CH. 12: Deep Learning

12.1 Introduction

- In principle, multilayer perceptron (MLP) with one hidden layer can approximate any function with arbitrary accuracy. However, it may need a very large number of hidden units to achieve the purpose.
- Empirically, a "long and thin" network not only has fewer parameters than a "short and fat" one but also achieves better generalization.

- Deep neural networks DNN consist of many hidden hidden layers, each with only a few units.
- Different from MLP (fully connected neural networks), in which each hidden unit is connected to all the inputs; in DNN, hidden units are fed with localized patches.

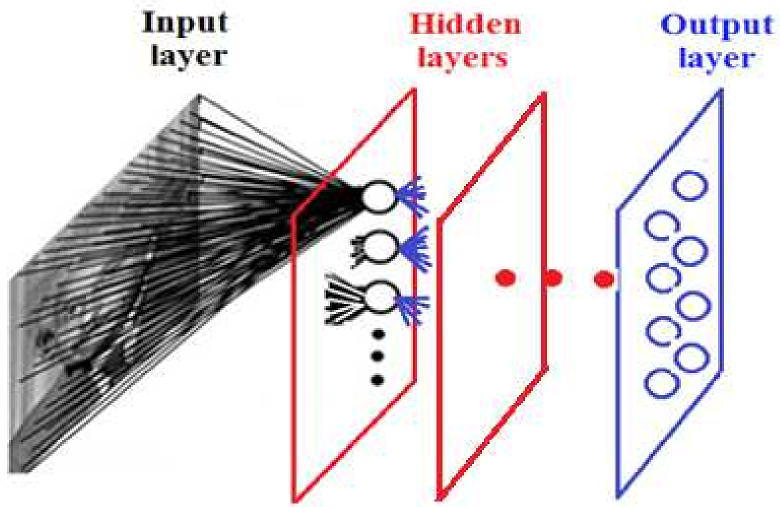
Fully connected networks (MLP)

e.g., 1000 by 1000 image

10^6 hidden units

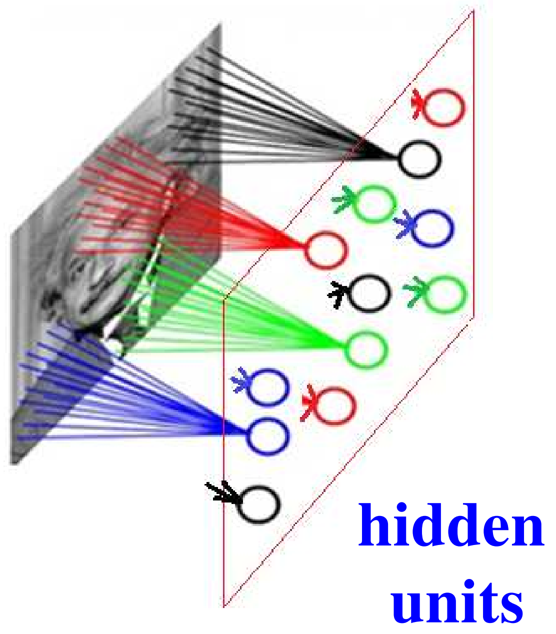
$10^3 \times 10^3 \times 10^6$

parameters



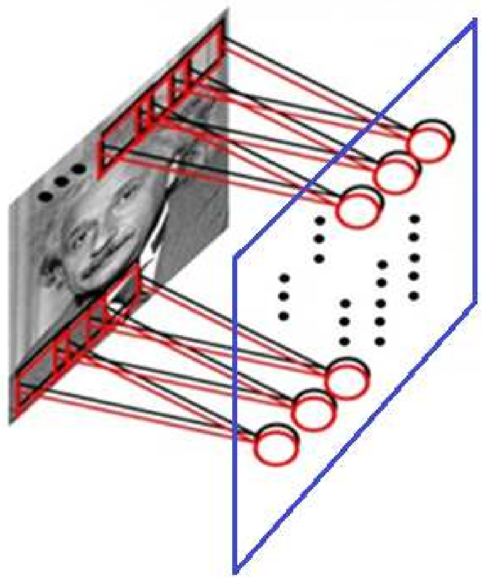
Locally connected networks

e.g., 1000 by 1000 image
10^6 hidden units each
with a different filter
Filter size 10 by 10
$10 \times 10 \times 10^6$ parameters



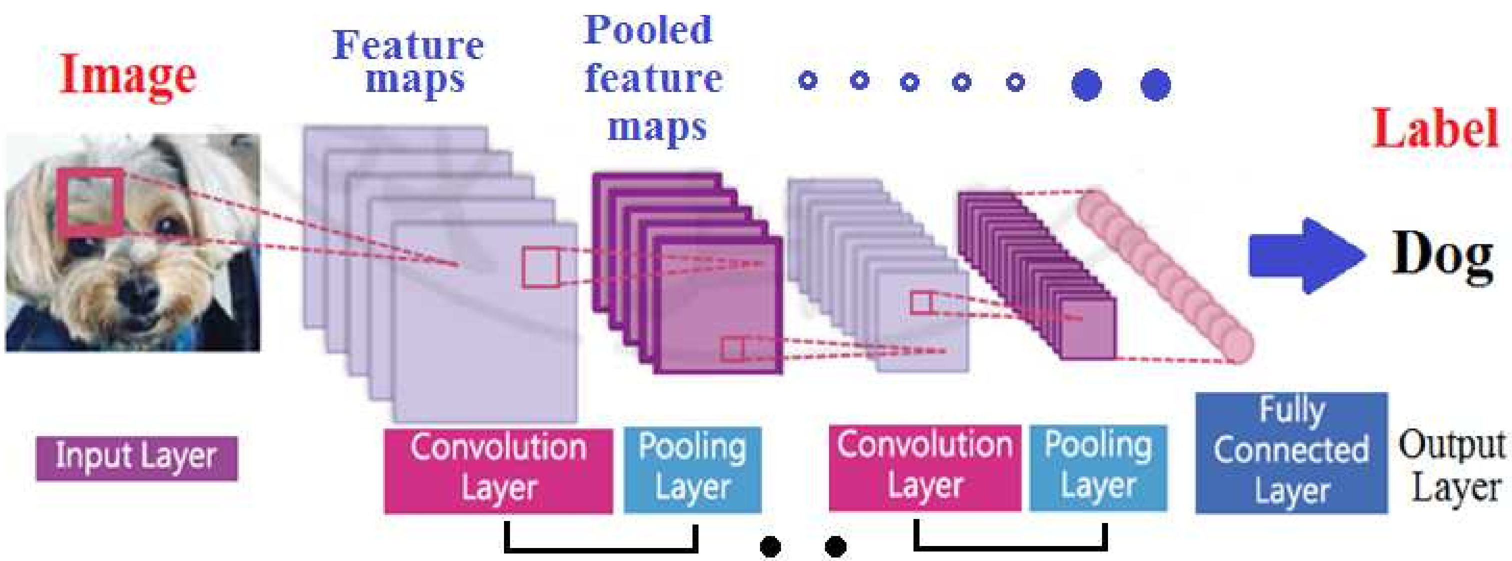
Convolutional networks

e.g., 1000 by 1000 image
10^6 hidden units each
with 2 different filters
Filter size 10 by 10
$2 \times 10 \times 10$ parameters



12.2 Convolutional Neural Networks

12.2.1 Architecture



Three kinds of layers:

(a) Convolution layer,

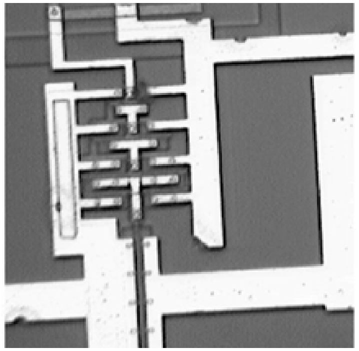
(b) Pooling layer,

(c) Fully connected layer

12.2.2 Production Phase

(a) Convolution layer – feature extraction

Two major ingredients: i) filters, ii) convolution

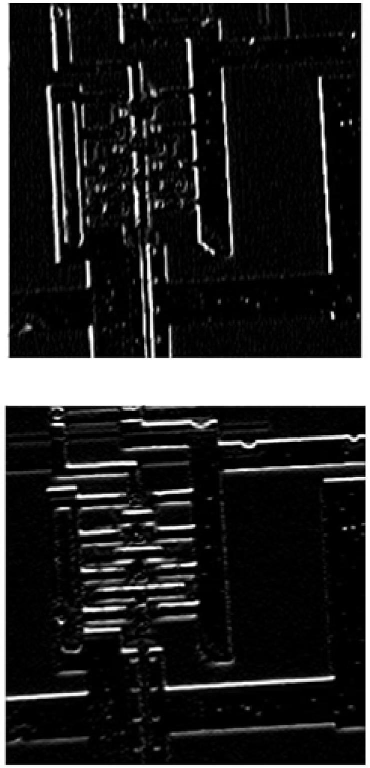


Input image

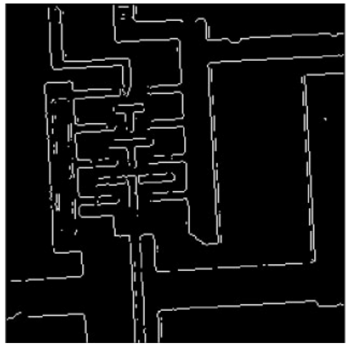
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Filters

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$



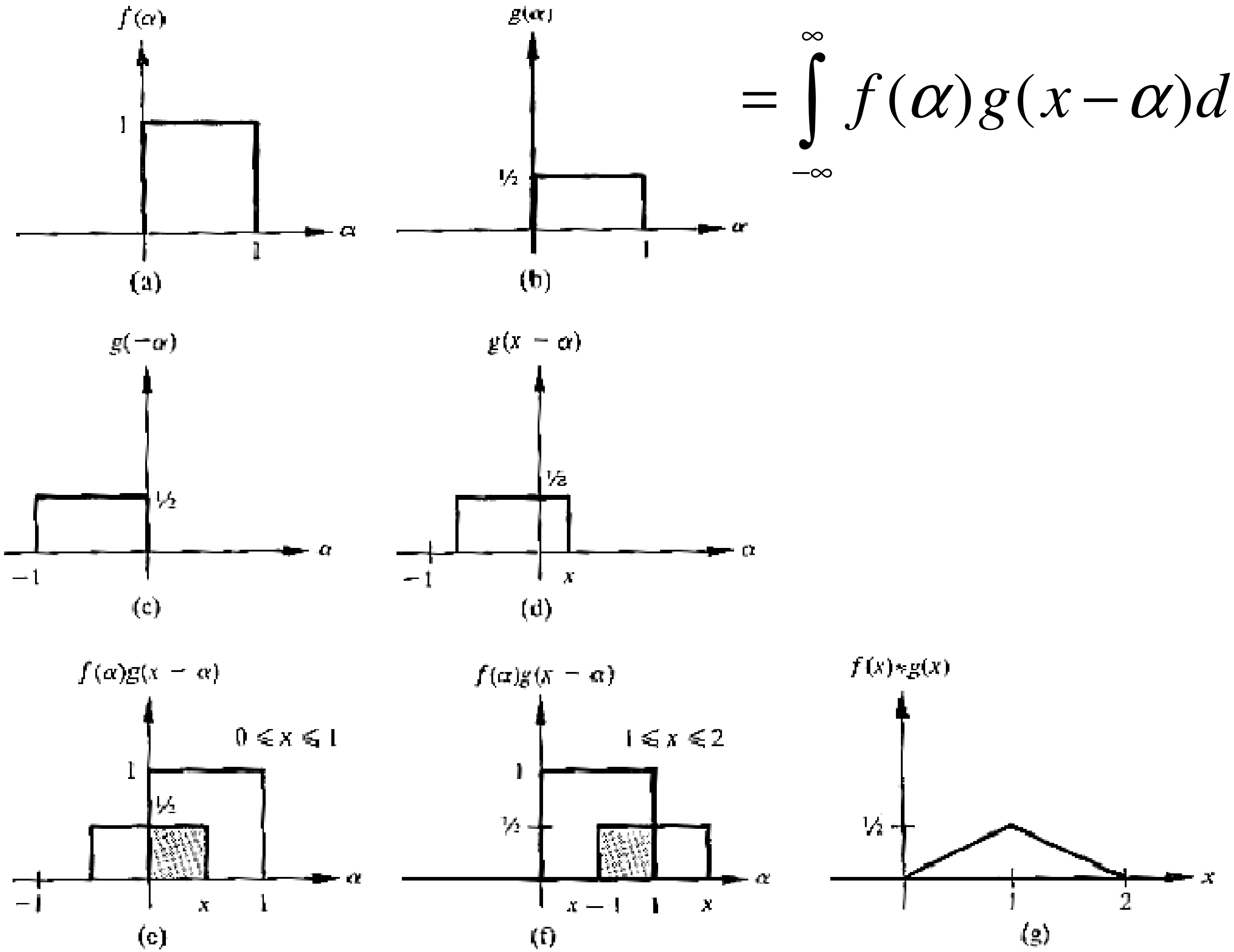
Feature maps



Edge image

- Convolution $f(x) * g(x) = \int_{-\infty}^{\infty} f(x-\alpha) g(\alpha) d\alpha$

$$= \int_{-\infty}^{\infty} f(\alpha) g(x-\alpha) d\alpha$$



Summarize the process of convolution

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha) g(x-\alpha) d\alpha$$

1. Reverse $g(x) \Rightarrow g(-x)$

2. Move $g(-x)$ from $-\infty \Rightarrow \infty$

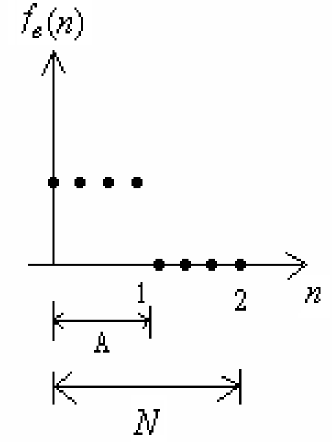
3. Calculate and record the overlapping area between $f(x)$ and $g(-x)$ at every point.

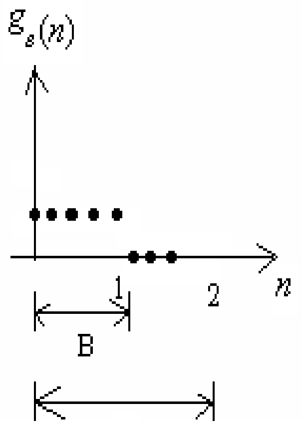
Discrete case:

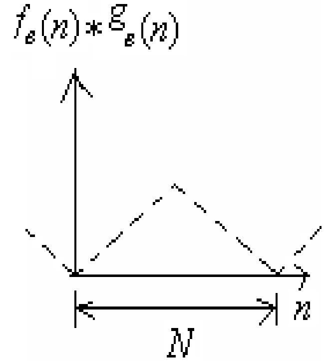
$$f_e(n) * g_e(k) = \sum_{n=0}^{N-1} f_e(n) g_e(n-k) \quad N \geq A + B - 1$$

where f_e, g_e : extended versions of f, g

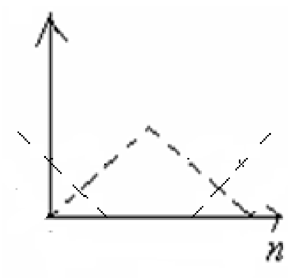
e.g., $A = 4, B = 5, A + B - 1 = 8, \quad N \geq 8$

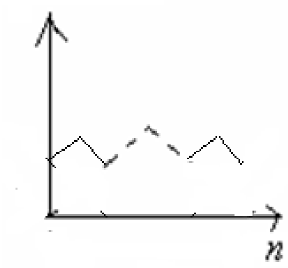


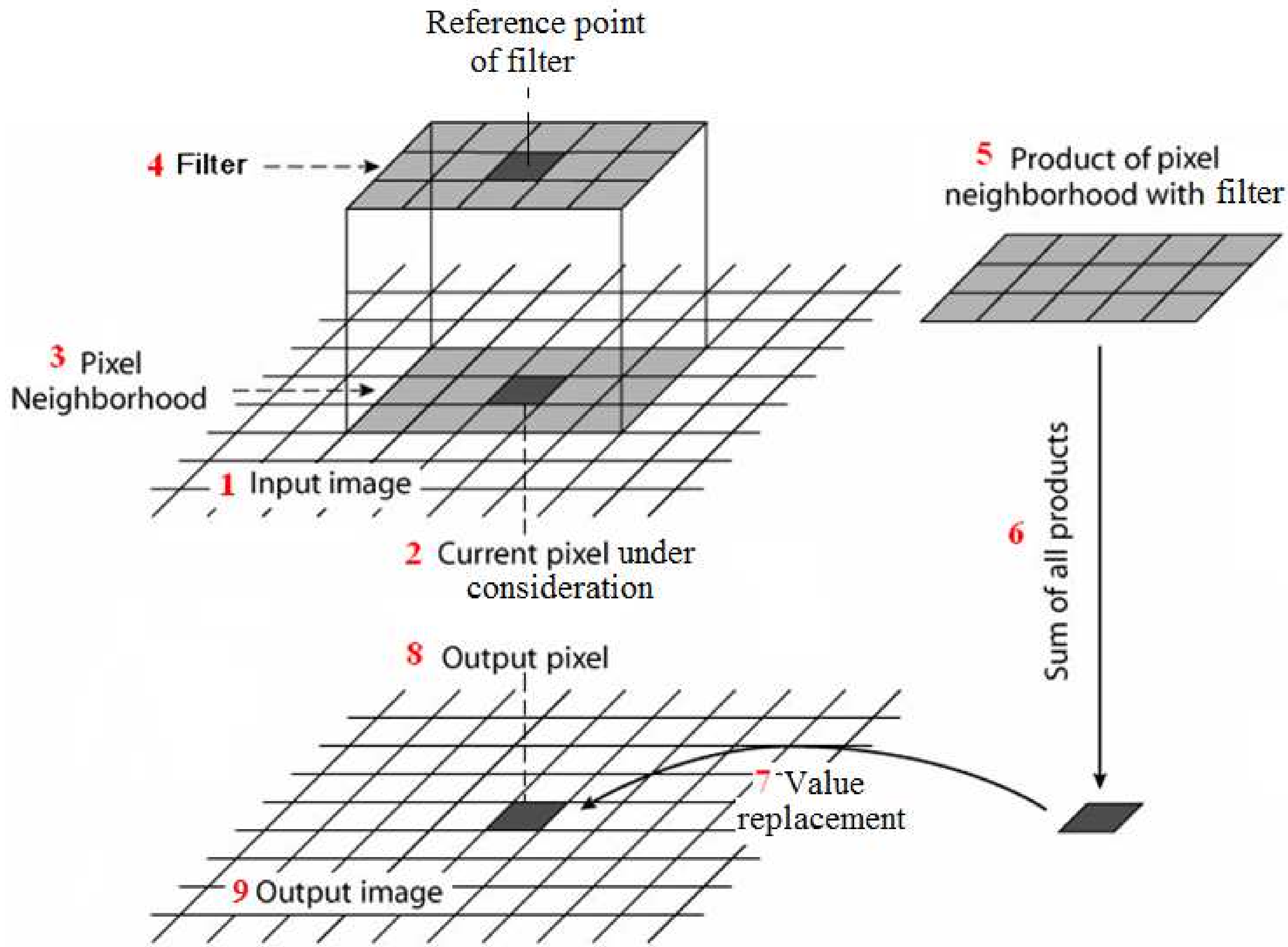




$N < 8$






Reference point
of filter
4 Filter
5 Product of pixel
neighborhood with filter
3 Pixel
Neighborhood
1 Input image
2 Current pixel under
consideration
6
Sum of all products
8 Output pixel
7 Value
replacement
9 Output image

Filter values

$m(-1,-2)$	$m(-1,-1)$	$m(-1,0)$	$m(-1,1)$	$m(-1,2)$
$m(0,-2)$	$m(0,-1)$	$m(0,0)$	$m(0,1)$	$m(0,2)$
$m(1,-2)$	$m(1,-1)$	$m(1,0)$	$m(1,1)$	$m(1,2)$

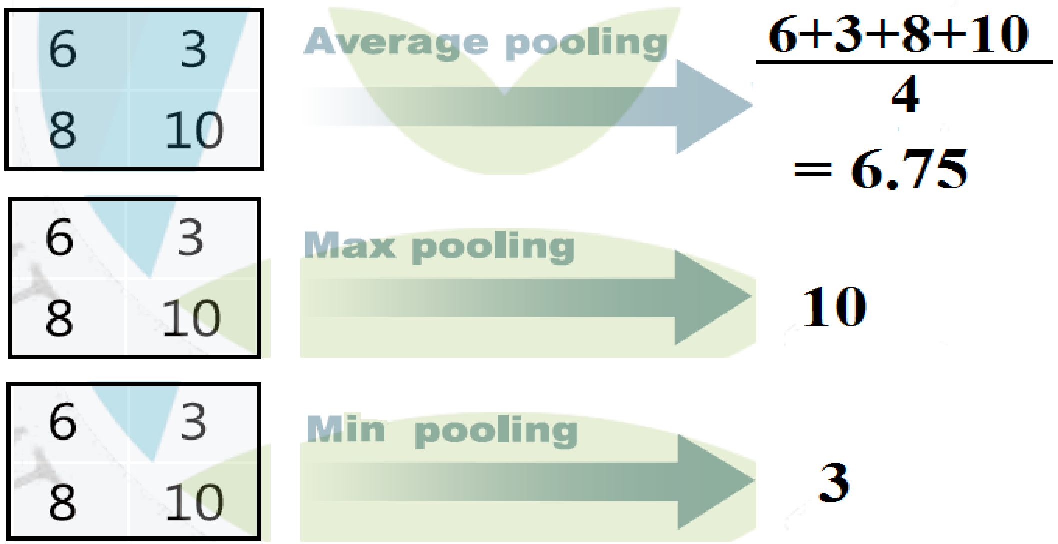
Pixel values

$p(x-1,y-2)$	$p(x-1,y-1)$	$p(x-1,y)$	$p(x-1,y+1)$	$p(x-1,y+2)$
$p(x,y-2)$	$p(x,y-1)$	$p(x,y)$	$p(x,y+1)$	$p(x,y+2)$
$p(x+1,y-2)$	$p(x+1,y-1)$	$p(x+1,y)$	$p(x+1,y+1)$	$p(x+1,y+2)$

$$\begin{aligned} p'(x,y) &= m(-1,-2)\,p(x-1,y-2) + m(-1,-1)\,p(x-1,y-1) \\ &\quad + \cdots\cdots + m(1,1)\,p(x+1,y+1) + m(1,2)\,p(x+1,y+2) \\ &= \sum_{s=-1}^{1}\sum_{t=-2}^{2} m(s,t)\,p(x+s,y+t) \end{aligned}$$

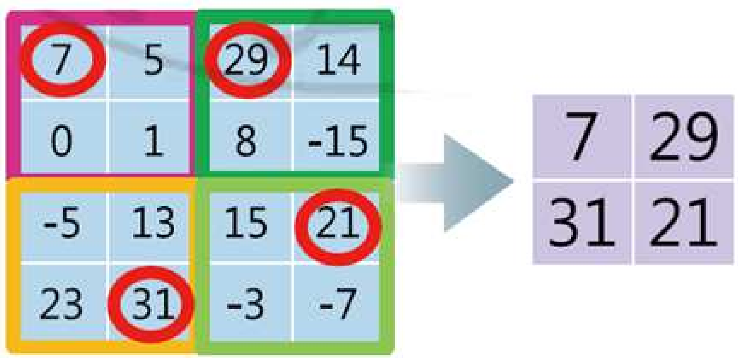
(b) **Pooling layer** – size adjustment

Potential operations: AVE, MAX, MIN



Example:

MAX pooling



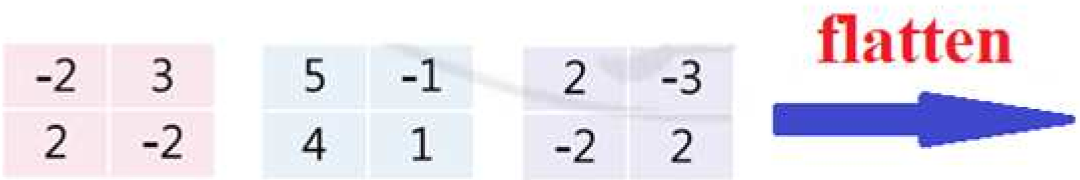
(c) **Fully connected layer** – produces outcome

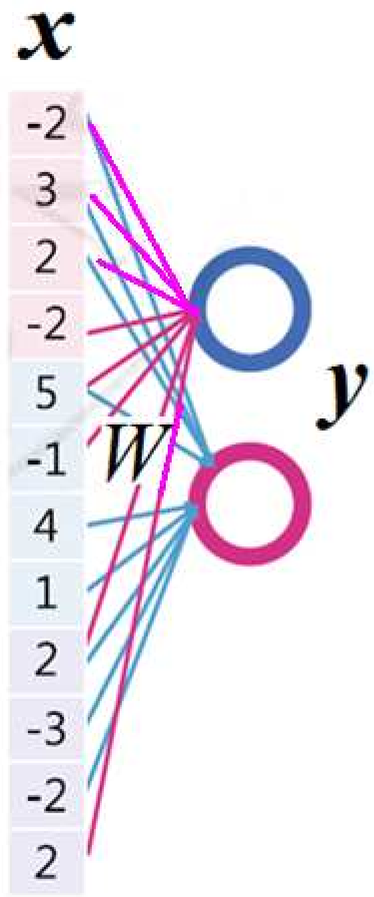
e.g., regression or classification

Two major operations:

i) flatten,

ii) multiplication

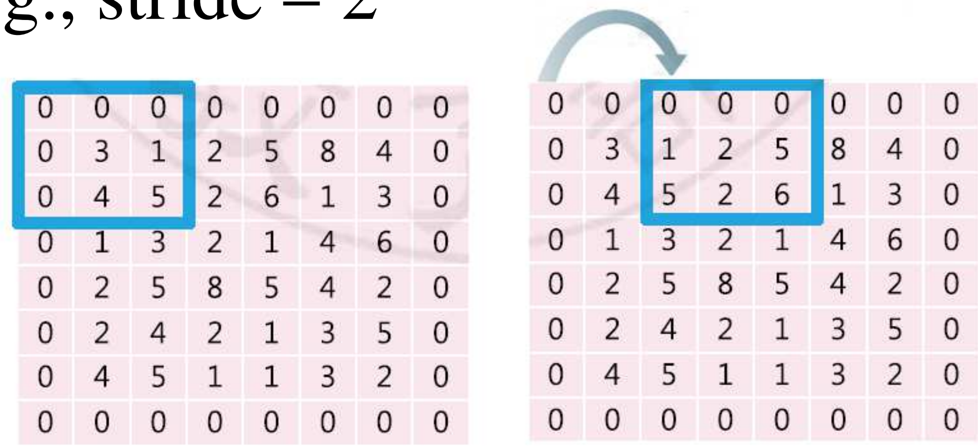




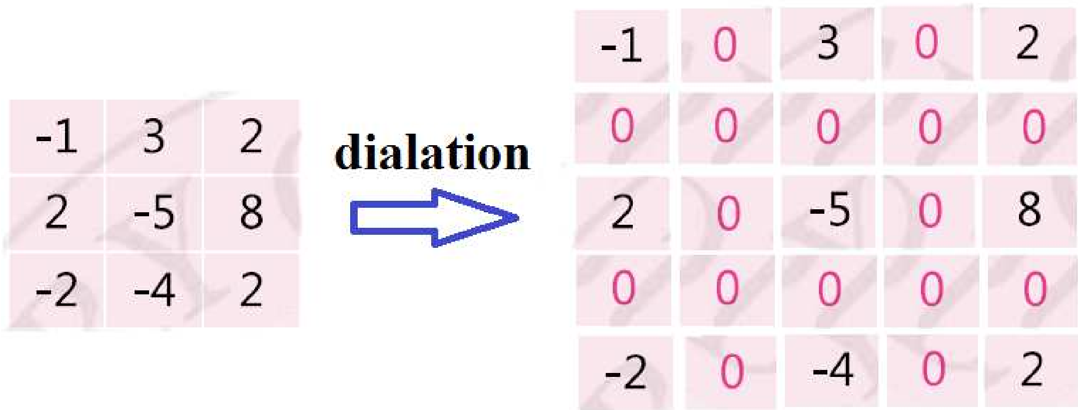
Multiplication: $\boldsymbol{y} = W\boldsymbol{x}$

12.2.3 Production Tactics

- Stride e.g., stride = 2



- Atrous Convolution e.g., rate = 1



12.2.3 Training Phase

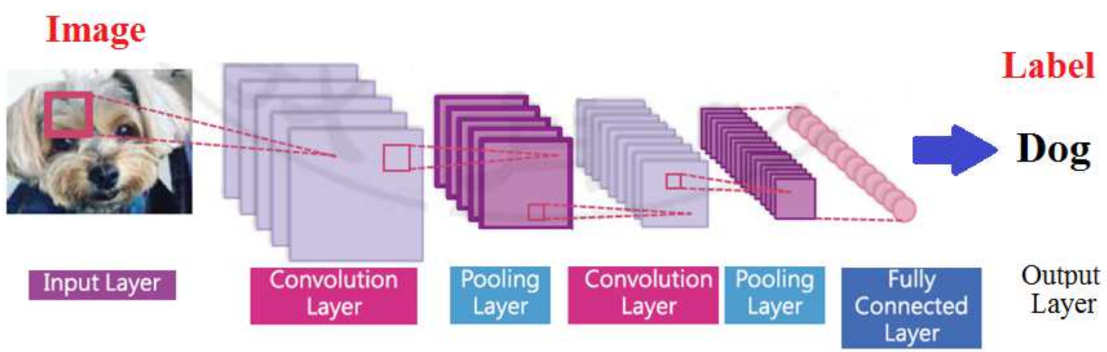
Parameters to be learned:

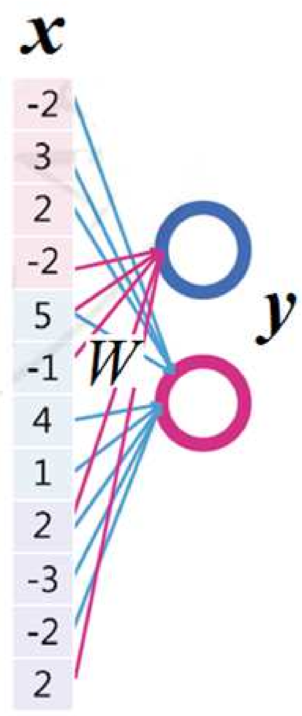
i) Convolution layer

-- convolutional filters

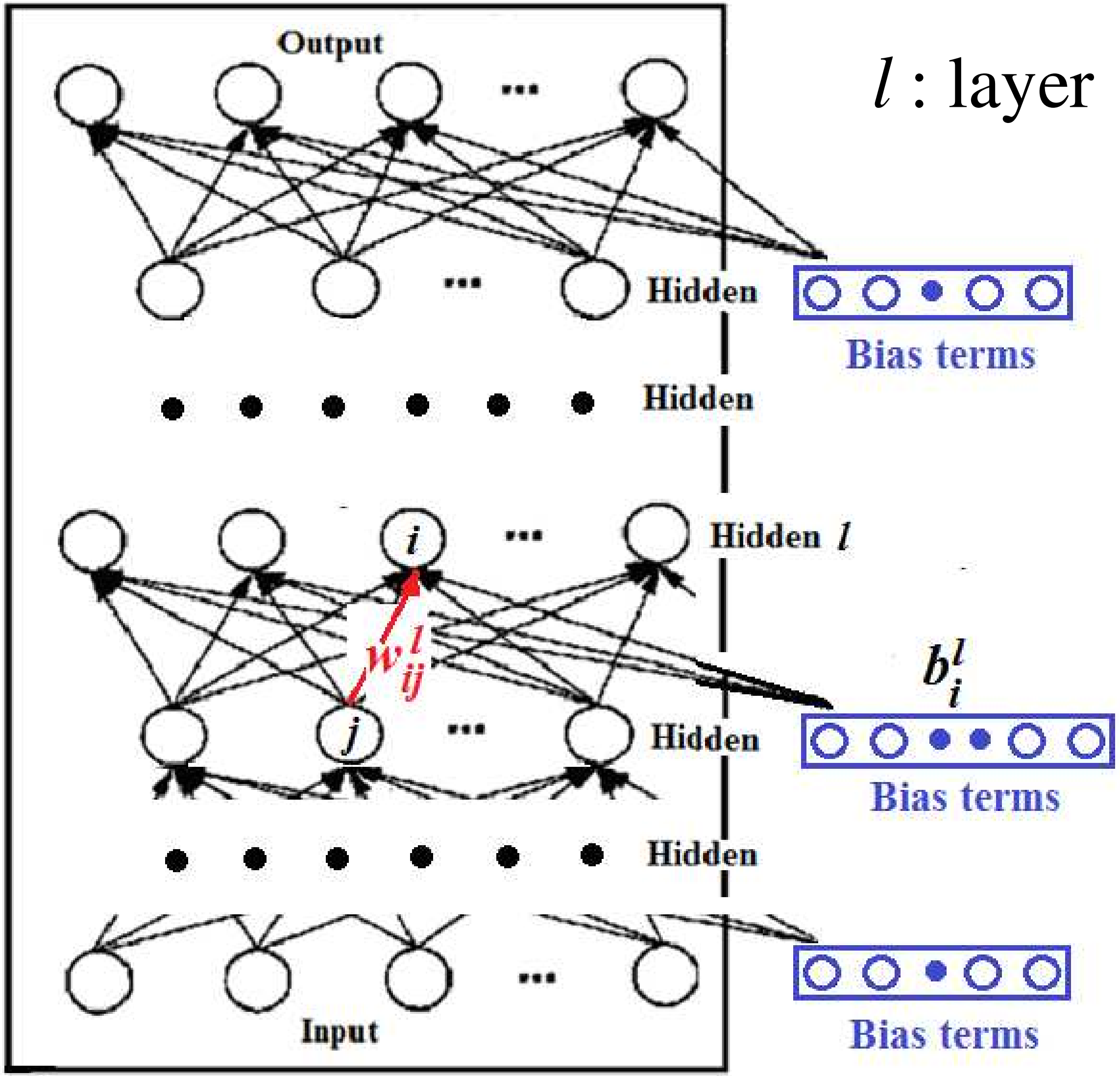
ii) Fully connected layer

-- synaptic weights

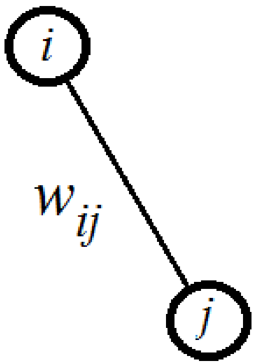




- Error (or loss) function $E(\boldsymbol{\theta})$, where $\boldsymbol{\theta} = (w_{ij}^{l}, b_{i}^{l})_{i,j,l}$



l : layer



Focus on parameters w_{ij}^{l}

Update rule: $W(t+1) = W(t) + \Delta W = W(t) - \eta \nabla_{\boldsymbol{\theta}} E$

- Chain Rule:

i) $y = g\left(x\right),\ z = h\left(y\right)$

$$\because\ \Delta x \to \Delta y \to \Delta z \quad \Rightarrow \quad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

ii) $x = g\left(s\right),\ y = h\left(s\right),\ z = k\left(x, y\right)$

$$\because \quad \begin{tikzcd} & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[dr] & & \Delta z \\ & \Delta y \arrow[ur] & \end{tikzcd} \quad \Rightarrow \quad \frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$$

Given a set of training examples

$\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \cdots, (\boldsymbol{x}_p, \boldsymbol{y}_p), \cdots, (\boldsymbol{x}_P, \boldsymbol{y}_P)\}$, where $\boldsymbol{y}_p = \Phi(\boldsymbol{x}_p)$

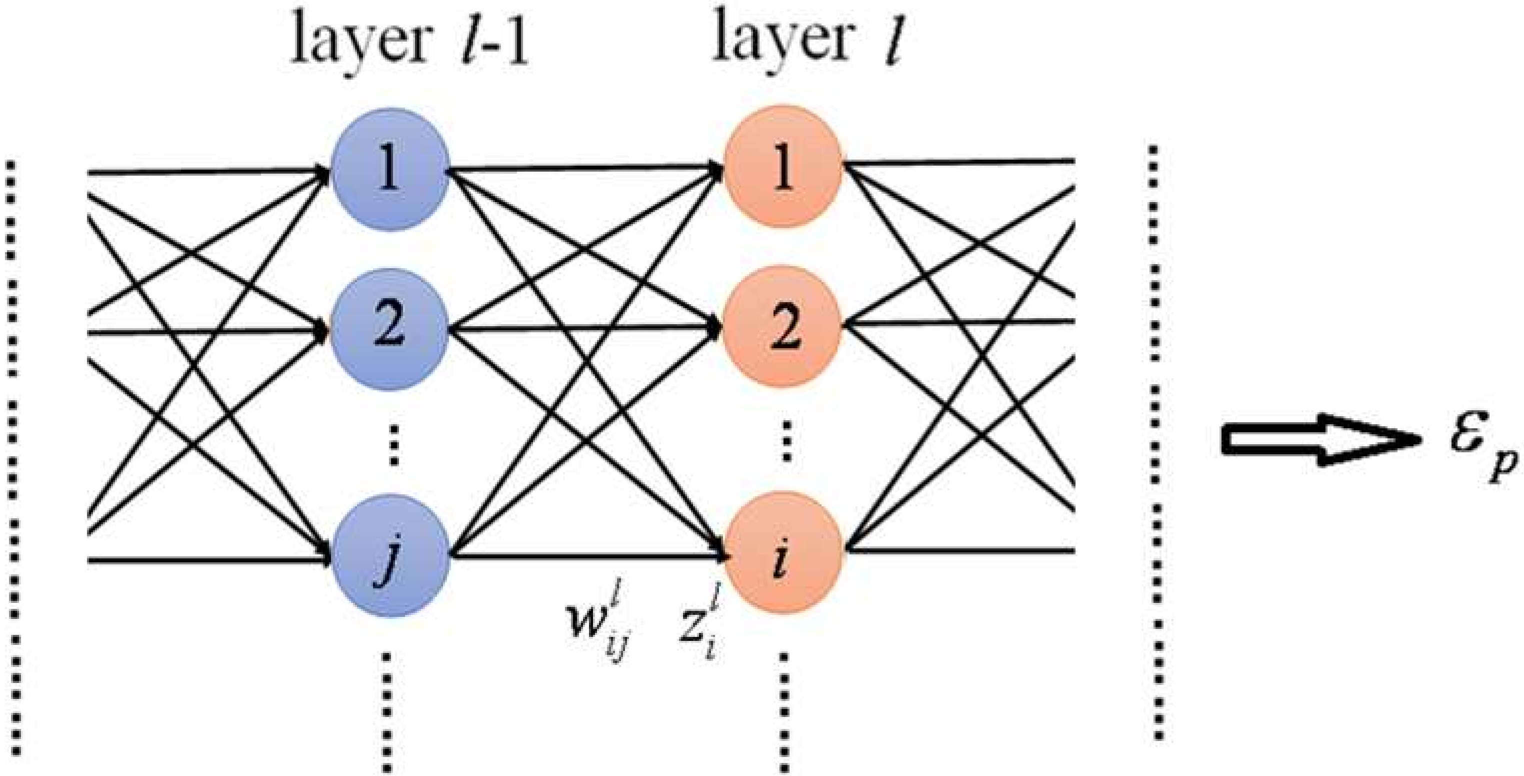
Find optimal $\boldsymbol{\theta}^*$ by minimizing

$$E(\boldsymbol{\theta}) = \frac{1}{P} \sum_{p=1}^{P} \left\| \Phi(\boldsymbol{x}_p \mid \boldsymbol{\theta}) - \boldsymbol{y}_p \right\| = \frac{1}{P} \sum_{p=1}^{P} \varepsilon_p(\boldsymbol{\theta})$$

Gradient: $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \frac{1}{P} \sum_{p=1}^{P} \nabla_{\boldsymbol{\theta}} \varepsilon_p(\boldsymbol{\theta})$

where $\nabla_{\boldsymbol{\theta}} \varepsilon_p(\boldsymbol{\theta}) = \left(\cdots, \frac{\partial \varepsilon_p}{\partial w_{ij}^l}, \cdots, \frac{\partial \varepsilon_p}{\partial b_i^l} \cdots \right)^T$

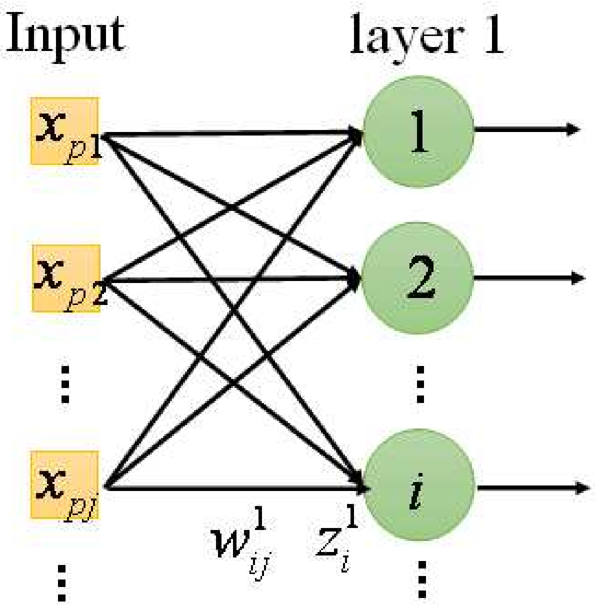
- **Consider** $\dfrac{\partial \varepsilon_p}{\partial w_{ij}^l}$



$$\Delta w_{ij}^l \rightarrow \Delta z_i^l \rightarrow \cdots \rightarrow \Delta \varepsilon_p \Rightarrow \frac{\partial \varepsilon_p}{\partial w_{ij}^l} = \frac{\partial \varepsilon_p}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l}$$

Compute $\dfrac{\partial z_i^l}{\partial w_{ij}^l}$

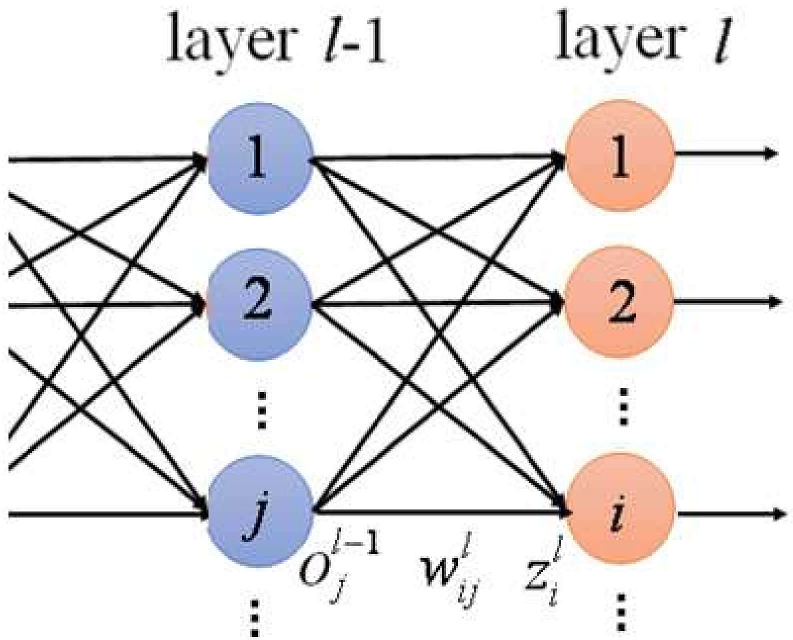
$l = 1$



$$z_i^1 = \sum_j x_{pj} w_{ij}^1 + b_i^1,$$

$$\frac{\partial z_i^1}{\partial w_{ij}^1} = x_{pj}$$

$l > 1$



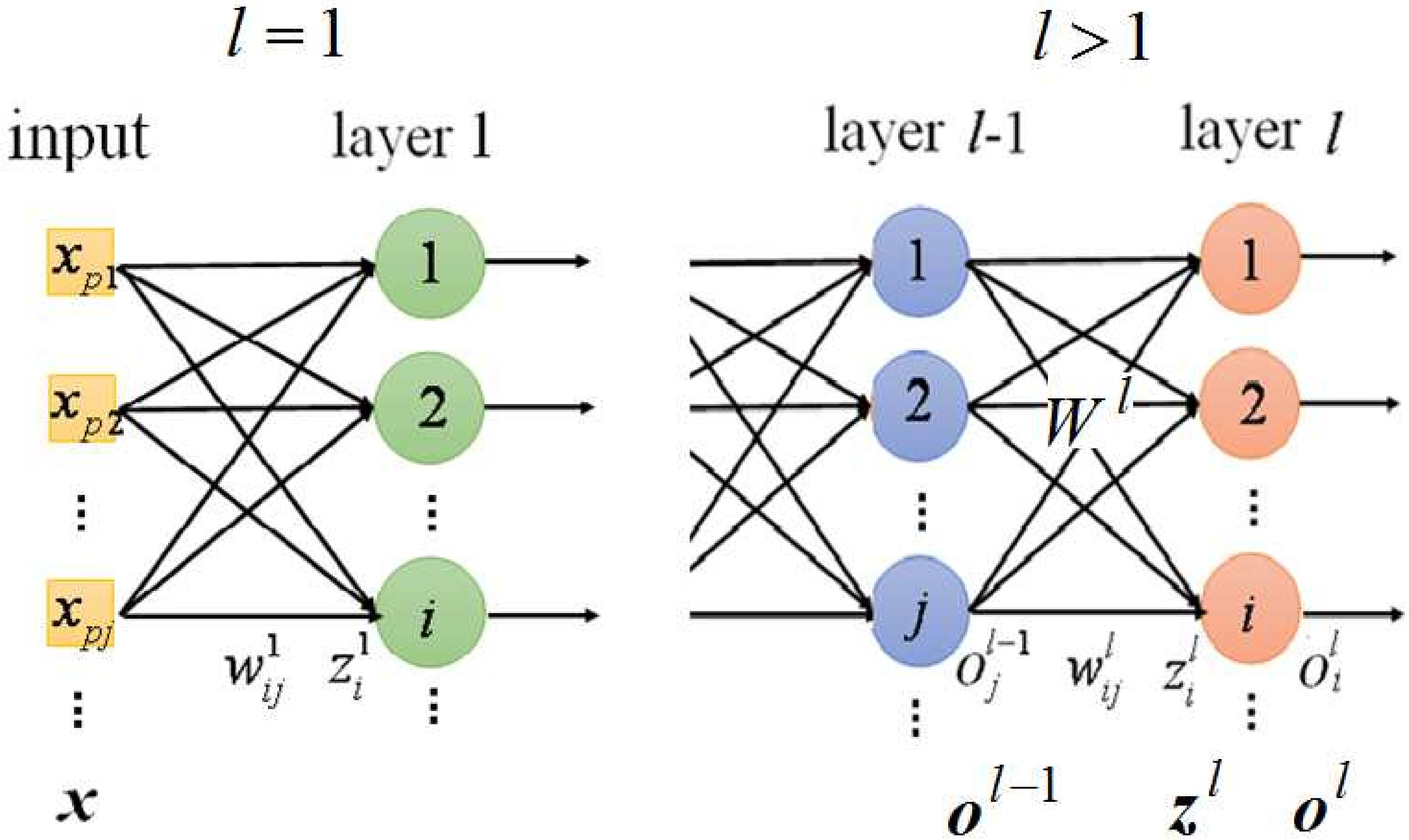
$$z_i^l = \sum_j o_j^{l-1} w_{ij}^l + b_i^l,$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = o_j^{l-1}$$

Forward Pass

In vector form,

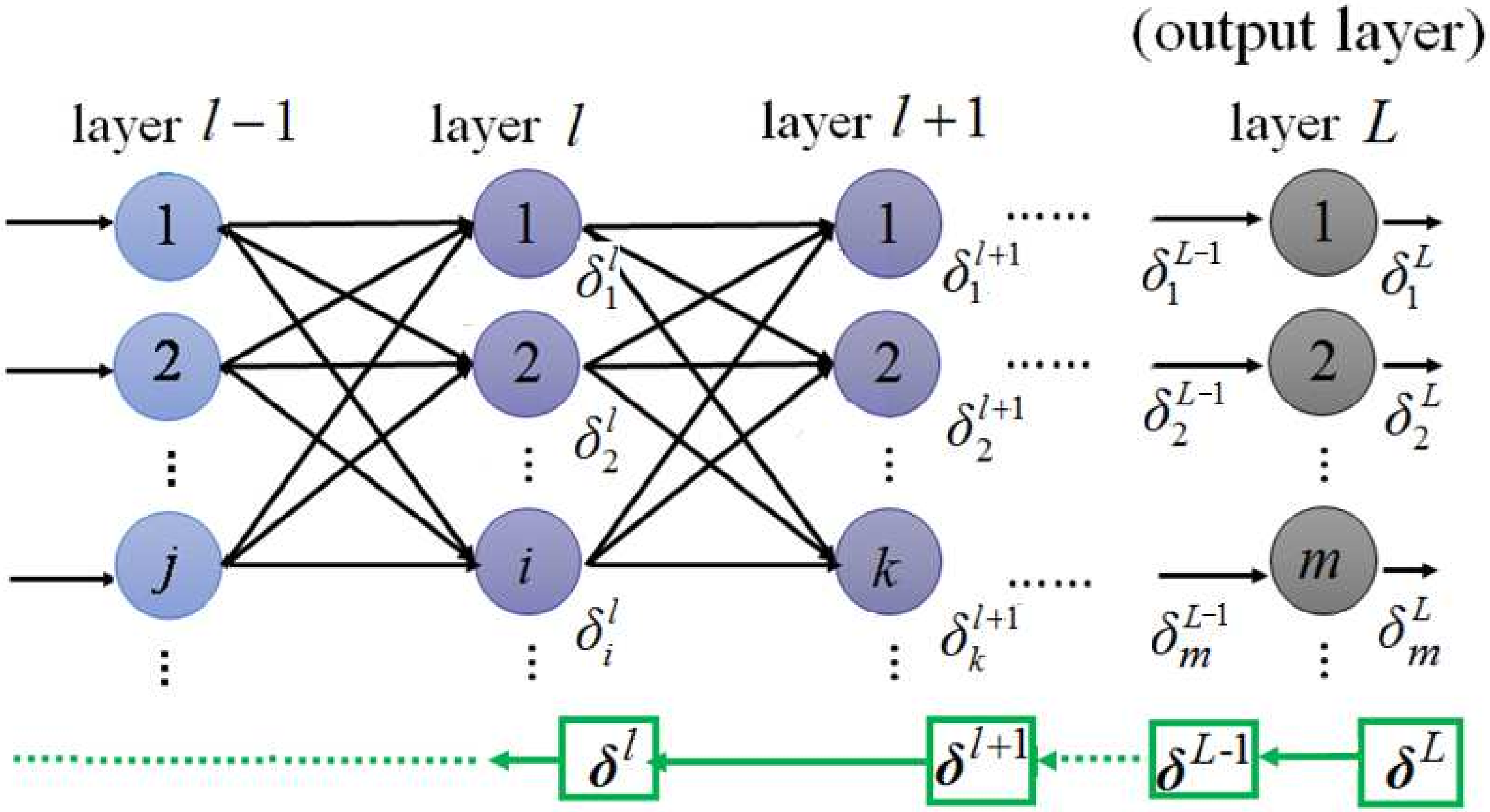
$$l = 1,\ \boldsymbol{x}; \quad l > 1,\ \boldsymbol{o}^l = \sigma(\boldsymbol{z}^l) = \sigma\left(W^l \boldsymbol{o}^{l-1} + \boldsymbol{b}^l\right)$$



Compute $\dfrac{\partial \varepsilon_p}{\partial z_i^l} = \delta_i^l$

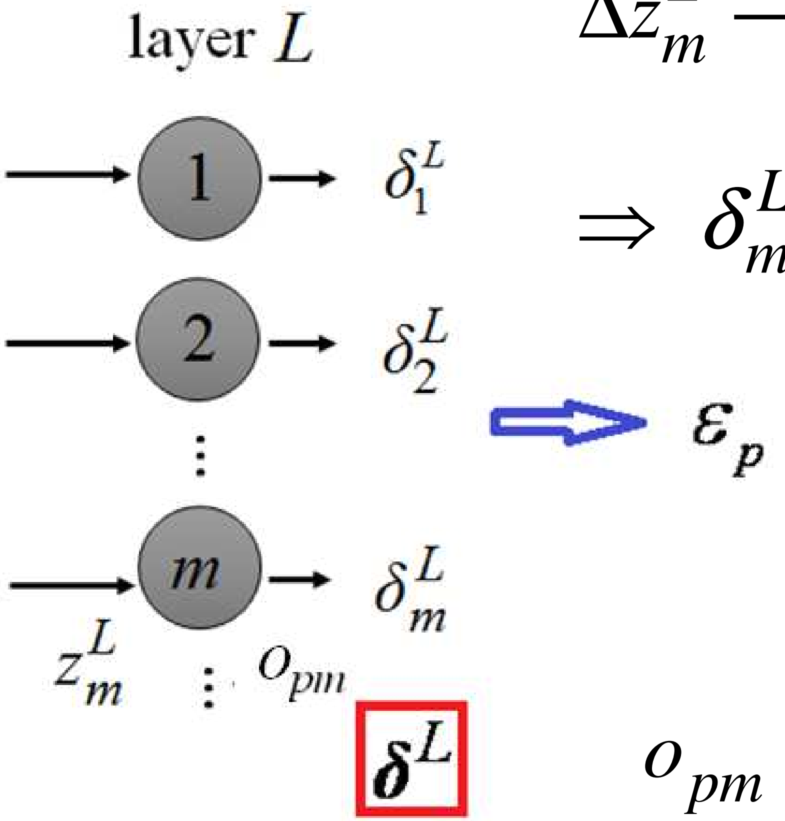
Step 1: Compute $\boldsymbol{\delta}^L = (\delta_1^L, \delta_2^L, \cdots, \delta_M^L)$

Step 2: Determine the relation between $\boldsymbol{\delta}^l$ and $\boldsymbol{\delta}^{l+1}$



Step 1: Compute $\boldsymbol{\delta}^L = (\delta_1^L, \delta_2^L, \cdots, \delta_M^L)$, $\delta_i^L = \dfrac{\partial \varepsilon_p}{\partial z_i^L}$

$$\Delta z_m^L \to \Delta o_{pm} \to \Delta \varepsilon_p$$



$$\Rightarrow \quad \delta_m^L = \frac{\partial \varepsilon_p}{\partial z_m^L} = \frac{\partial \varepsilon_p}{\partial o_{pm}} \frac{\partial o_{pm}}{\partial z_m^L}$$

$$= \frac{\partial \varepsilon_p}{\partial o_{pm}} \sigma'\left(z_m^L\right)$$

$o_{pm} = \sigma\left(z_m^L\right)$, where

$\sigma(\cdot)$: activation function

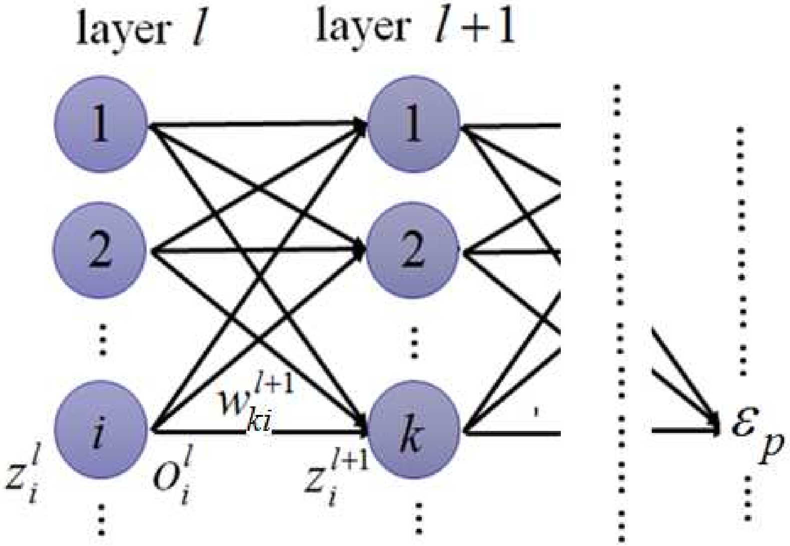
$$\delta_m^L = \sigma'\left(z_m^L\right)\frac{\partial \varepsilon_p}{\partial o_{pm}}, \quad m = 1, 2, \cdots, M$$

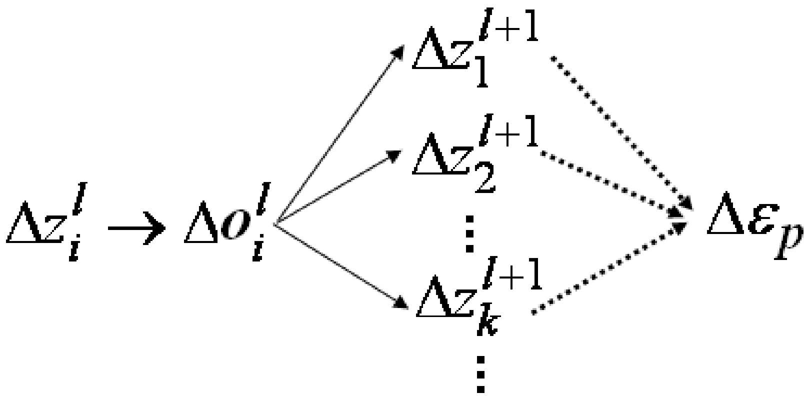
In vector form, $\boldsymbol{\delta}^L = \sigma'\left(\boldsymbol{z}^L\right) \odot \nabla \varepsilon_p\left(\boldsymbol{o}_p\right)$

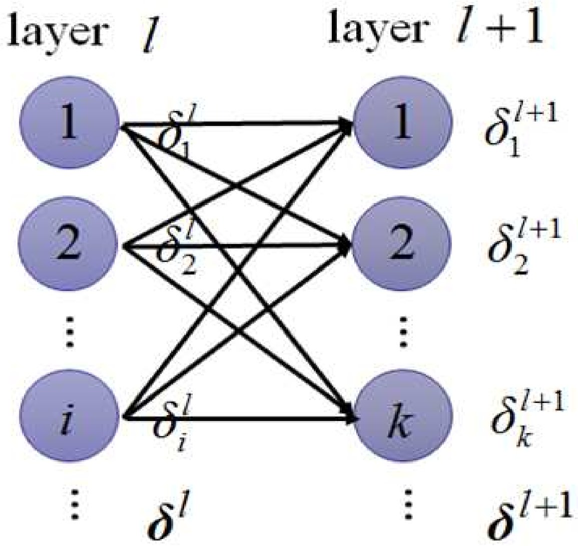
where $\odot$: element-wise multiplication

$$\sigma'\left(\boldsymbol{z}^L\right) = \begin{bmatrix} \sigma'\left(z_1^L\right) \\ \sigma'\left(z_2^L\right) \\ \vdots \\ \sigma'\left(z_m^L\right) \\ \vdots \end{bmatrix}, \quad \nabla \varepsilon_p\left(\boldsymbol{o}_p\right) = \begin{bmatrix} \partial \varepsilon_p / \partial o_{p1} \\ \partial \varepsilon_p / \partial o_{p2} \\ \vdots \\ \partial \varepsilon_p / \partial o_{pm} \\ \vdots \end{bmatrix}$$

Step 2: Determine the relation between $\boldsymbol{\delta}^l$ and $\boldsymbol{\delta}^{l+1}$







$$\delta_i^l = \frac{\partial \varepsilon_p}{\partial z_i^l} = \left(\sum_k \frac{\partial \varepsilon_p}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial o_i^l}\right)\frac{\partial o_i^l}{\partial z_i^l}$$

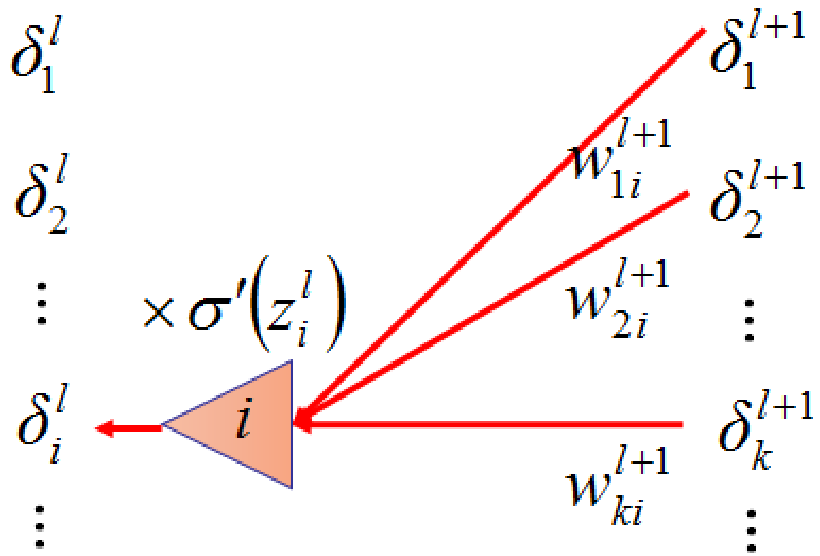
$$= \frac{\partial o_i^l}{\partial z_i^l}\left(\sum_k \frac{\partial z_k^{l+1}}{\partial o_i^l}\frac{\partial \varepsilon_p}{\partial z_k^{l+1}}\right)$$

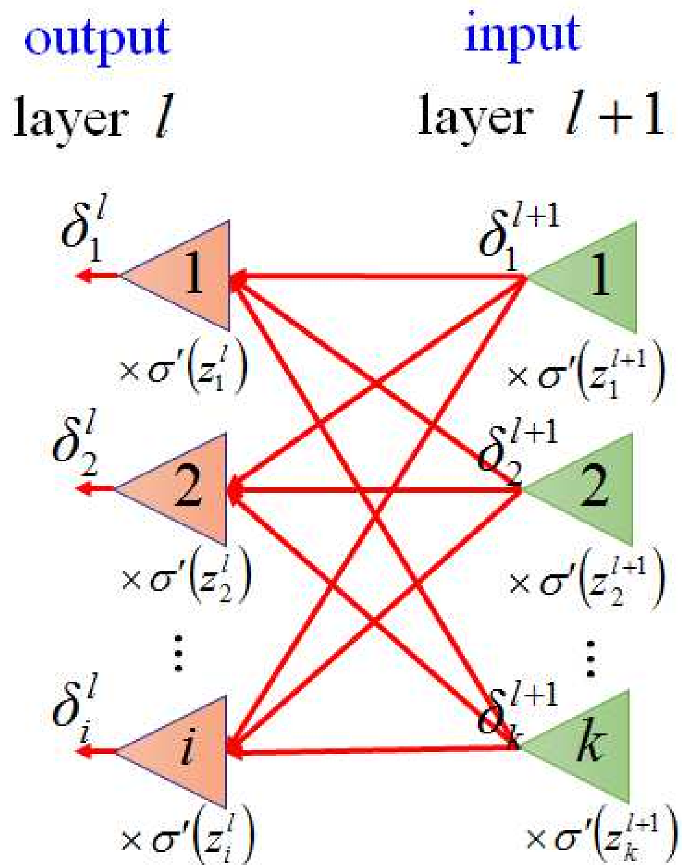
$$\delta_i^l = \frac{\partial o_i^l}{\partial z_i^l}\left(\sum_k \frac{\partial z_k^{l+1}}{\partial o_i^l}\frac{\partial \varepsilon_p}{\partial z_k^{l+1}}\right) = \frac{\partial o_i^l}{\partial z_i^l}\left(\sum_k \frac{\partial z_k^{l+1}}{\partial o_i^l}\delta_k^{l+1}\right)$$

$$= \sigma'\left(z_i^l\right)\left(\sum_k w_{ki}^{l+1}\delta_k^{l+1}\right)$$

$$\left(\begin{array}{c} o_i^l = \sigma\left(z_i^l\right),\ \dfrac{\partial o_i^l}{\partial z_i^l} = \sigma'\left(z_i^l\right) \\ z_k^{l+1} = \sum_i w_{ki}^{l+1} o_i^l + b_k^{l+1} \\ \dfrac{\partial z_k^{l+1}}{\partial o_i^l} = w_{ki}^{l+1} \end{array}\right)$$

$$\delta_i^l = \sigma'\left(z_i^l\right)\left(\sum_k w_{ki}^{l+1}\delta_k^{l+1}\right)$$





In vector form,

$$\boldsymbol{\delta}^l = \sigma'\left(\boldsymbol{z}^l\right) \odot \left(W^{l+1}\right)^T \boldsymbol{\delta}^{l+1}$$

Backward Pass

$$l = L,\ \boldsymbol{\delta}^L = \sigma'\left(z^L\right) \odot \nabla \varepsilon_p\left(\boldsymbol{o}_p\right)$$

$$l < L,\ \boldsymbol{\delta}^l = \sigma'\left(z^l\right) \odot \left(W^{l+1}\right)^T \boldsymbol{\delta}^{l+1}$$

- **Summary:** $\Delta W(t) = -\eta \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$,

$$E(\boldsymbol{\theta}) = \frac{1}{P}\sum_{p=1}^{P} \varepsilon_p(\boldsymbol{\theta}), \qquad \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \frac{1}{P}\sum_{p=1}^{P} \nabla_{\boldsymbol{\theta}} \varepsilon_p(\boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \varepsilon_p(\boldsymbol{\theta}) = \left(\cdots, \frac{\partial \varepsilon_p}{\partial w_{ij}^l}, \cdots, \frac{\partial \varepsilon_p}{\partial b_i^l} \cdots\right)^T, \quad \frac{\partial \varepsilon_p}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial \varepsilon_p}{\partial z_i^l}$$

Forward pass: $\dfrac{\partial z_i^l}{\partial w_{ij}^l}$

$$l = 1, \quad \boldsymbol{x}_p$$

$$l > 1, \quad \boldsymbol{o}^{l+1} = \sigma\left(W^{l+1}\boldsymbol{o}^l + \boldsymbol{b}^{l+1}\right)$$

Backward pass: $\dfrac{\partial \varepsilon_p}{\partial z_i^l} = \delta_i^l$

$$l = L, \quad \boldsymbol{\delta}^L = \sigma'\left(\boldsymbol{z}^L\right) \odot \nabla \varepsilon_p\left(\boldsymbol{o}_p\right)$$

$$l < L, \quad \boldsymbol{\delta}^l = \sigma'\left(\boldsymbol{z}^l\right) \odot \left(W^{l+1}\right)^T \boldsymbol{\delta}^{l+1}$$

$$\frac{\partial \varepsilon_p}{\partial w_{ij}^l} = \boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}} \boxed{\frac{\partial \varepsilon_p}{\partial z_i^l}}$$

$$\begin{cases} x_{pj} & l = 1 \\ o_j^{l-1} & l > 1 \end{cases}$$

$$\delta_i^l$$

Forward Pass

$$\boldsymbol{z}^1 = W^1 \boldsymbol{x}_p + \boldsymbol{b}^1$$

$$\boldsymbol{o}^1 = \sigma\left(\boldsymbol{z}^1\right)$$

$$\cdots\cdots\cdots\cdots\cdots$$

$$\boldsymbol{z}^{l-1} = W^{l-1} \boldsymbol{o}^{l-2} + \boldsymbol{b}^{l-1}$$

$$\boldsymbol{o}^{l-1} = \sigma\left(\boldsymbol{z}^{l-1}\right)$$

Backward Pass

$$\boldsymbol{\delta}^L = \sigma'\left(\boldsymbol{z}^L\right) \odot \nabla \varepsilon_p\left(\boldsymbol{o}_p\right)$$

$$\boldsymbol{\delta}^{L-1} = \sigma'\left(\boldsymbol{z}^{L-1}\right) \odot \left(W^L\right)^T \boldsymbol{\delta}^L$$

$$\cdots\cdots\cdots\cdots\cdots$$

$$\boldsymbol{\delta}^l = \sigma'\left(\boldsymbol{z}^l\right) \odot \left(W^{l+1}\right)^T \boldsymbol{\delta}^{l+1}$$

$$\cdots\cdots\cdots\cdots\cdots$$

12.3 Improving Training Convergence

- Momentum $w_i^{t+1} = w_i^t + \eta \Delta w_i^t$

$$\Delta w_i^t = \alpha \Delta w_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i}$$

- Adaptive momentum

e.g., Adam (Adaptive moments):

$$s_i^t = \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i}, \quad \tilde{s}_i^t = \frac{s_i^{t-1}}{1-\alpha^t}$$

$$r_i^t = \rho\, r_i^{t-1} + (1-\rho)\left|\partial E^t / \partial w_i\right|^2, \quad \tilde{r}_i^t = \frac{r_i^t}{1-\rho^t}$$

$$\Delta w_i^t = -\eta \frac{\tilde{s}_i^t}{\sqrt{\tilde{r}_i^t}}.$$

t : index for s_i^t and power for α^t and ρ^t.

- Adaptive learning rate

i) η is kept large when learning takes place and decreases when learning slows down

$$\eta(t+1) = \eta(t) + \Delta\eta(t),$$

$$\Delta\eta(t) = \begin{cases} +a & \text{if } E^{t+1} < E^{t} \\ -b & \text{otherwise} \end{cases}$$

i.e., increase η if the error decreases and decrease η if the error increases.

ii) η can be adapted separately for each weight.

e.g., AdaGrad, RMSProp

$$\Delta w_i^t = -\eta_i^t \frac{\partial E^t}{\partial w_i}, \ \eta_i^t = -\frac{\eta}{\sqrt{r_i^t}},$$

$$r_i^t = \rho\, r_i^{t-1} + (1-\rho)\left|\partial E^t / \partial w_i\right|^2$$

where

r_i^t is the accumulated past gradients.

- Weight Decay

  -- A large weight increases the complexity of the model.

  -- Initially, weights are assigned values close to zero.

  -- As learning proceeds, more weights move away from zero.

  -- Weight decay is to force a weight toward zero so as to reduce the complexity of the model.

Example: Introduce $-\lambda w_i$ into the weight updating rule

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i \Leftrightarrow \text{Introduce } \frac{\lambda}{2}\sum_{i,j} w_{i,j}^2$$

into error function $E' = E + \frac{\lambda}{2}\sum_{i,j} w_{i,j}^2$

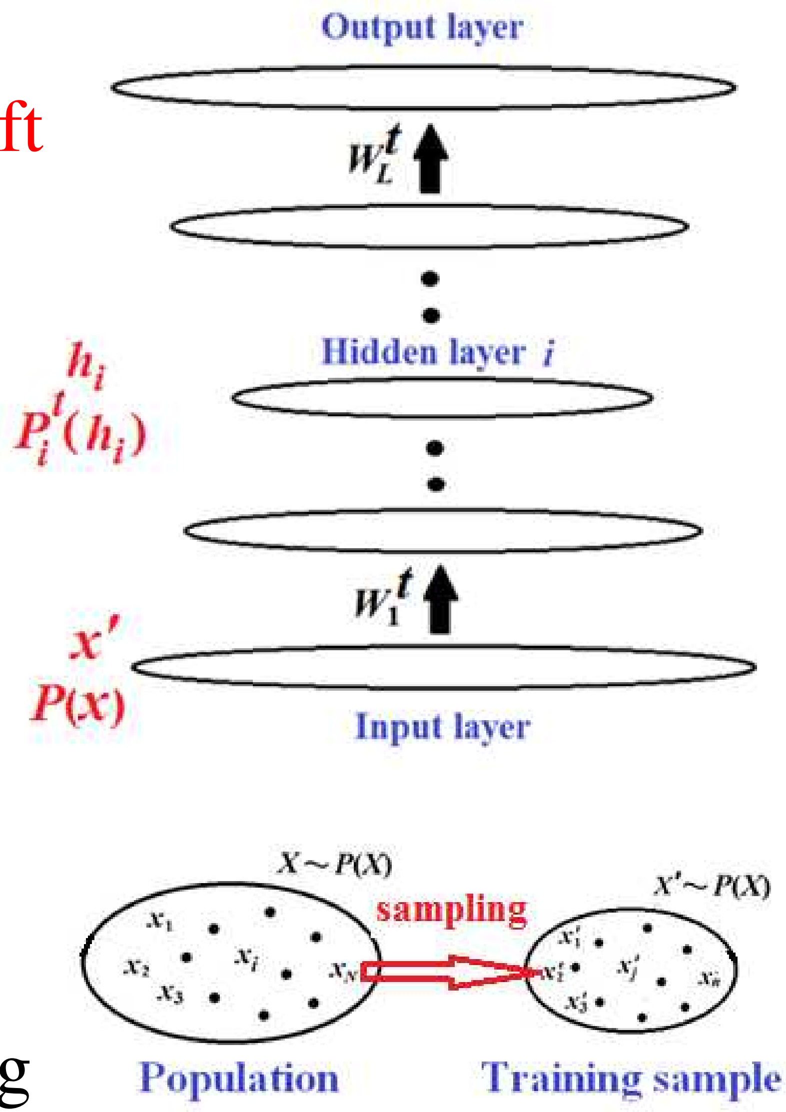
$$E' = E + \frac{\lambda}{2}\sum_{i,j} w_{i,j}^2 : \ L2 \text{ regularization}$$

$$E' = E + \frac{\lambda}{2}\sum_{i,j} \left| w_{i,j} \right| : \ L1 \text{ regularization}$$

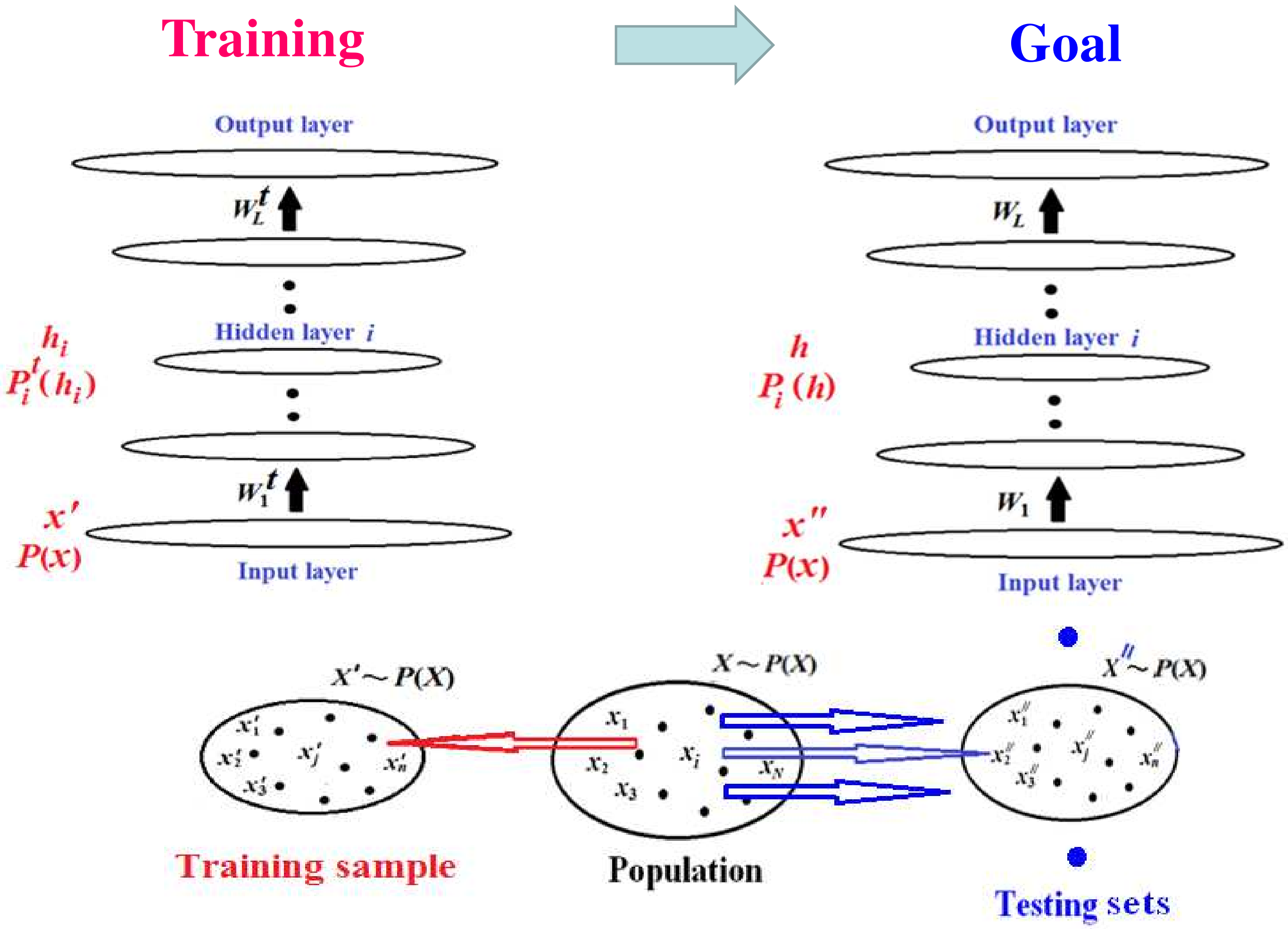
- Batch Normalization

(A) Reduces covariate shift (or class imbalance, sample selection, bias)

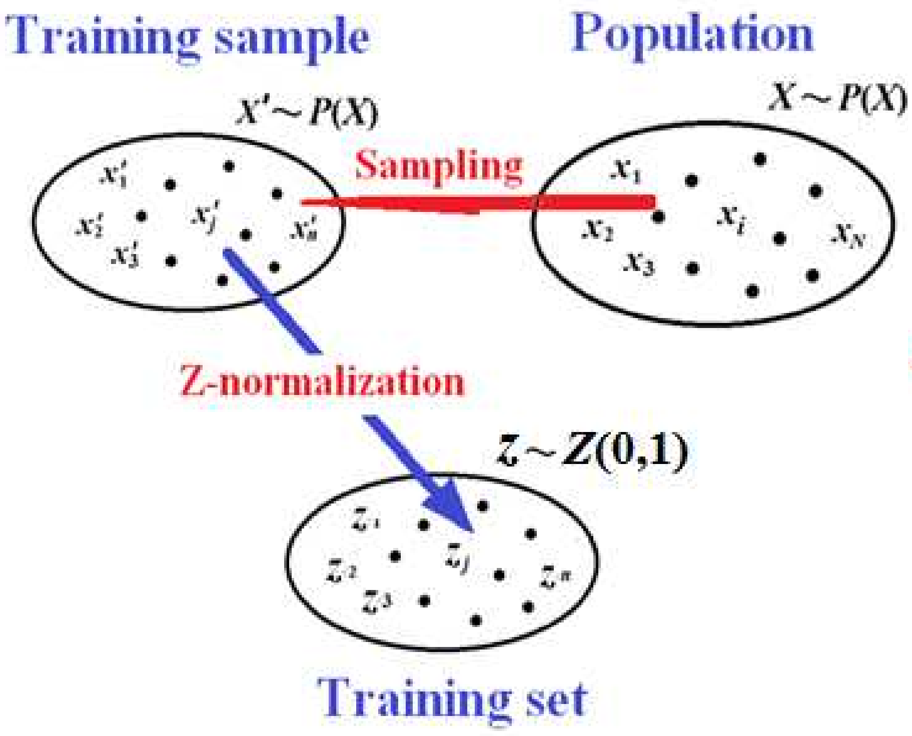
Covariate shift (CS): the change in the distribution of neural activations due to the change in network parameters during training

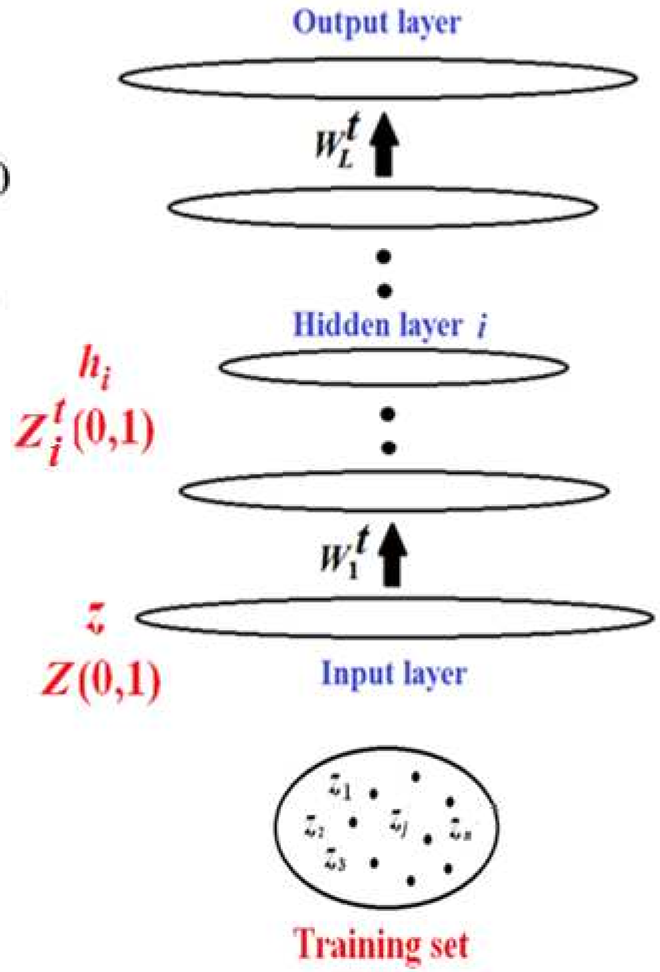


CS slows down training leading to the requirements of lower learning rate and careful parameter initialization.



Remedy: Apply z-normalization every iteration to the input of each layer except the input layer





Z-normalization

Given a batch of vectors, $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots\cdots, \boldsymbol{x}_N\}$

where $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \cdots, x_{id})$

Matrix representation:

$$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^T = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & & & \\ x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix}$$

z-normalization: $z_{ij} = \dfrac{x_{ij} - m_j}{\sigma_j} \sim Z(0,1)$

where $Z(0,1)$: the distribution has zero (0) mean and unit (1) variance.

Map z_{ij} to have arbitrary mean and scale by

$$\tilde{z}_{ij} = \alpha_{ij} z_{ij} + \beta_{ij}$$

Advantages: Values of different attributes of input vectors can be spread in the same scale through z-normalization, so do corresponding weights in the same scale. As a result, the same learning rate can be used.

(B) Reduce dependence of gradients

Current neural networks concerning deep learning conduct the backpropagation technique during training.

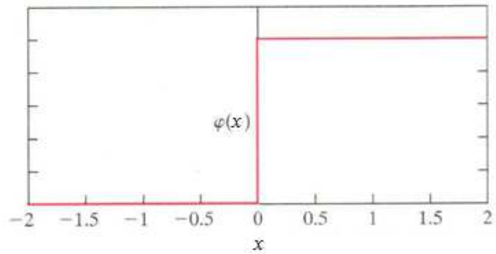
The backpropagation technique primarily relies on gradient decent approach.

Gradient computation requires performing derivatives.

Sigmoidal functions $\varphi(\cdot)$ are often employed to serve activation functions of neural networks, e.g.,

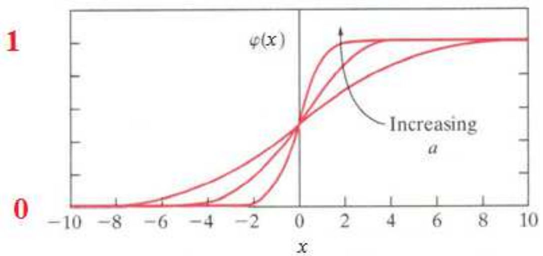
1. Threshold function:

$$\varphi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$



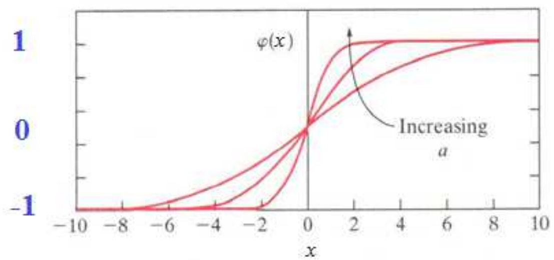
2. Logistic function

$$\varphi(x) = \frac{1}{1+\exp(-ax)}$$



3. Hyperbolic function

$$\varphi(x) = \tanh(ax)$$



4. Softsign function

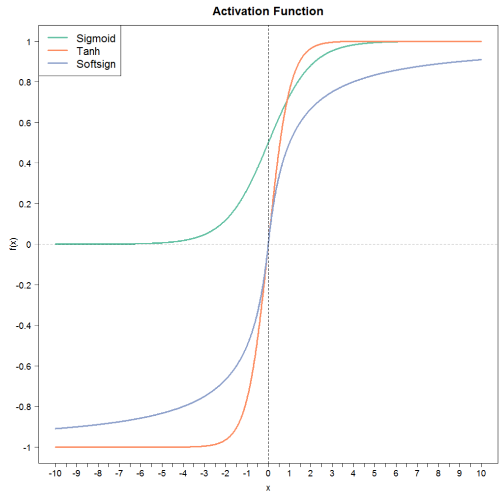
$$\varphi(x) = \frac{x}{1+|x|}$$

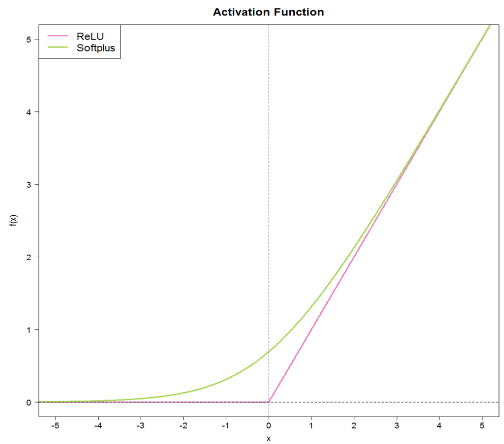
5. Softplus function

$$\varphi(x) = \ln(1+e^{x})$$

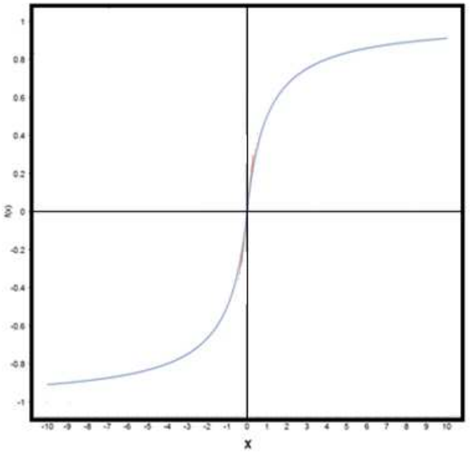
6. ReLU function

$$\varphi(x) = \max(\theta, x)$$

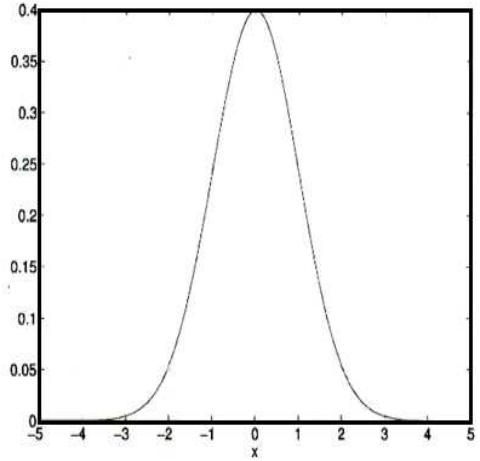




The derivatives of sigmoidal functions suffer from only having significant outcomes around the origin.



Activation functions

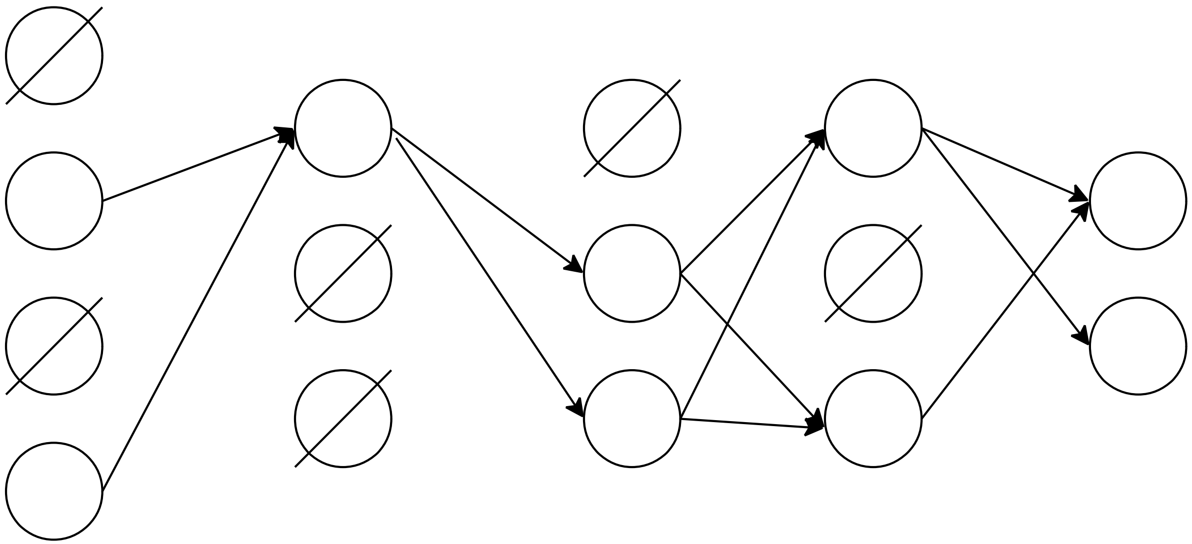


Derivative

In order to compensate for the aforementioned difficulty, input data x are z-normalized

$$z = \frac{x-\mu}{\sigma} \text{ and } z \sim Z(0,1).$$

- Dropout -- Randomly discard inputs or hidden units of a neural network during training with small batches, i.e., minibatches.



- Transfer learning – use training results of another network that has been trained for a similar task.

- Pretrain –

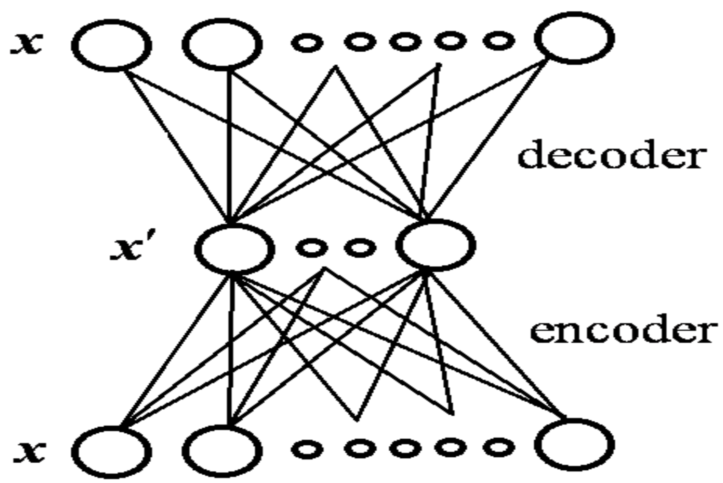
  Autoencoder – reconstruct its input at the output.

  Tacked autoencoder – 2 encoders back to back.

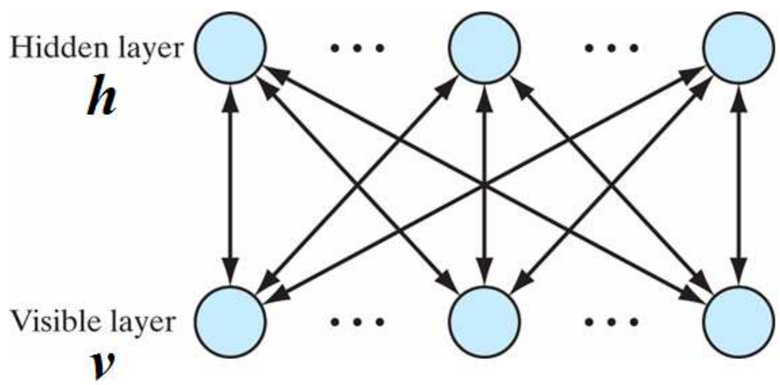
  Deep autowncoders – multiple hidden layers.

  Restricted Boltzmann machine (RBM)

Autoencoder



RBM



- Regularization -- Any modification made to a learning algorithm that intended to reduce its generalization (test) error but not its training error, e.g.,
  1. Put constraints on a machine learning model
  2. Add restrictions on the parameter values
  3. Add extra terms in the objective function
  4. Encode specific kinds of prior knowledge
  5. Express generic preferences
  6. Use ensemble methods
  7. Multiple hypotheses

12.4 Tuning the Network Structure

- Destructive approach --

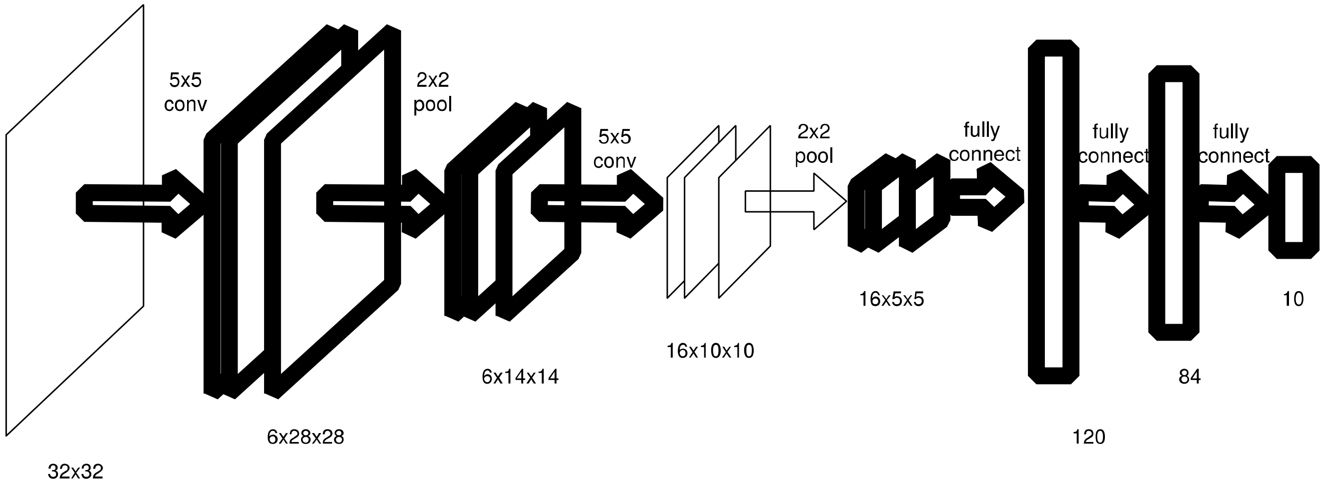
  Start with a **large** network gradually **prune** unnecessary weights

- Constructive approach --

  Start from a **small** network and progressively **add** units and connections.

- **Example DNNs**

LeNet-5:



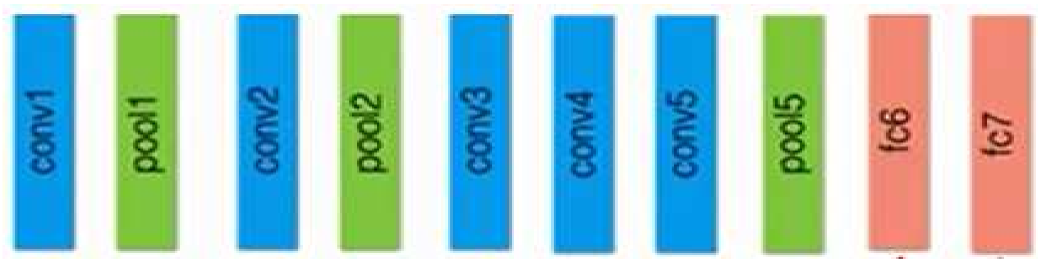
Input data: 32 x 32 image

Convolution filters: 6 5 x 5 kernels

2 x 2 average pooling

Output function: Euclidean radial basis function

AlexNet:



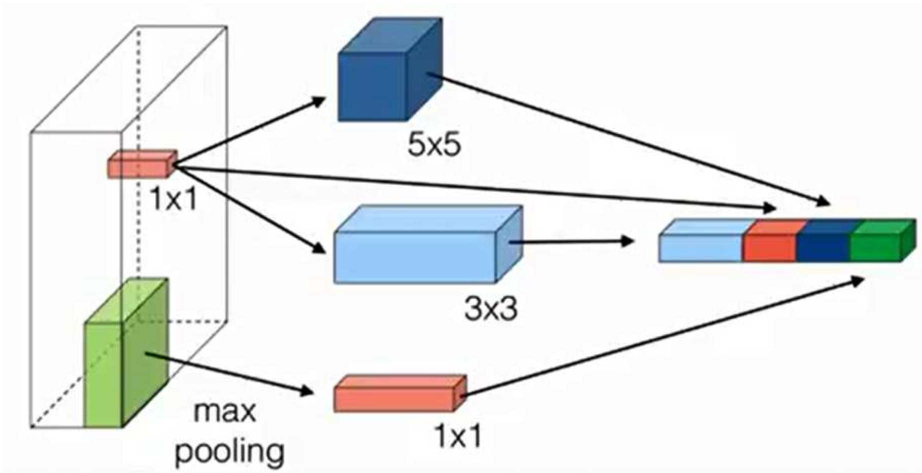
11 layers: 5 convolution layers, 3 pooling layers, 2 fully connected layers

Activation function: ReLU

Output function: softmax

GoogleNet:

Different sizes of filters even in the same layer



VGG: 16 layers

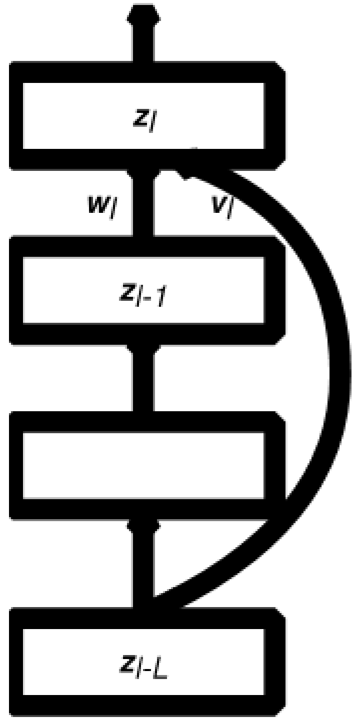
3 conv layers (3 by 3 filter with stride 1 and pad 1)

Pool layers (2 by 2 max pooling with stride 2 and pad 0)

Layer	Output shape
Input	(224, 224, **3**)
CONV (3x3x64)	(224, 224, **64**)
CONV (3x3x64)	(224, 224, **64**)
POOL (2x2)	(112, 112, **64**)
CONV (3x3x128)	(224, 224, **128**)
CONV (3x3x128)	(224, 224, **128**)
POOL (2x2)	(56, 56, **128**)
CONV (3x3x256)	(56, 56, **256**)
CONV (3x3x256)	(56, 56, **256**)
CONV (3x3x256)	(56, 56, **256**)
POOL (2x2)	(28, 28, **256**)
CONV (3x3x256)	(28, 28, **512**)
CONV (3x3x256)	(28, 28, **512**)
CONV (3x3x256)	(28, 28, **512**)
POOL (2x2)	(14, 14, **512**)
CONV (3x3x512)	(14, 14, **512**)
CONV (3x3x512)	(14, 14, **512**)
CONV (3x3x512)	(14, 14, **512**)
POOL (2x2)	(7, 7, **512**)
AFFINE (4096 units)	(4096, 1)
AFFINE (4096 units)	(4096, 1)
AFFINE (100 units)	(100, 1)

Residual Networks:

-- A hidden unit may be connected not only to the units in its preceding layer, but also to units in a layer much earlier.

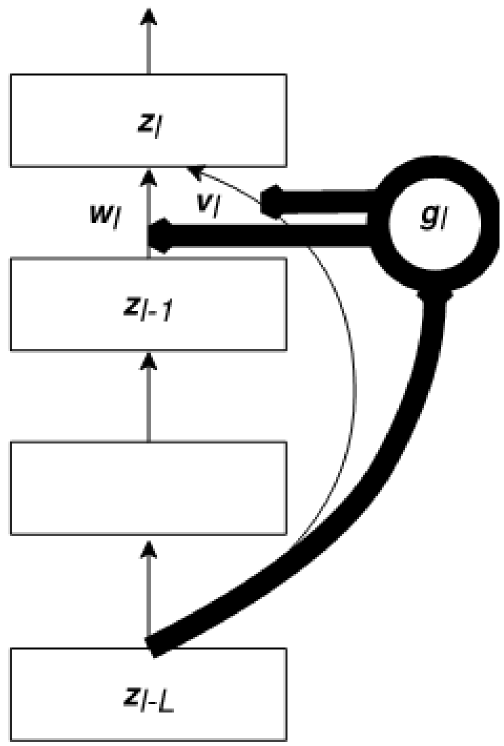


Advantage -- Avoid vanishing and exploding gradients during training .

$$z_{lh} = f\left(\sum_i w_{l,h,i} z_{l-1,i} + \sum_j v_{l,h,j} z_{l-L,j}\right)$$

Highway networks:

-- Take weighted sum of inputs from long and short paths



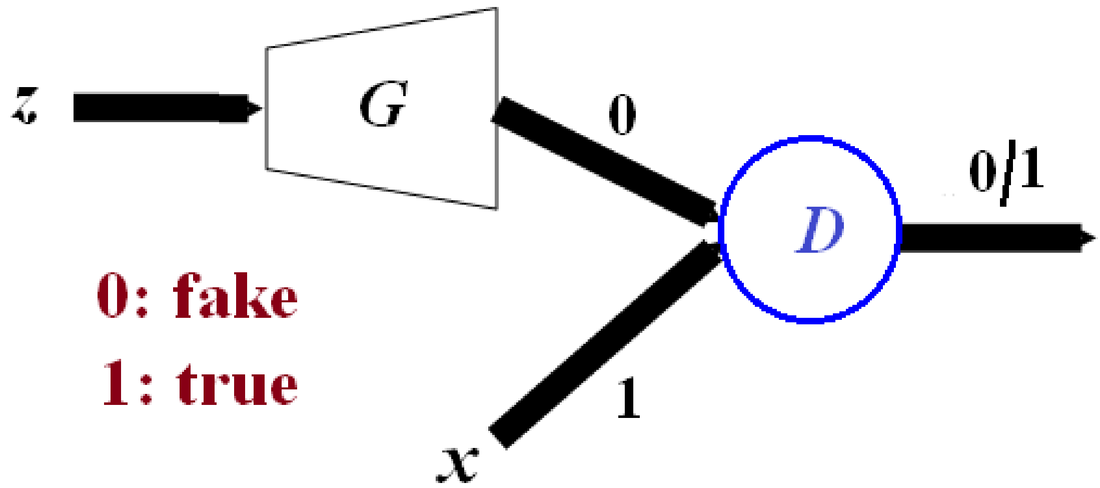
$$z_{lh} = f\left(g_{lh} \sum_i w_{lhi} z_{l-1,i} + (1 - g_{lh}) \sum_j v_{lhj} z_{l-L,j} \right)$$

Generative Adversarial Network (GAN)

A GAN is composed of two agents: *G* (generator) and *D* (discriminator).

G synthesizes fake examples

D tells apart true and fake examples



$X = \{\boldsymbol{x}^t\}_t$: training examples drawn from probability distribution $p(\boldsymbol{x})$

G takes a $\boldsymbol{z}$ as input and gives out a fake $G(\boldsymbol{z})$.

Criterion: $$\sum_t \log D(\boldsymbol{x}^t) + \sum_{\boldsymbol{z} \sim p(\boldsymbol{z})} \log \left(1 - D(G(\boldsymbol{z}))\right)$$

which is maximized by D and minimized by G.

D is trained to output a large value for a true instance and a small value for a fake instance.

G is trained to generate fakes for which D gives large values.

During training, G gets better in generating fakes, D gets better in discriminating them, which in turn forces G to generate much better fakes, etc..

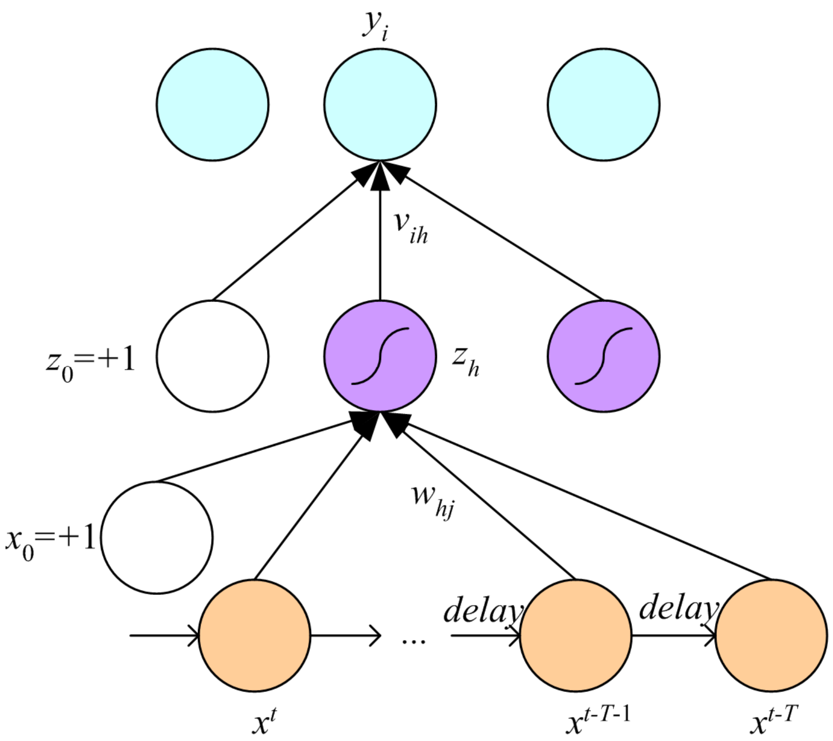
G and D are trained alternately. For a fixed G, the weights of D are updated so that the loss $\sum_t \log D(\mathbf{x}^t)$ is maximized. Then, fix D and update the weights of G to minimize $\sum_{\mathbf{Z} \sim p(\mathbf{Z})} \log\left(1 - D(G(\mathbf{z}))\right)$

Another criterion: $\sum_t E[D(\mathbf{x}^t)] - \sum_{\mathbf{Z} \sim p(\mathbf{Z})} E[D(G(\mathbf{z}))]$

which is maximized by D and minimized by G.

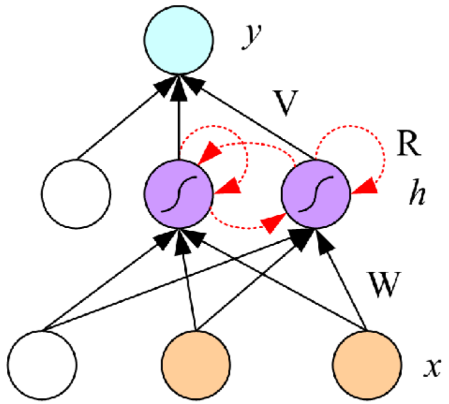
Time-Delay Neural Networks:

-- Previous input are delayed so as to synchronize with the recent inputs and all are fed together as input to the system.

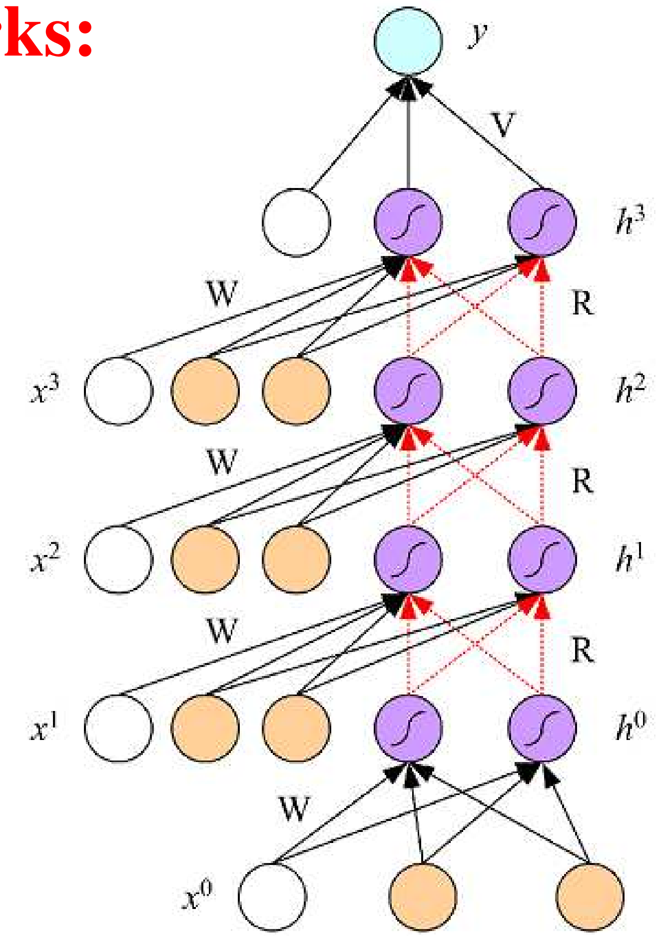


Recurrent Neural Networks:

Examples:



(a)

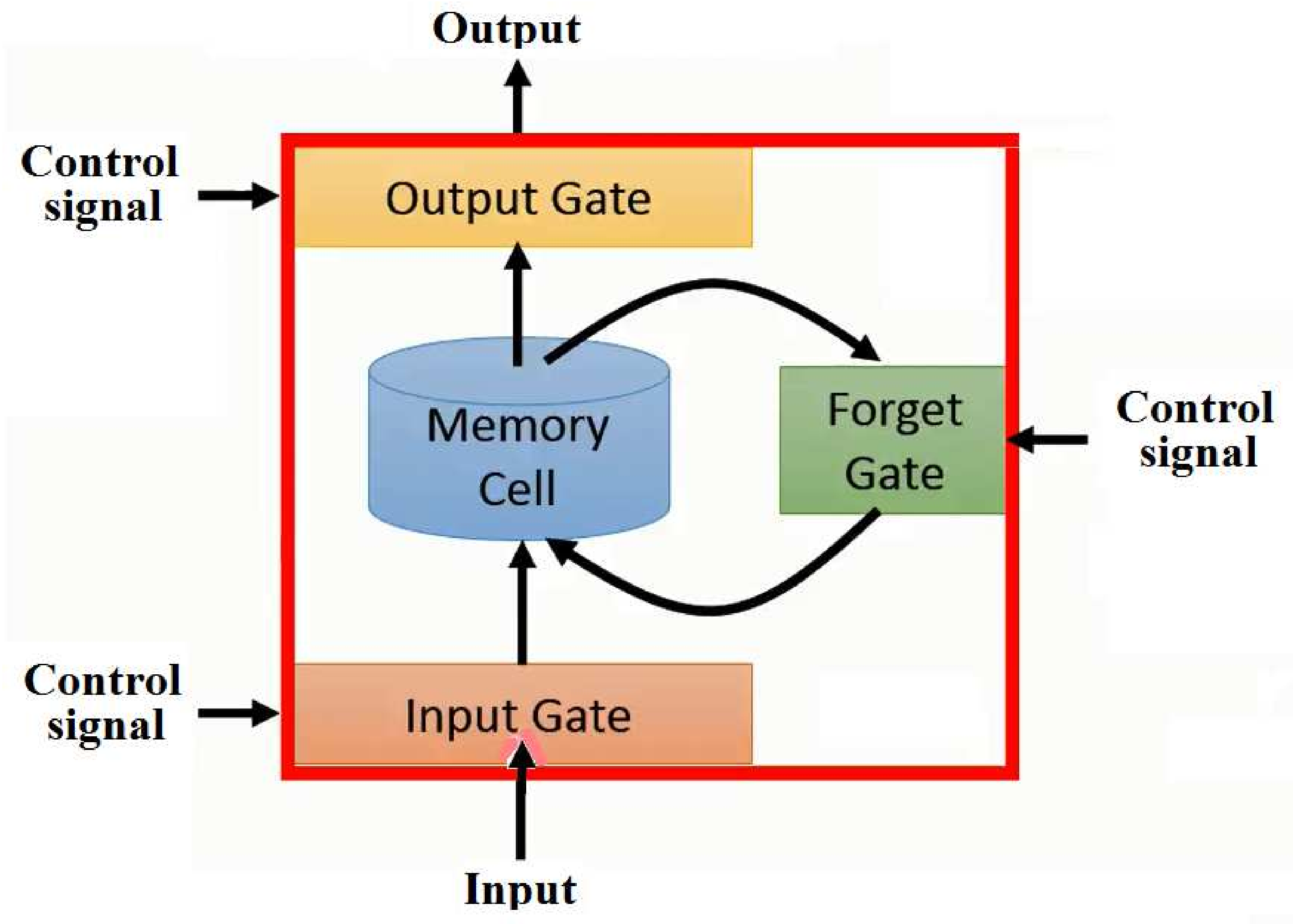


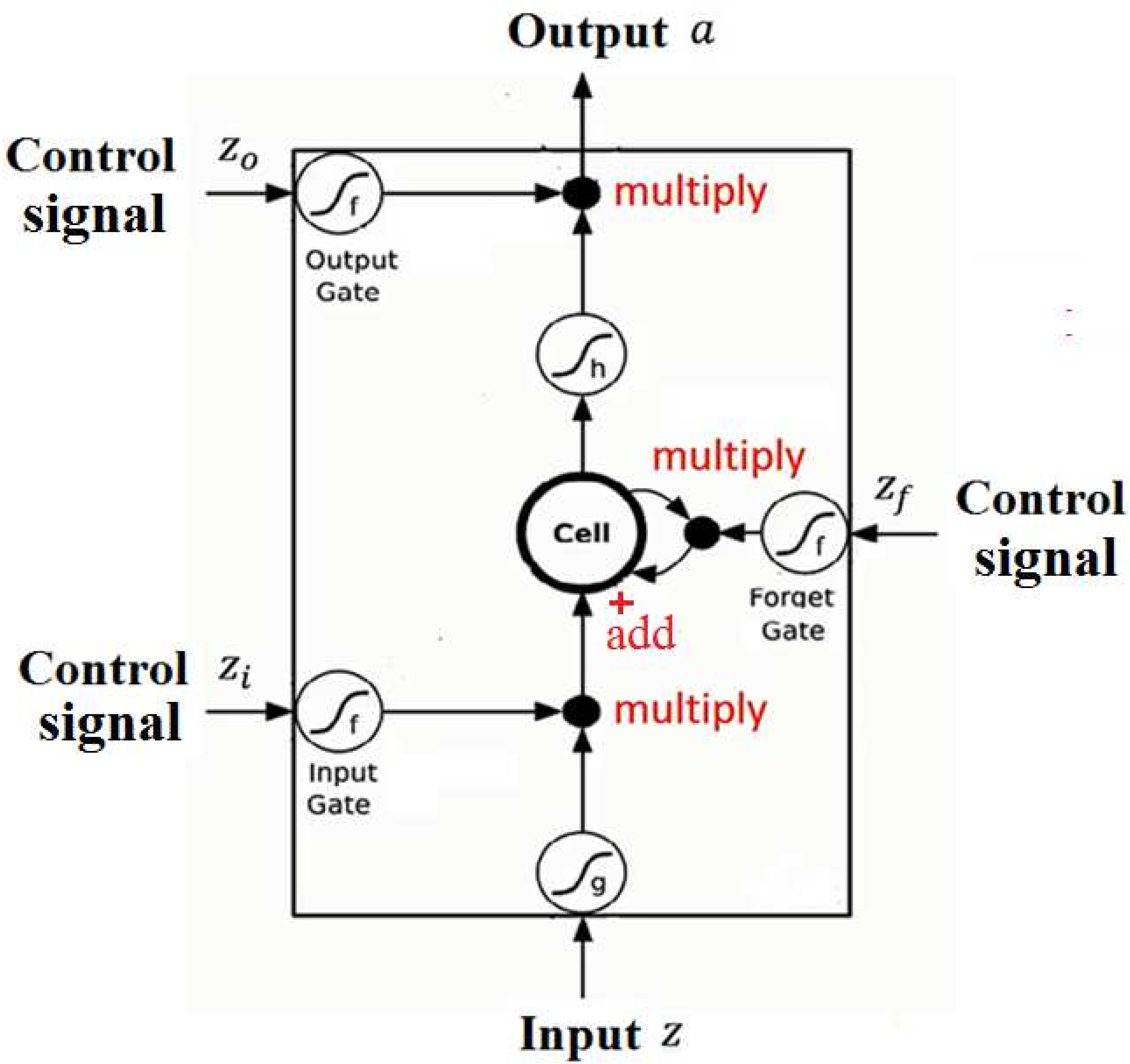
(b)

$$z_h^t = f\left(\sum_{j=0}^{d} w_{hj} x_j^t + \sum_{l=1}^{H} r_{hl} z_l^{t-1}\right) \qquad y^t = g\left(\sum_{h=0}^{H} v_h z_h^t\right)$$

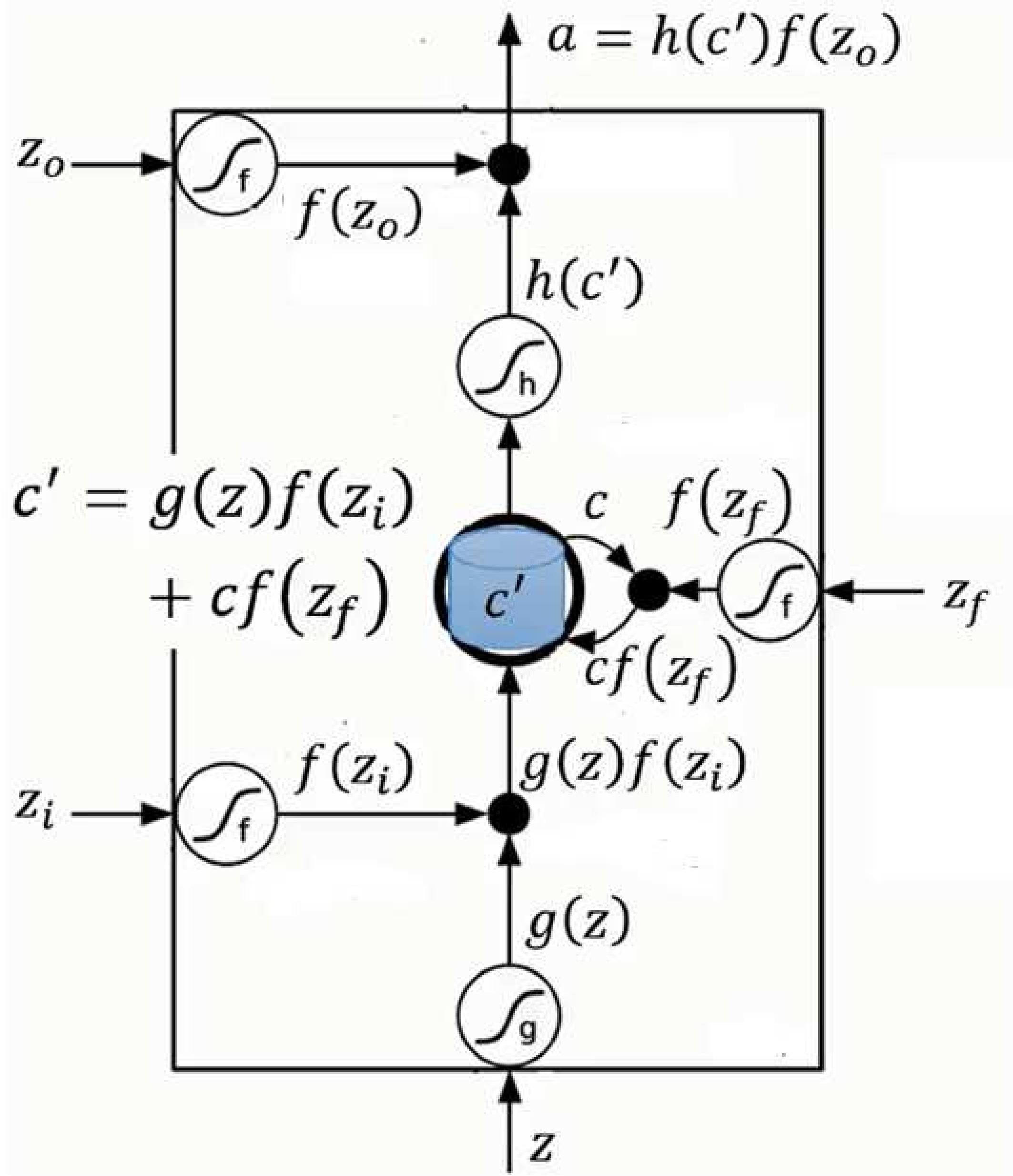
Long Short-Term Memory (LSTM) Unit:

– A small MLP with memory and gate units





Output a
Control signal
z_o
multiply
Output Gate
multiply
Cell
Control signal
z_f
Forget Gate
+ add
Control signal
z_i
multiply
Input Gate
Input z



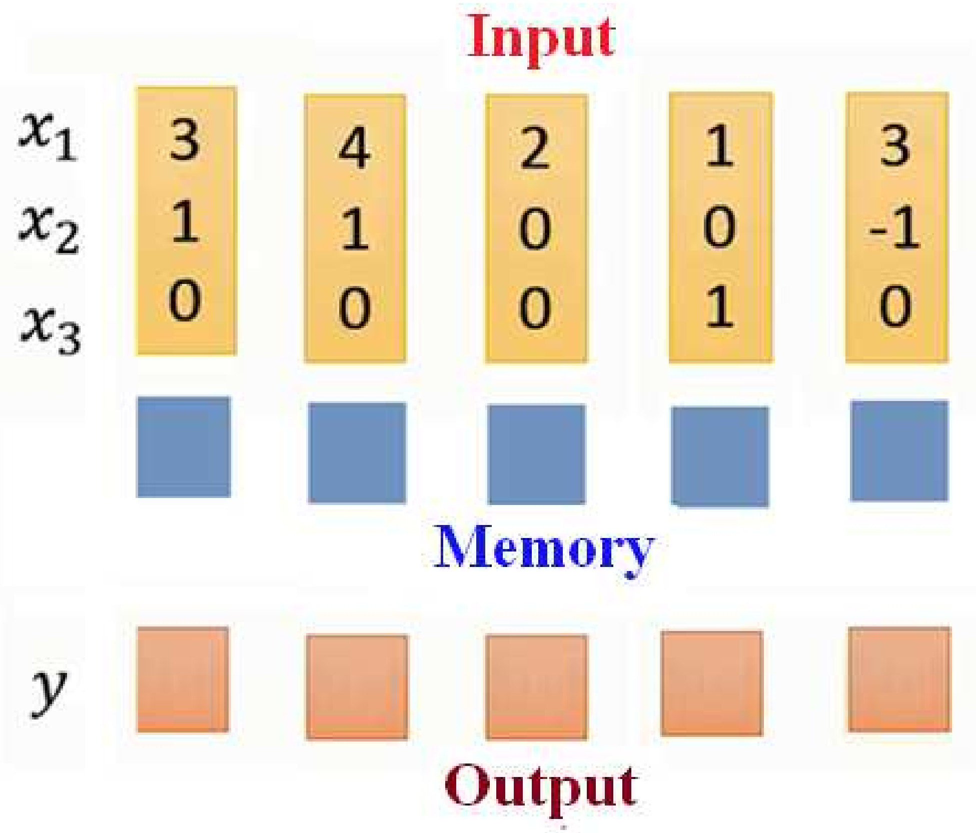
$a = h(c')f(z_o)$
z_o
$f(z_o)$
$h(c')$
$c' = g(z)f(z_i)$
$+ cf(z_f)$
c
$f(z_f)$
c'
z_f
$cf(z_f)$
$f(z_i)$
$g(z)f(z_i)$
z_i
$g(z)$
z

Example:

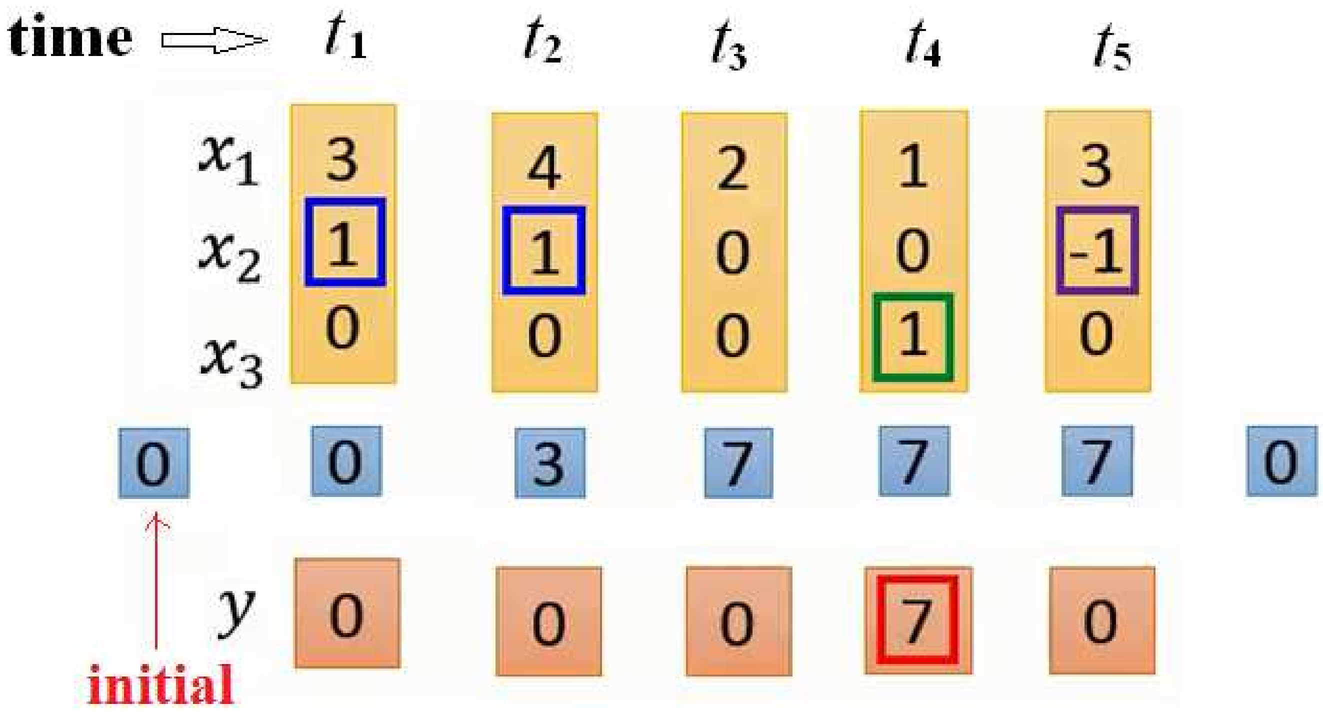
When $x_2 = 1$, add the numbers of x_1 into the memory

When $x_2 = -1$, reset the memory

When $x_3 = 1$, output the number in the memory



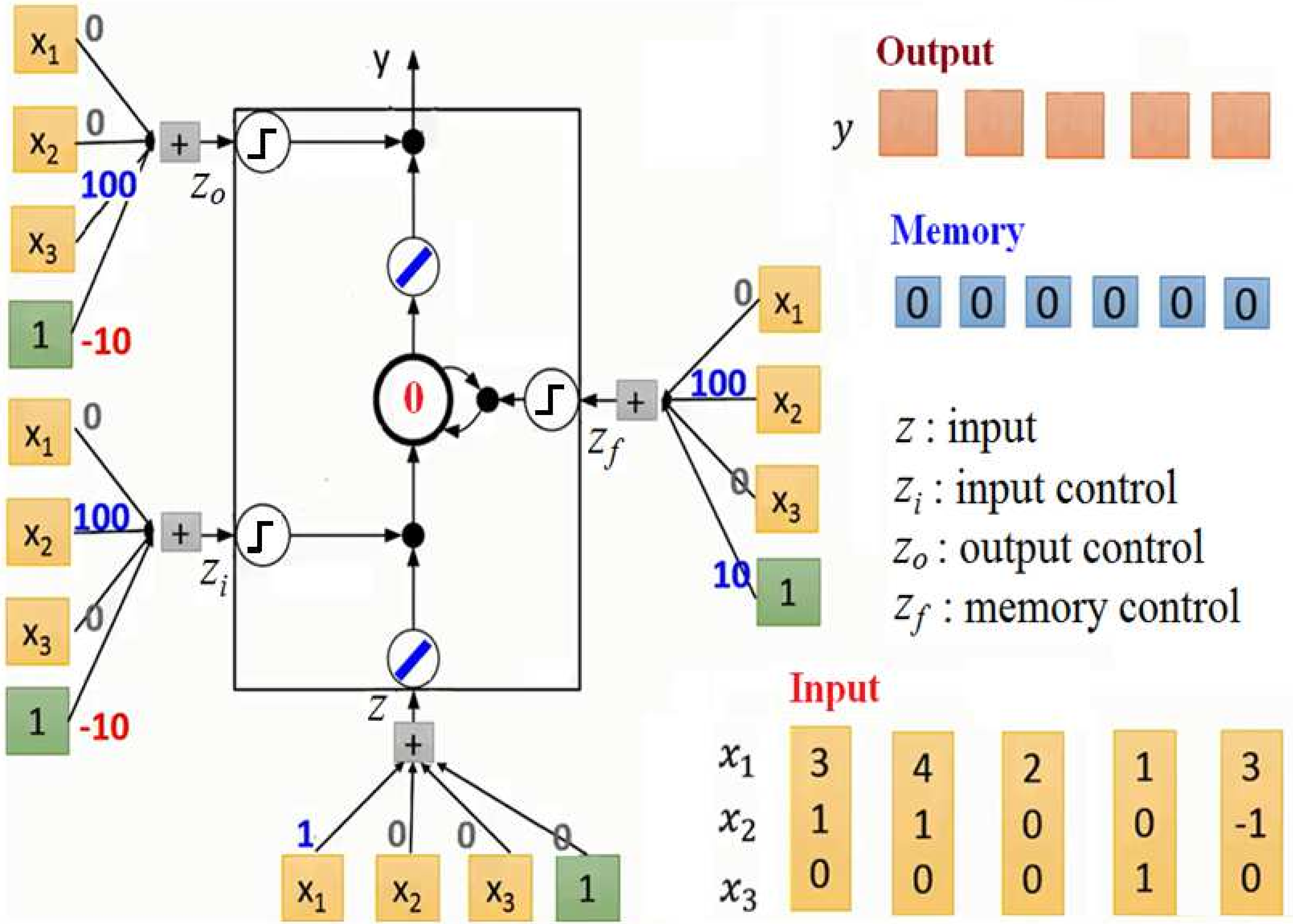
What is this process doing?



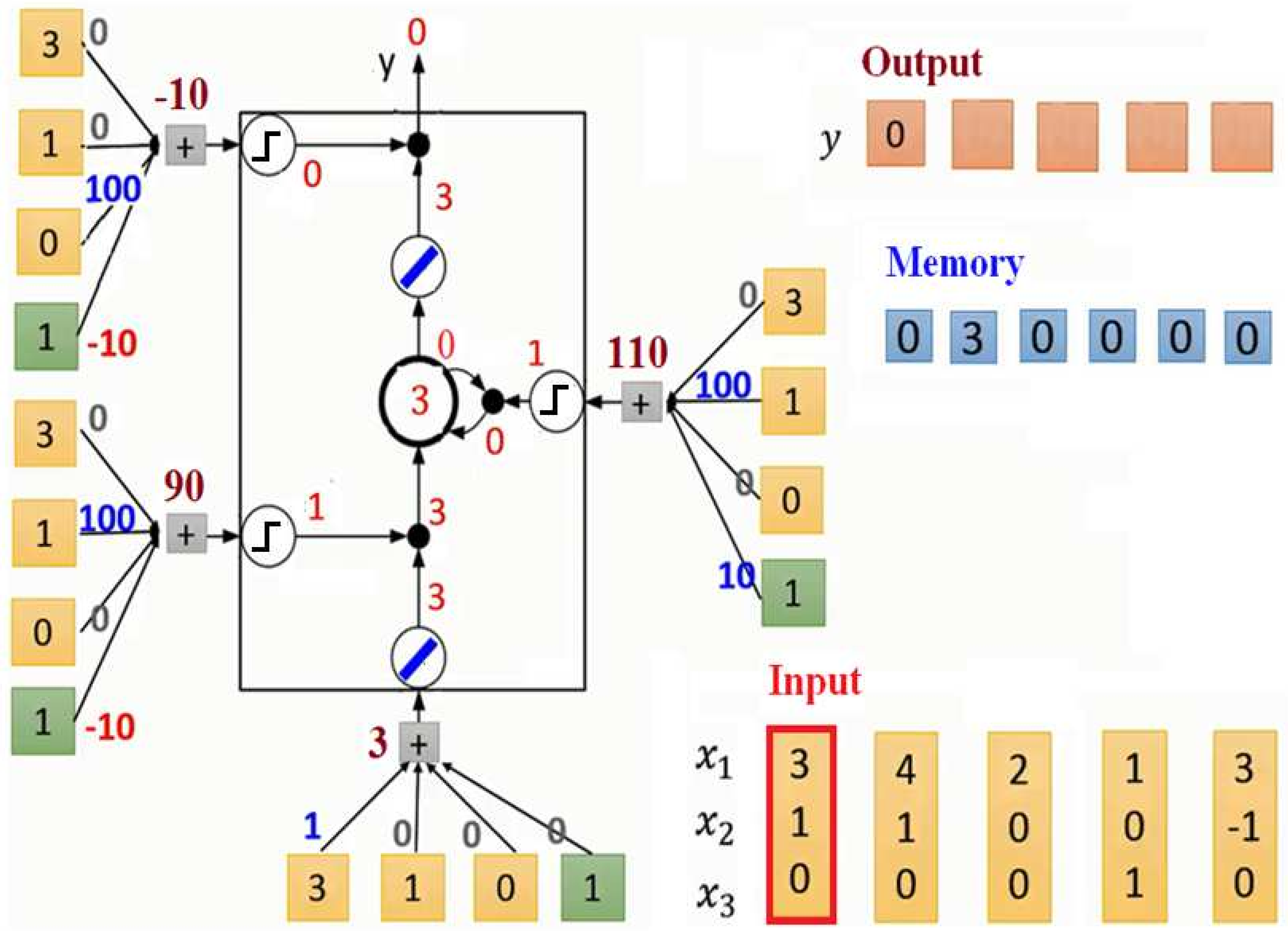
When $x_2 = 1$, add the numbers of x_1 into the memory

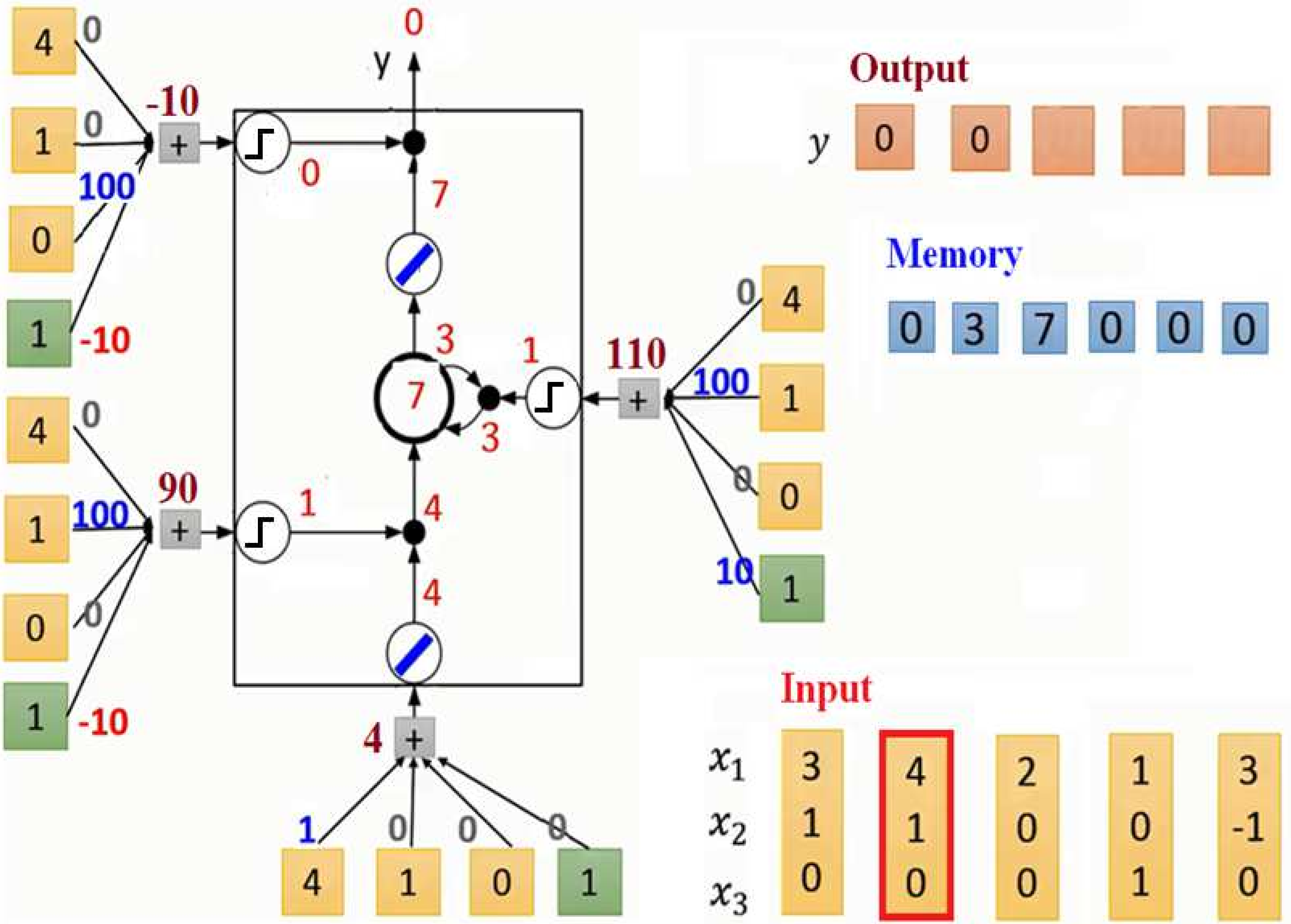
When $x_2 = -1$, reset the memory

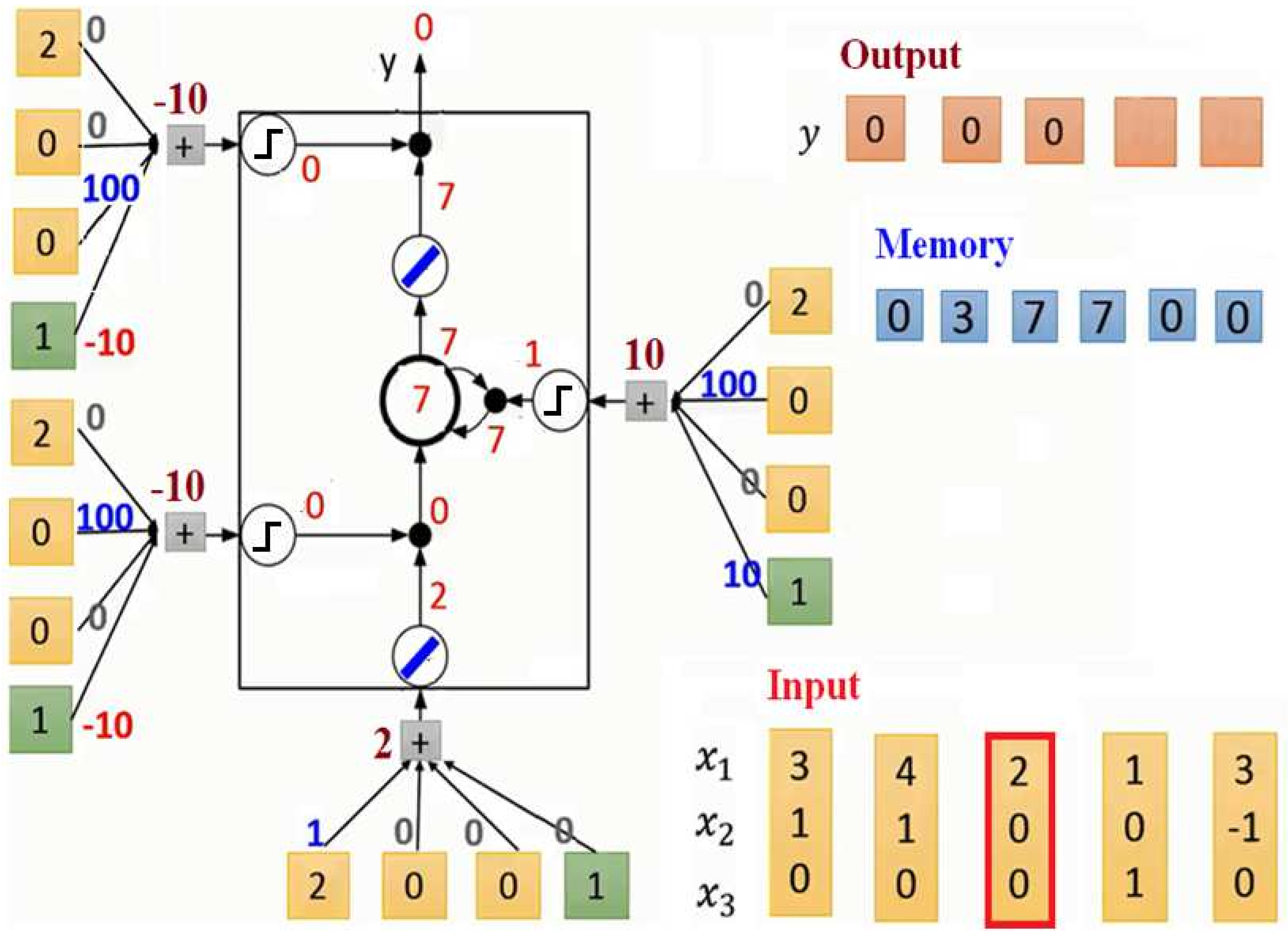
When $x_3 = 1$, output the number in the memory



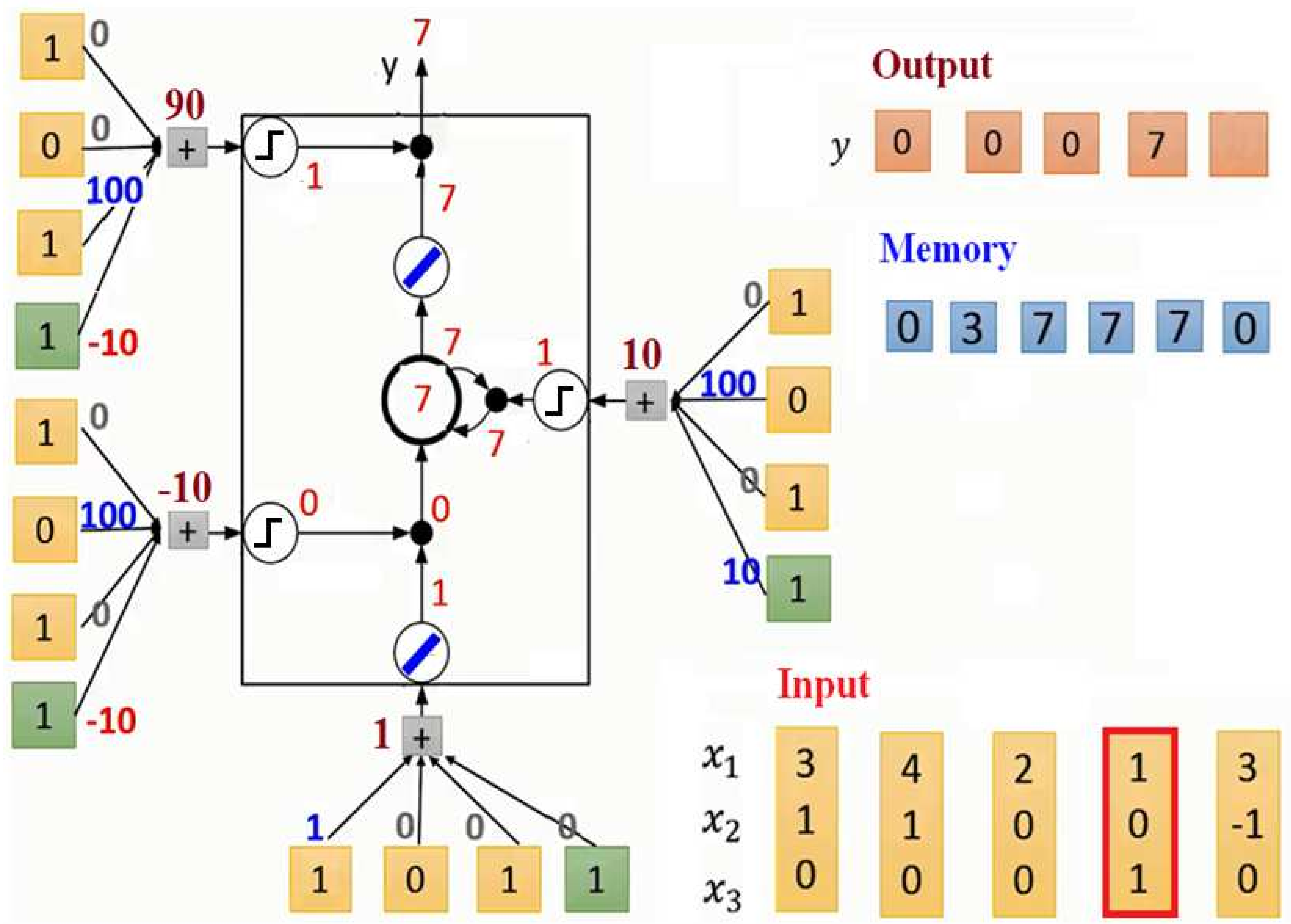
Output
y
Memory
0 0 0 0 0 0
z : input
z_i : input control
z_o : output control
z_f : memory control
Input
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0
x_1 x_2 x_3 1
0 0 100 -10
0 100 0 -10
0 100 0 10
1 0 0 0
z_o
z_i
z_f
z
y


Output
y
Memory
Input
x_1
x_2
x_3

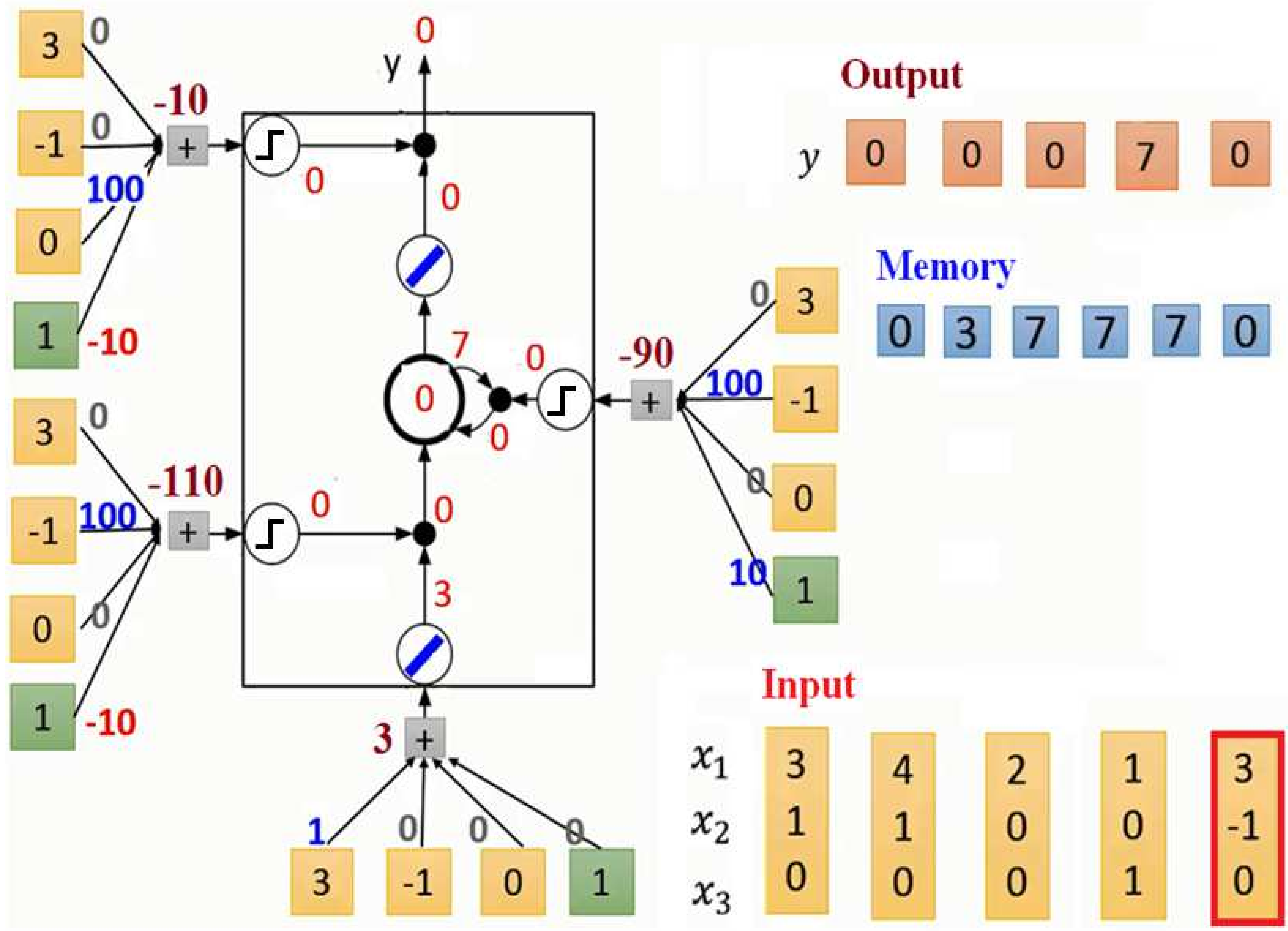

Output
y
Memory
Input
x_1
x_2
x_3



Output
y
Memory
Input
x_1
x_2
x_3



Output
y
0
0
0
7
Memory
0
3
7
7
7
0
Input
x_1
x_2
x_3
3 1 0
4 1 0
2 0 0
1 0 1
3 -1 0



Output
y
Memory
Input
x_1
x_2
x_3
-10
-110
-90
100
10
-10