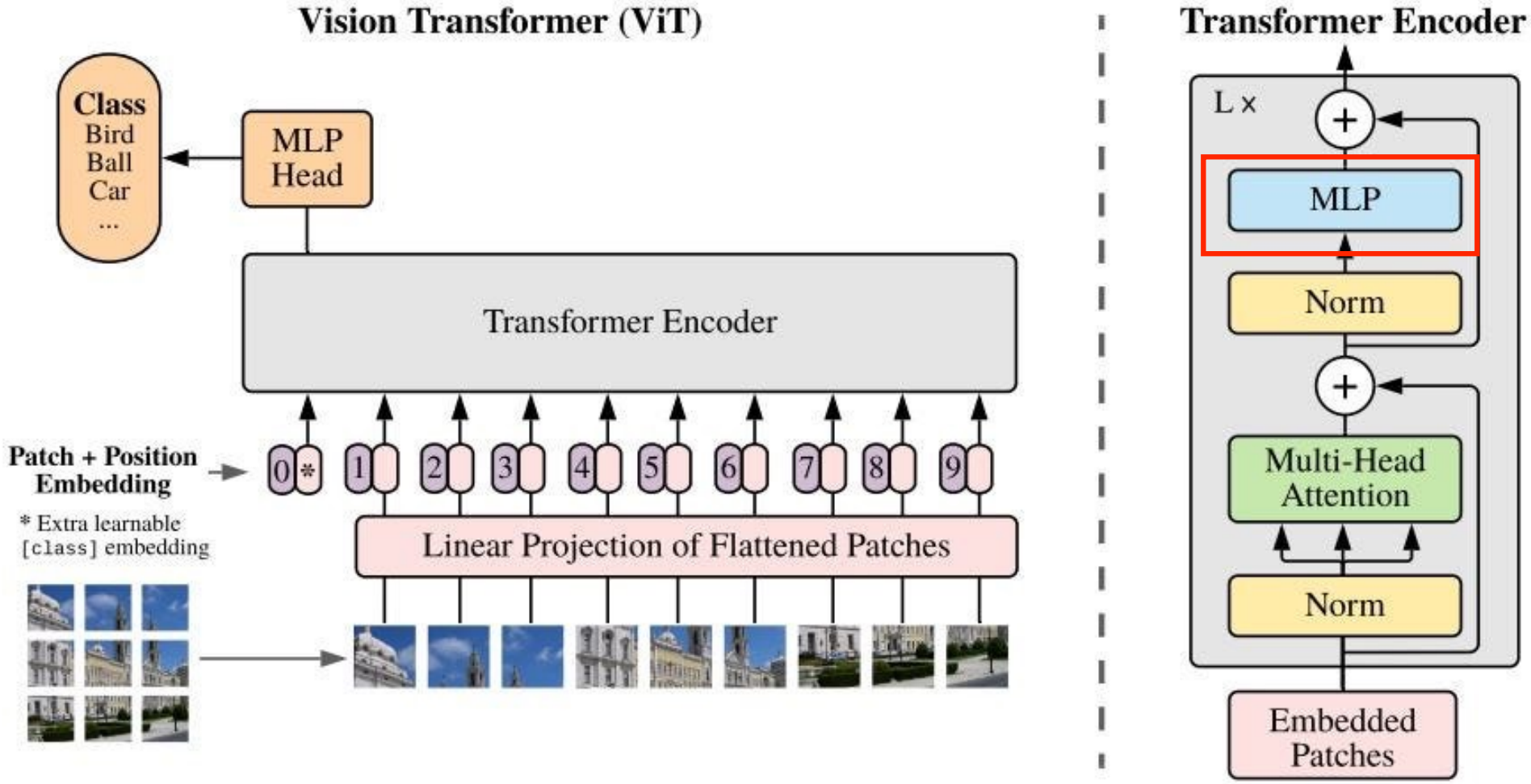


like "channels", same parameter apply to all tokens

$$\mathbf{X} = \begin{pmatrix} X1 \\ X2 \\ X3 \\ X4 \end{pmatrix}. \quad \mathbf{Q} = \mathbf{X}\boldsymbol{W_q}, \mathbf{K} = \mathbf{X}\boldsymbol{W_k}, \mathbf{V} = \mathbf{X}\boldsymbol{W_v}$$

$$\mathbf{Y} = f(f(f(\mathbf{X}\boldsymbol{W_1})\boldsymbol{W_2})\,\boldsymbol{W_3}).$$ feed forward

in X different row are never combined



note that this is not MLP, actually Feed Forward.
MLP must weighted sum all tokens

Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Vision Transformer vs CNN in Computer Vision

- The Vision Transformer (ViT) outperforms state-of-the-art convolutional networks in multiple benchmarks of computer vision while requiring fewer computational resources to train, after being pre-trained on **large amounts of data**.

- While CNNs have a proven track record in various computer vision tasks and handle large-scale datasets efficiently, Vision Transformers offer advantages in scenarios where ~~global~~ dependencies and contextual understanding are crucial. pair–wise among token relationship (non–distance dependency)

Vision-Language Transformer

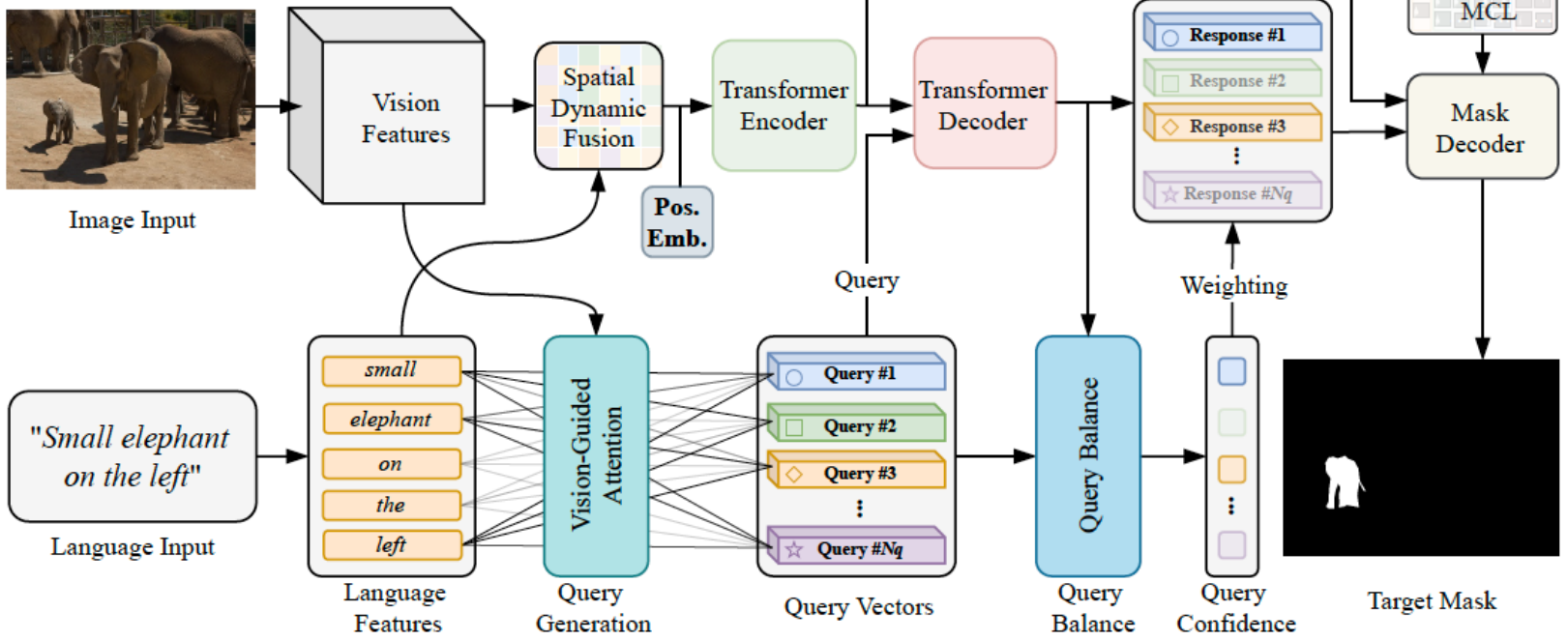


Fig. 2: The overview architecture of the proposed Vision-Language Transformer (VLT). Firstly, the given image and language expression are projected into visual and linguistic feature spaces, respectively. A Spatial Dynamic Fusion module is then employed to fuse vision and language features, generating multi-modal feature inputted to the transformer encoder. The proposed Query Generation Module generates a set of input-specific queries according to the vision and language features. These input-specific queries are sent to the decoder, producing corresponding query responses. These resulting responses are selected by the Query Balance Module and then decoded to output the target mask by a Mask Decoder. "Pos. Emb.": Positional Embeddings. "MCL": Masked Contrastive Learning.

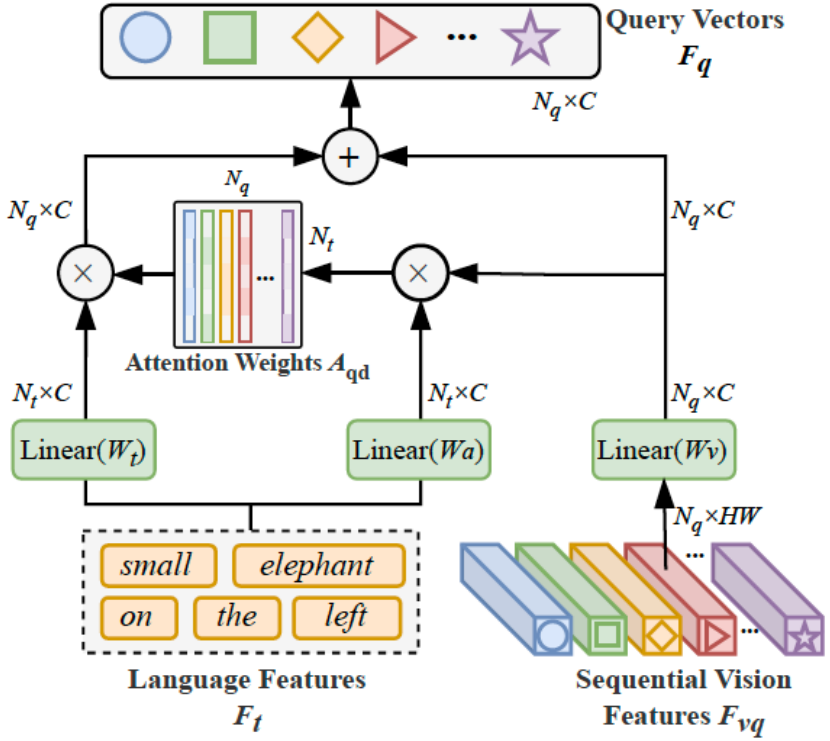


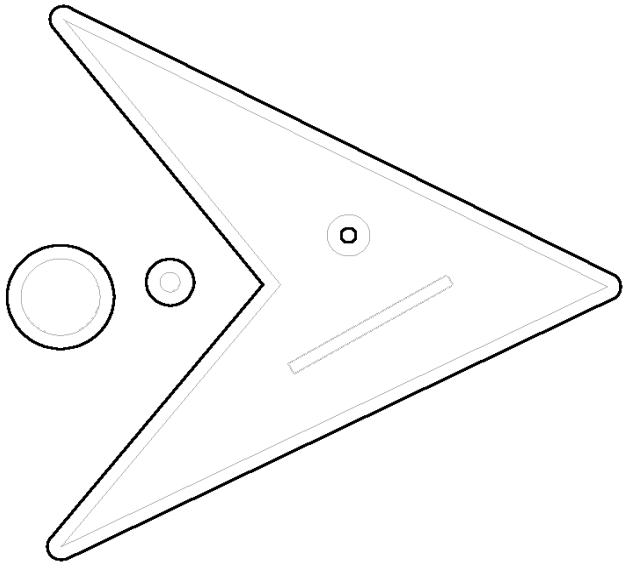
Fig. 6: Query Generation Module (QGM). The QGM takes sequential vision feature F_{vq} and language features F_t as inputs and generates a group of input-specific query vectors F_q, which are then sent to the transformer decoder of our VLT.

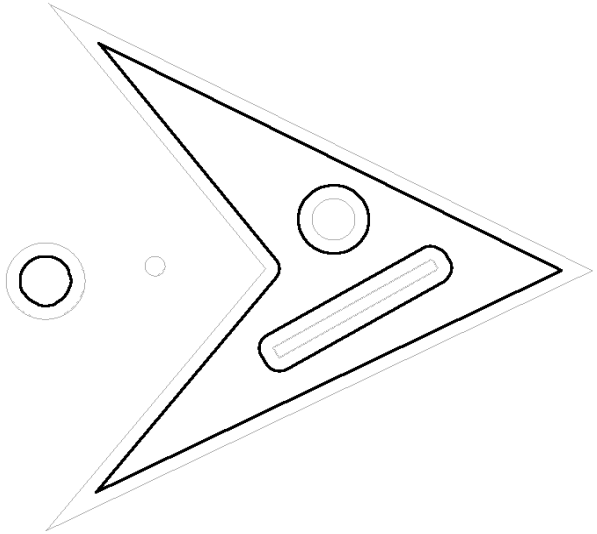
H. Ding, C. Liu, S. Wang, X. Jiang, "Vision-Language Transformer and Query Generation for Referring Segmentation," ***International Conference on Computer Vision* (ICCV'21)**, Oct 2021.

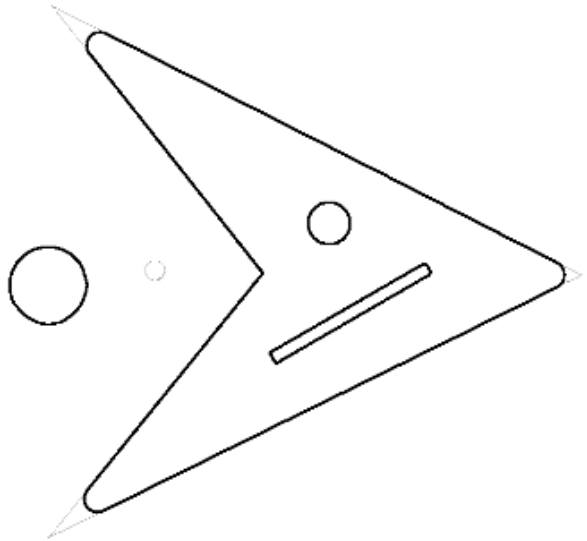
H. Ding, C. Liu, S. Wang, X. Jiang, "VLT: Vision-Language Transformer and Query Generation for Referring Segmentation," ***IEEE Transactions on Pattern Analysis and Machine Intelligence,*** vol. 45. no. 6, 7900–7916, Jun. 2023.

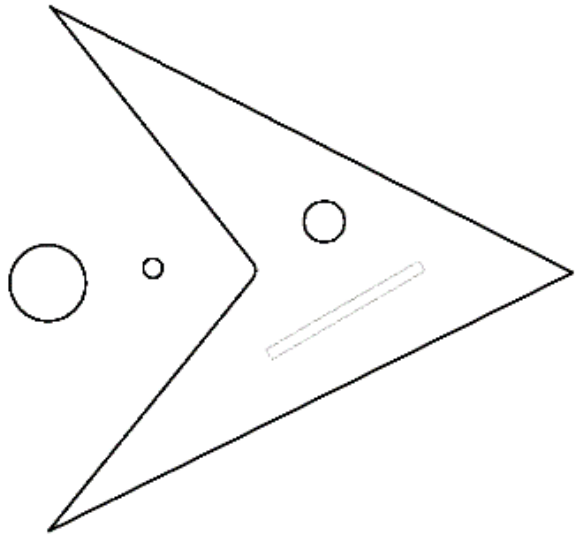
Conclusion of Deep Learning

- Artificial Intelligence (AI) is booming because of the machine learning.
- Machine learning is booming because of the deep learning.
- Deep learning is booming thanks to the Convolutional Neural Networks (CNN).
- CNN is a very strongly regularized NN. Local attention/correlation/relation of CNN has its merits and limitations.
- Transformer is again a very strongly regularized NN. It performs all possible (global) specific attention/correlation/relation. It also applies convolution concept!
- The most critical of machine learning is not simply to let machine to learn from the big data, but to use human knowledge to guide (regularize) the machine to learn the data.
- This is the central task of all computer vision and machine learning problems!









$$\text{Decide } \omega_k = \arg\min_{\omega_i}[p(e_i \mid \mathbf{x})] = \arg\min_{\omega_i}[1 - p(\omega_i \mid \mathbf{x})]$$

$$= \arg\max_{\omega_i}[p(\omega_i \mid \mathbf{x})] = \arg\max_{\omega_i}[p(\omega_i)\,p(\mathbf{x} \mid \omega_i)]$$

So the decision rule is: decide ω_1 if $p(\omega_1)p(x|\omega_1) > p(\omega_2)p(x|\omega_2)$, otherwise decide ω_2.

From the plot of $p(x|\omega_1)$, $p(x|\omega_2)$, both $p(x|\omega_1)$ and $p(x|\omega_2)$ are nonzero only if $1 < x < 2$.

For $1 < x < 2$, we have

$p(\omega_1)p(x|\omega_1) = 0.4(-x + 2)$

$p(\omega_2)p(x|\omega_2) = (1 - 0.4)[(0.5/3)x - 0.5/3]$

Decision rule is: decide ω_1 if $0.4(-x+2) > 0.6((0.5/3)x - 0.5/3)$

$\rightarrow \quad -0.4x+0.8 > 0.1x - 0.1$

$\rightarrow \quad 0.5x < 0.9$

$\rightarrow$ decide ω_1 if $x < 1.8$

Course Content

1. Image Fundamentals and Human Perception.
2. LSI Systems and Transforms
3. Image Denoising and Enhancement
4. Morphological Image Processing
5. MAP Decision and Classifiers
6. Statistical Estimation and Machine Learning
7. Eigenvalue and Eigenvector Decomposition of Data Matrix
8. Visual Data Dimensionality Reduction
9. Neural Networks and Deep Machine Learning: from MLP to CNN
10. Deep Learning: from CNN to Transformer
11. Video Analysis
12. Video Recognition
13. Three-dimensional Machine Perception
14. Three-dimensional Machine Vision

1

2

3

4