Chapter Five

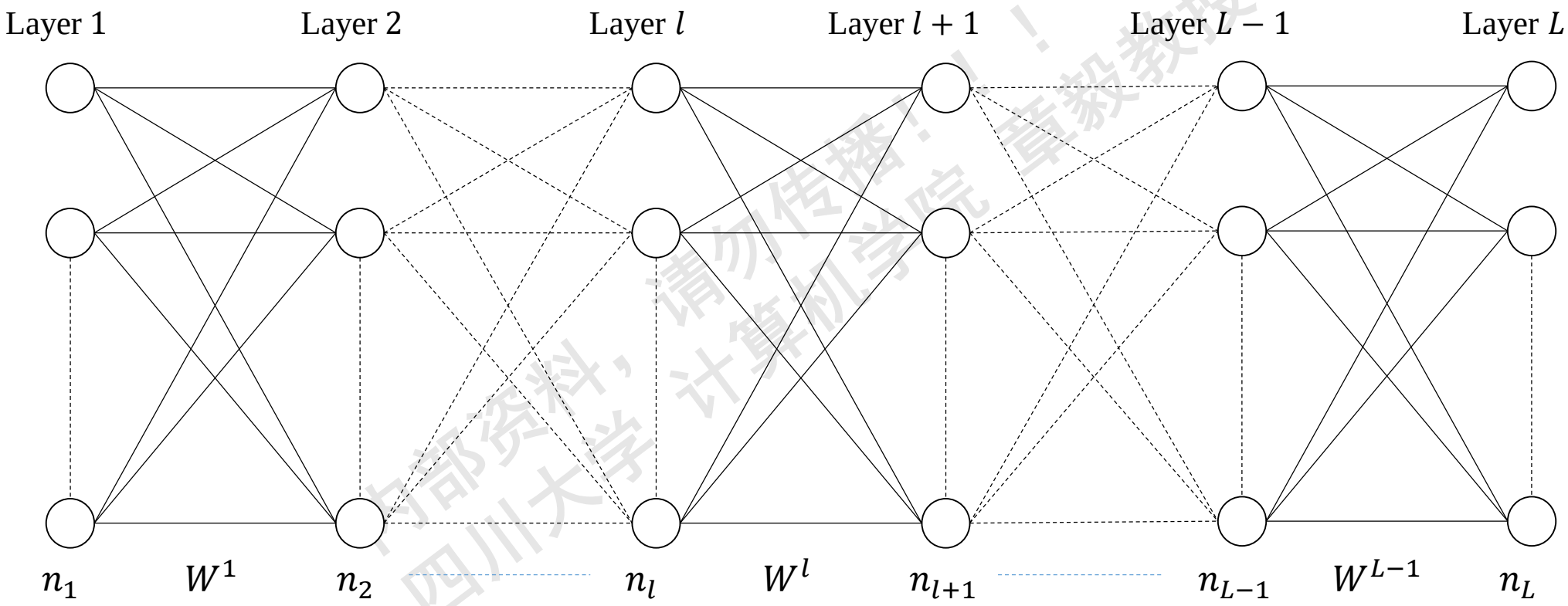
On Some Problems of BP

Zhang Yi, *IEEE Fellow*
Autumn 2018

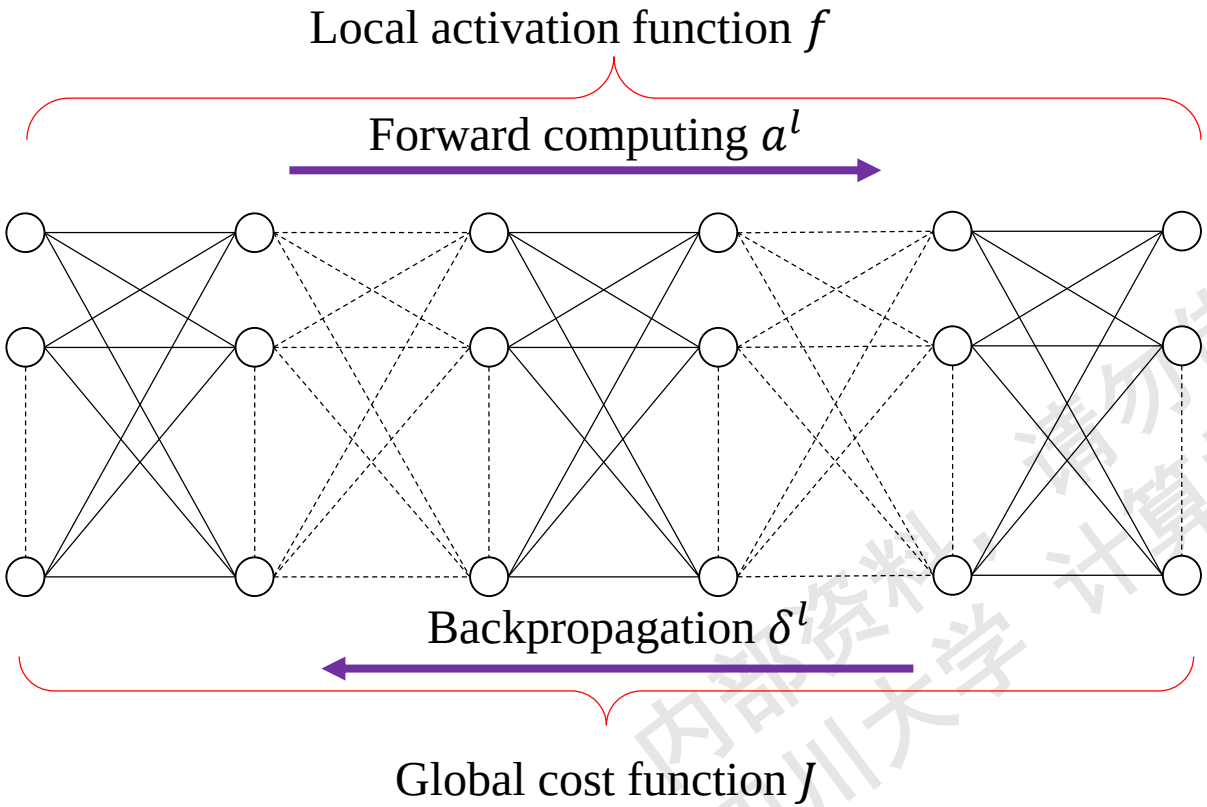
Outline

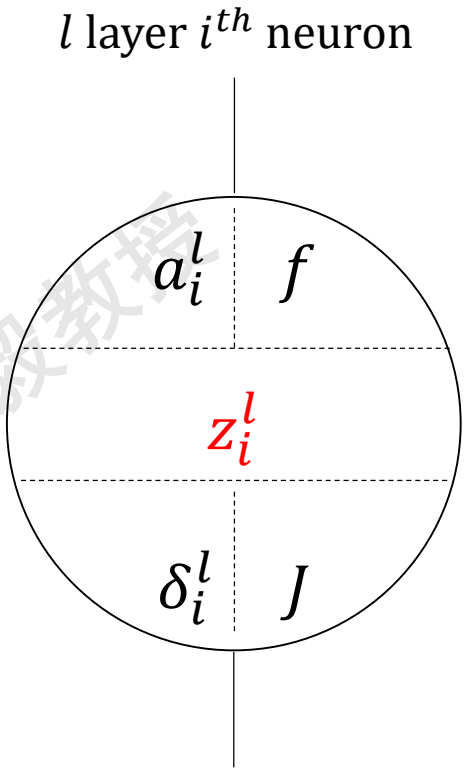
- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

Network Structure



Network Concepts

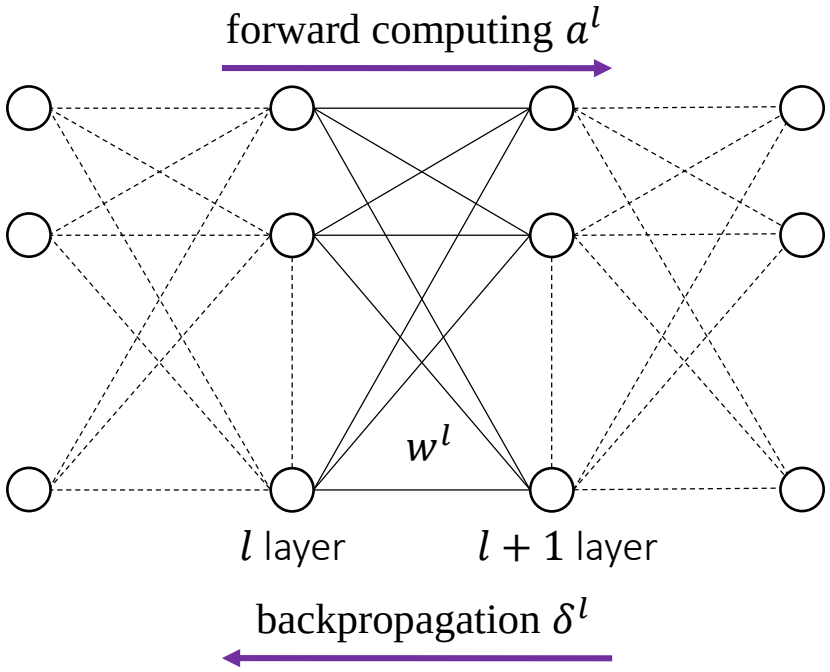


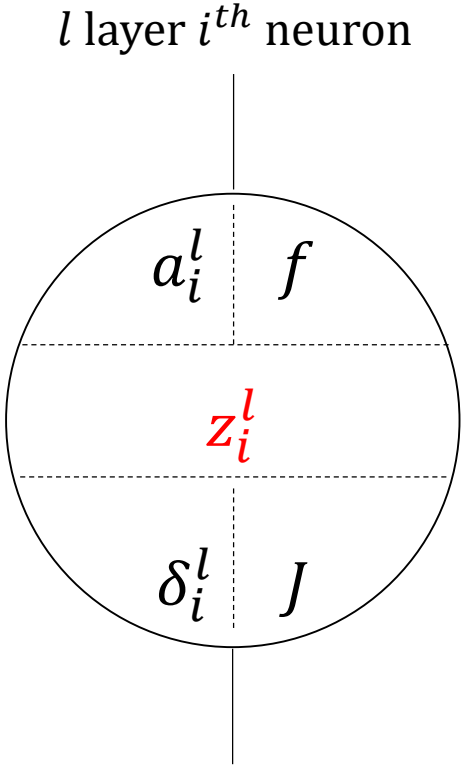


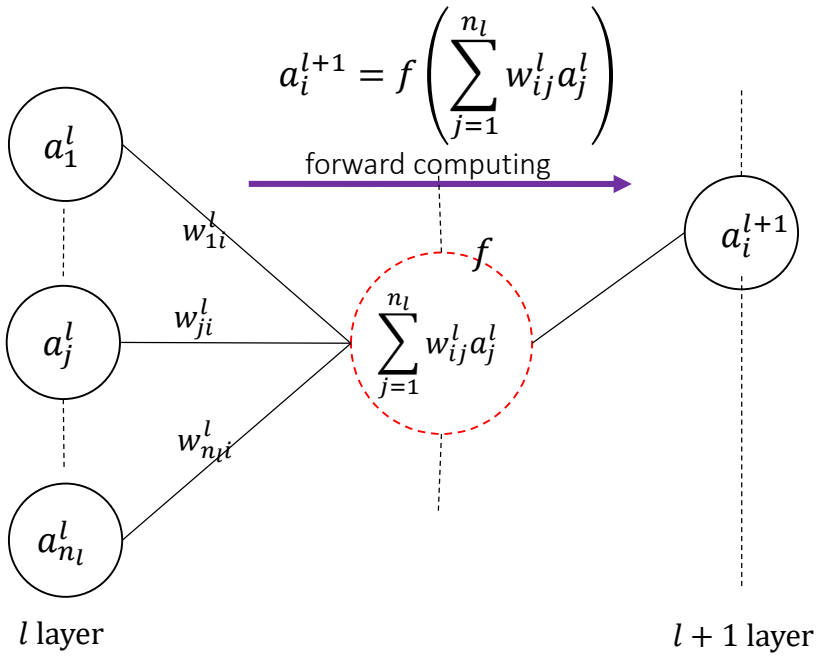
$$\frac{\partial J}{\partial z_i^l} = \delta_i^l \quad \overset{\text{Global } J}{\longleftarrow} \; z_i^l \; \overset{\text{Local } f}{\longrightarrow} \quad a_i^l = f(z_i^l)$$

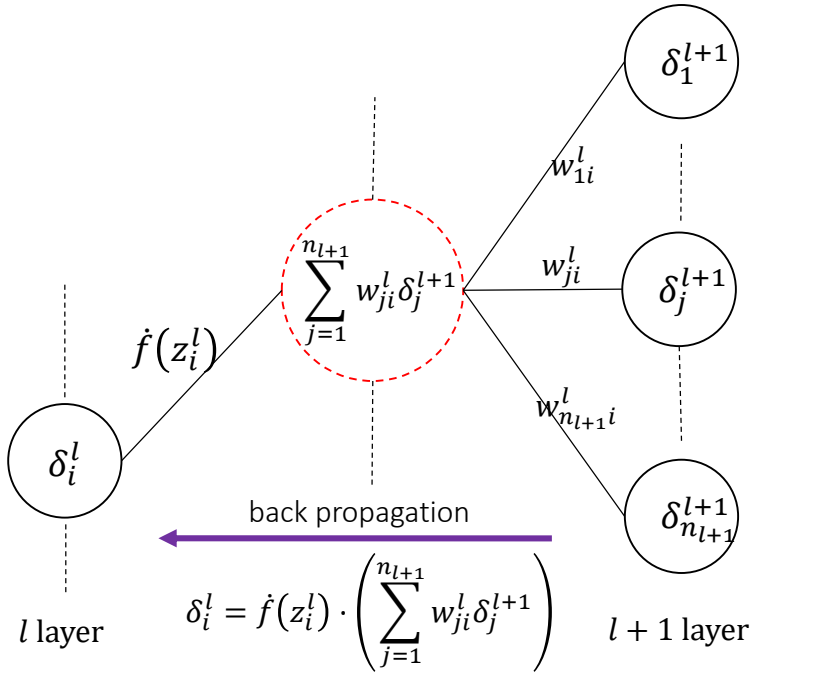
Bridge

Network Operations









Network Functions

function $fc(w^l, a^l)$

$for\ i = 1{:}\, n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

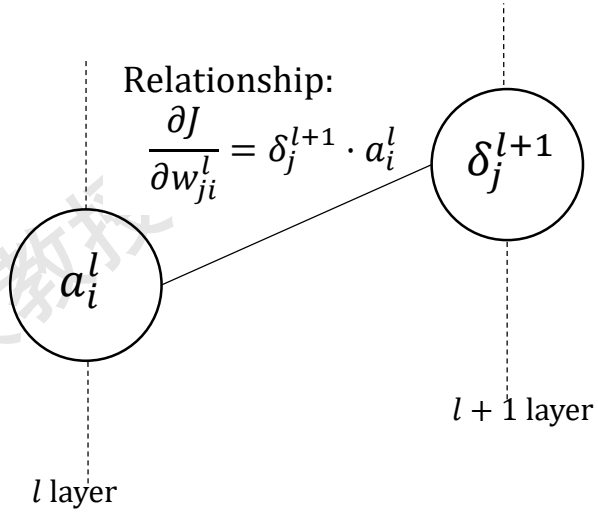
end

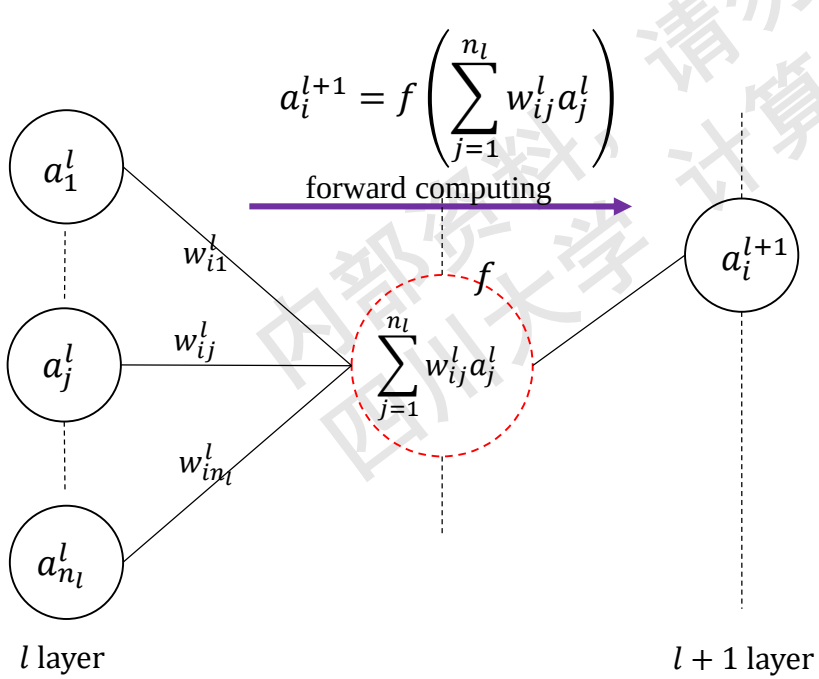
function $bc(w^l, \delta^{l+1})$

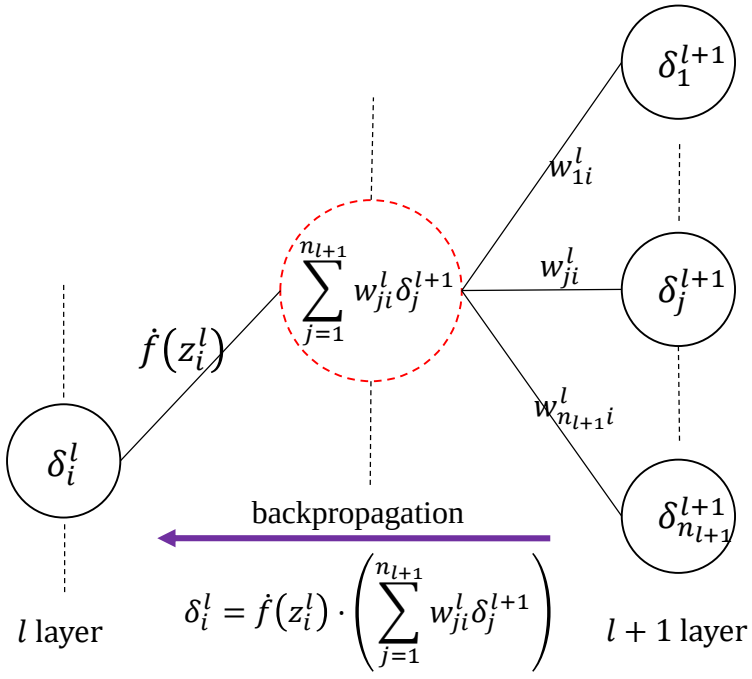
$for\ i = 1{:}\, n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

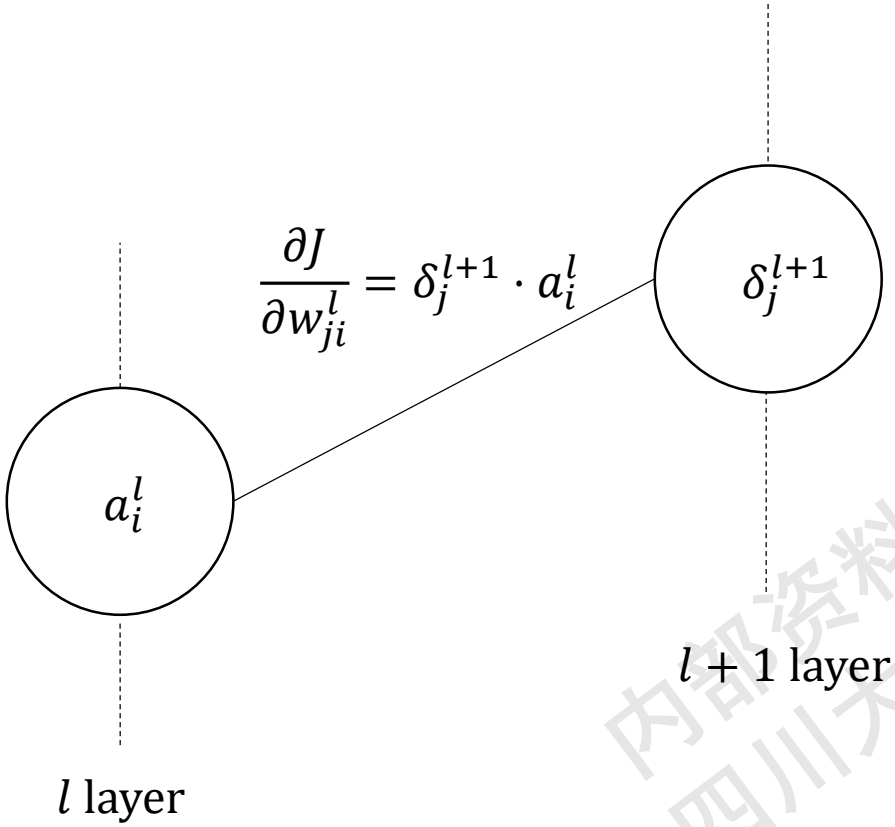
end

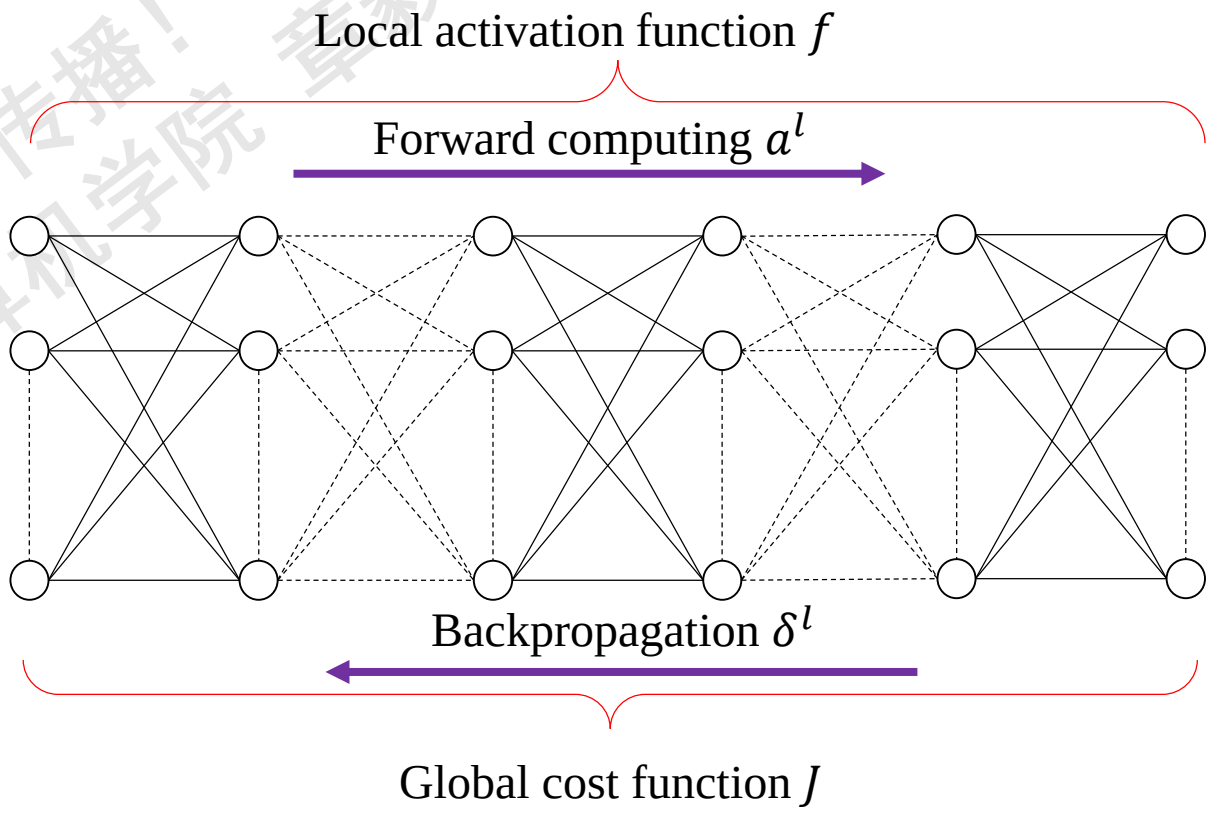






Network Relationship

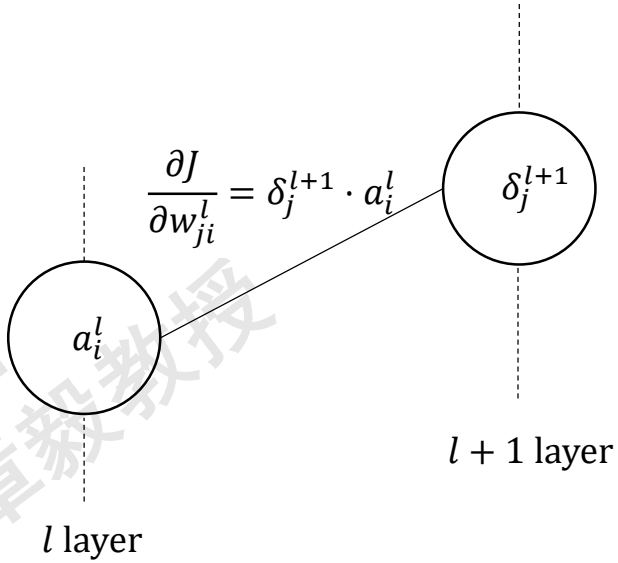




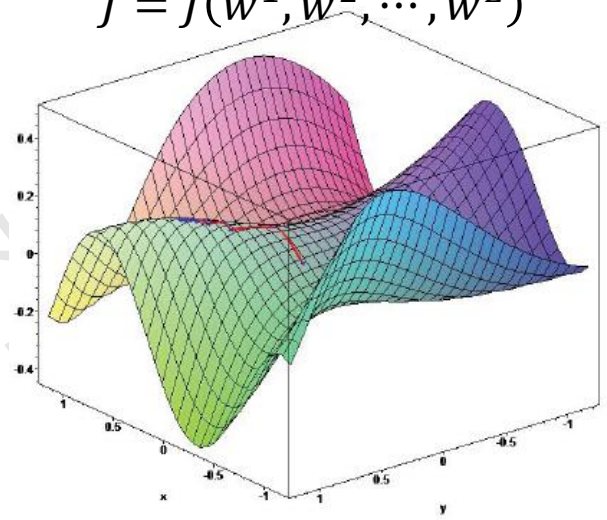
Network Updating Rule

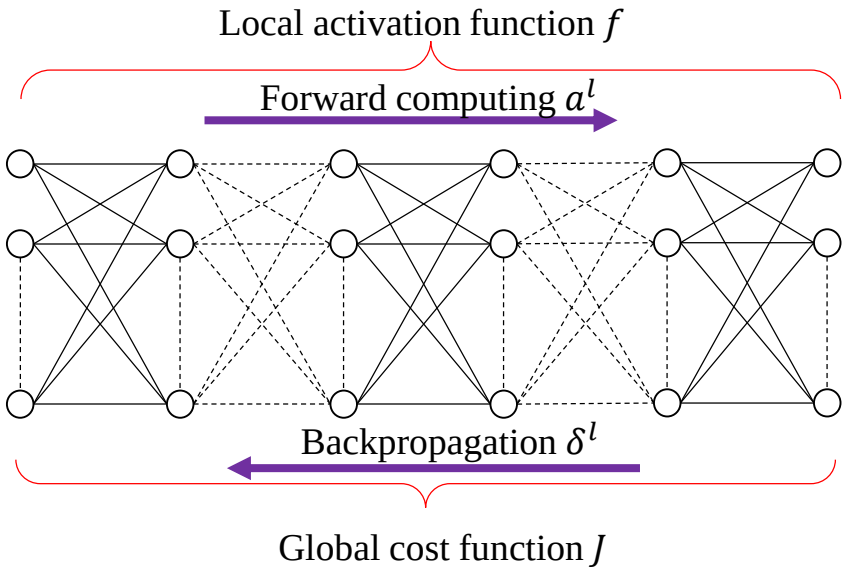
Updating rule

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}}$$



$$J = J(w^1, w^2, \cdots, w^L)$$





The BP Algorithm

Step 1. Input the training data set $D = \{(x, y)\}$

Step 2. Initial each w_{ij}^{l}, and choose a learning rate α.

Step 3. For each mini-batch sample $D_m \subseteq D$

for each $x \in D_m$

$a^1 \leftarrow x$;

for $l = 2: L$

$a^l \leftarrow fc(w^l, a^l)$;

end

$\delta^L = \frac{\partial J}{\partial z^L}$;

for $l = L - 1: 2$

$\delta^l \leftarrow bc(w^l, \delta^{l+1})$;

end

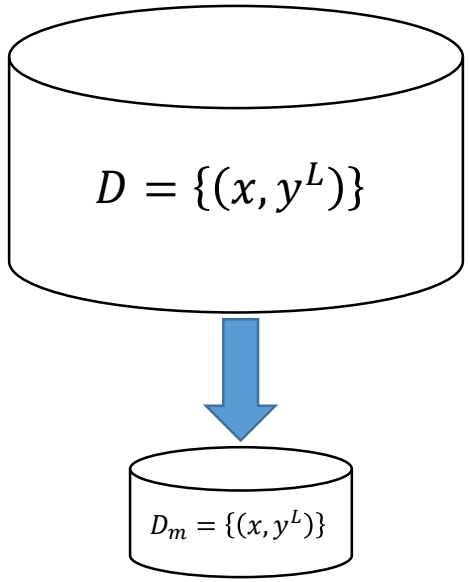
$\frac{\partial J}{\partial w_{ji}^{l}} \leftarrow \frac{\partial J}{\partial w_{ji}^{l}} + \delta_j^{l+1} \cdot a_i^l$;

end

Step 4. Updating

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}};$$

Step 5. Return to Step 3 until each w^lconverge.



function $fc(w^l, a^l)$

$for\ i = 1: n_{l+1}$

$$z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^{l} a_j^l$$

$$a_i^{l+1} = f(z_i^{l+1})$$

end

Relationship:

$$\frac{\partial J}{\partial w_{ji}^{l}} = \delta_j^{l+1} \cdot a_i^l$$

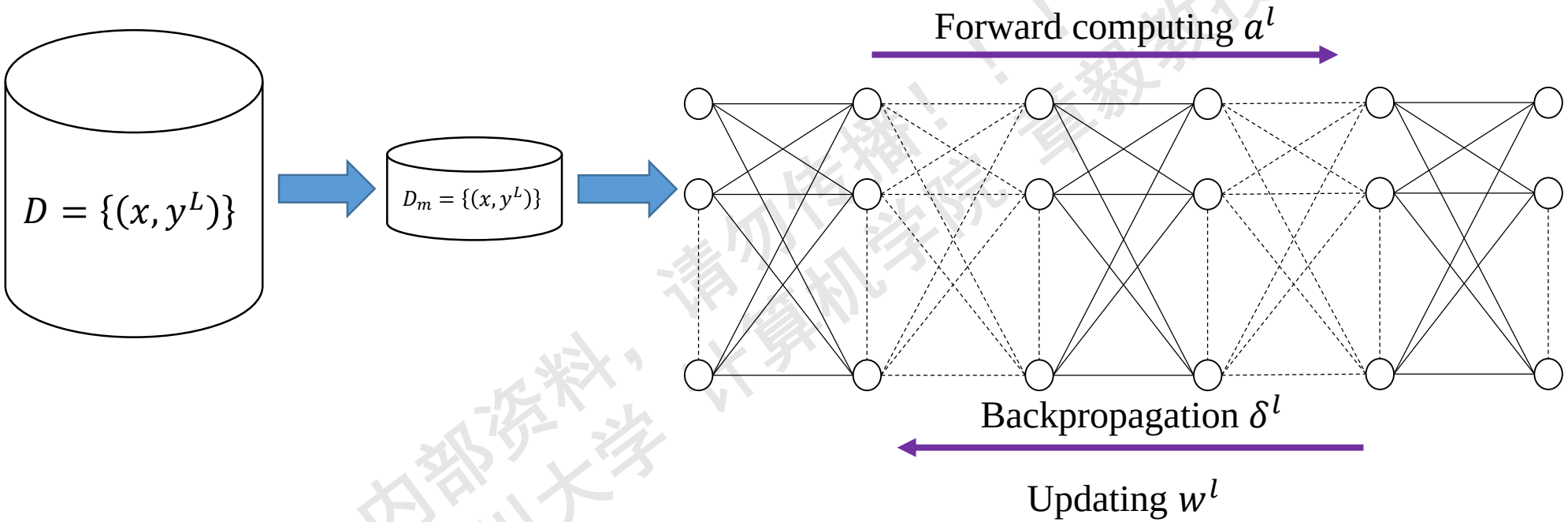
function $bc(w^l, \delta^{l+1})$

$for\ i = 1: n_l$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^{l} \delta_j^{l+1} \right)$$

end

Network Training



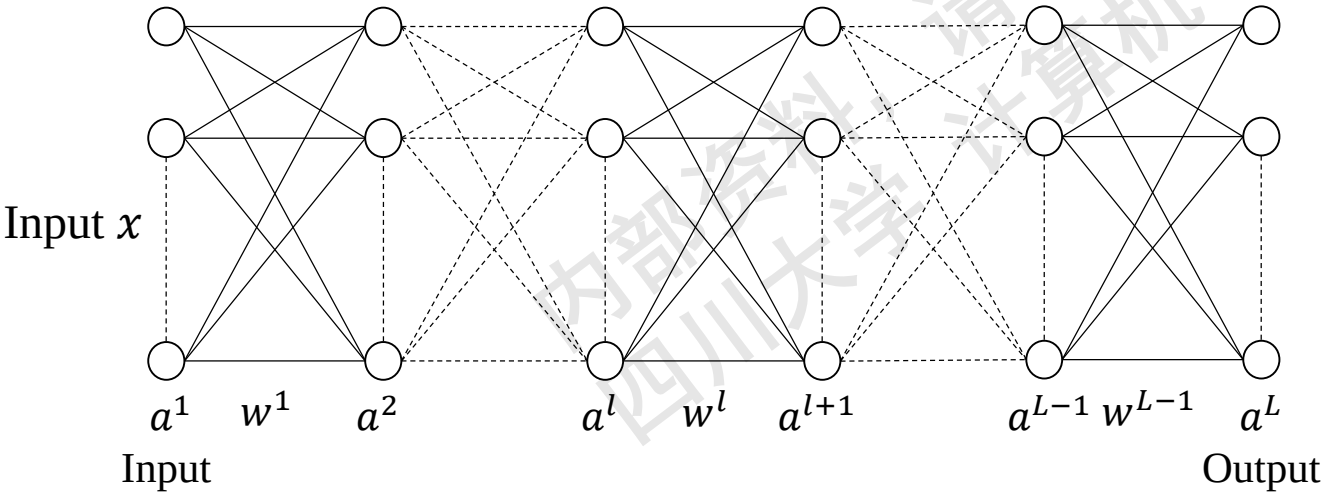
Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

On the Target Output

Problem: How to define target output?

In principle, it can be defined in any way by users. However, it must fit the meaning of applications. Thus, it is application originated. A target output must correspond to its associated input.



Defined on the last layer
Target Output

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix} \Longleftrightarrow \text{Input } x$$

A training sample (x, y^L)

On the Target Output

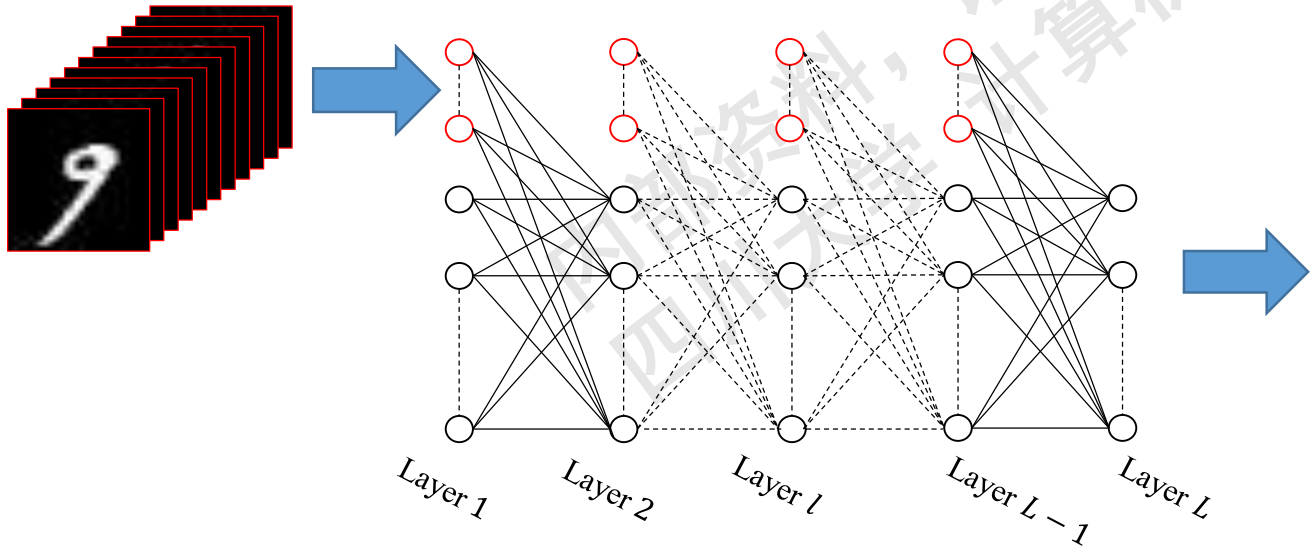
Classification Problem

The target is to assign each input data sample to its class label. Thus, the target output can be defined by the representation of the label.

Representation of the label associated with the input.

Target Output

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$



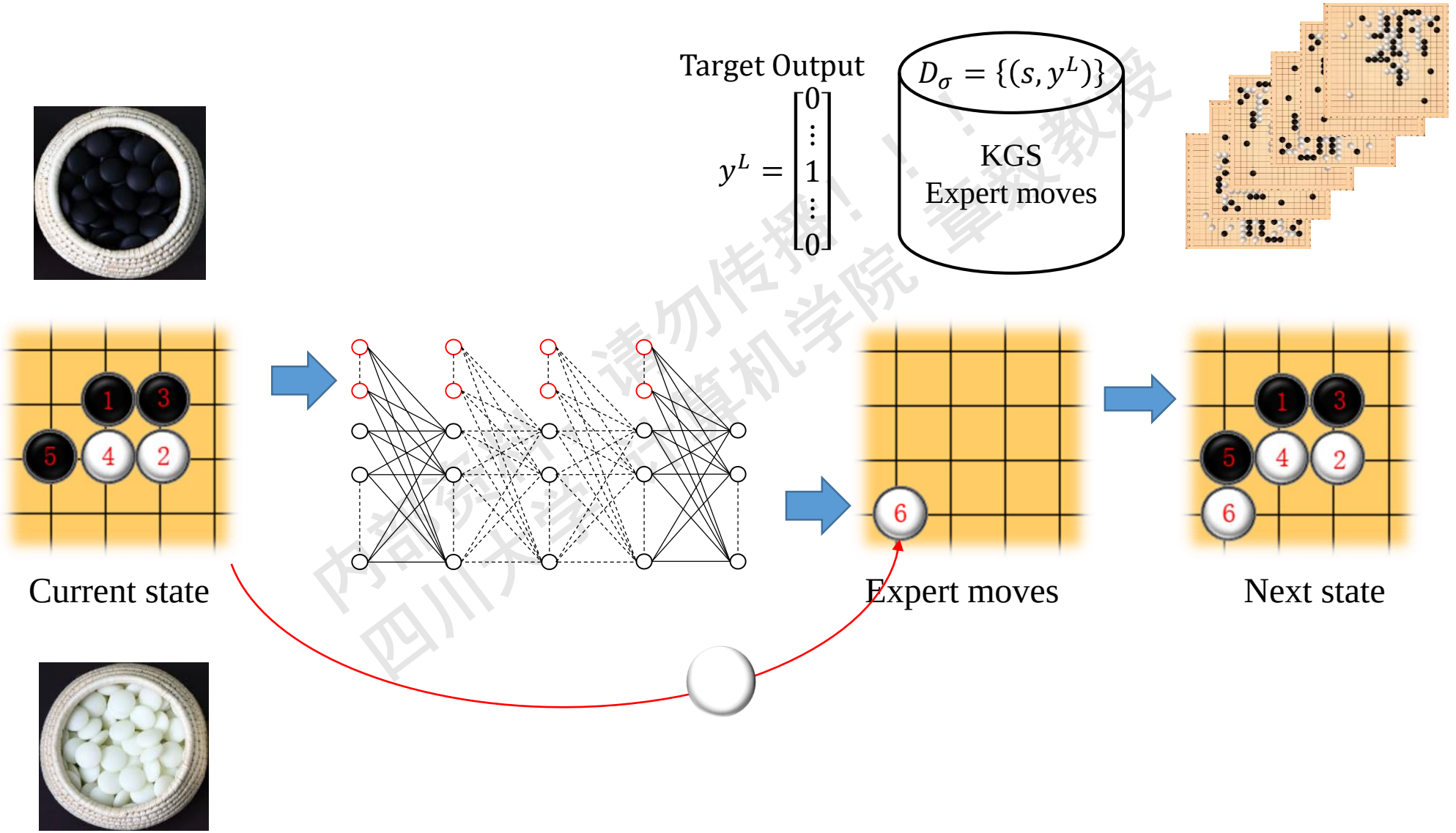
Tip:
The number of output neurons equals to the number of classes.

Classes Label

0	1	2	3	4	5	6	7	8	9
[0 0 0 0 0 0 0 0 0 1]	[1 0 0 0 0 0 0 0 0 0]	[0 1 0 0 0 0 0 0 0 0]	[0 0 1 0 0 0 0 0 0 0]	[0 0 0 1 0 0 0 0 0 0]	[0 0 0 0 1 0 0 0 0 0]	[0 0 0 0 0 1 0 0 0 0]	[0 0 0 0 0 0 1 0 0 0]	[0 0 0 0 0 0 0 1 0 0]	[0 0 0 0 0 0 0 0 1 0]

Representation

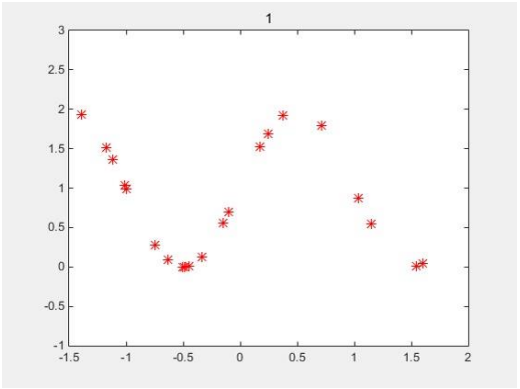
On the Target Output



On the Target Output

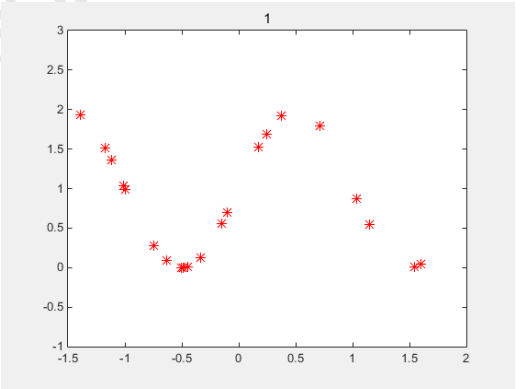
Curve Fitting Problem

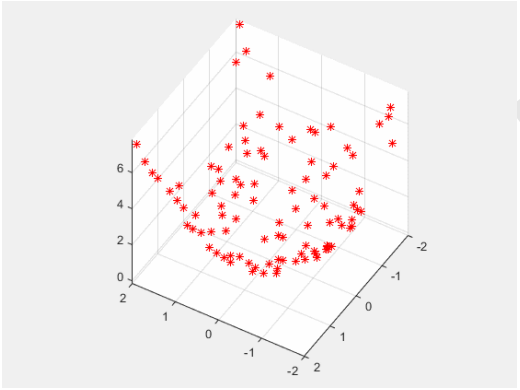
Given a set of sample data, estimates a curve that go through the samples.



Sample data

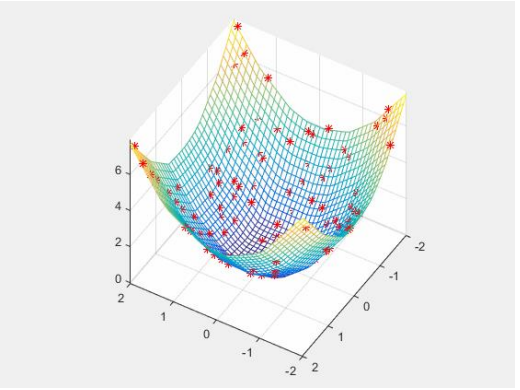
	1	2	3	4	5	6
x	-0.5000	0.1740	0.7100	-0.9980	-0.6340	1.0400
y	0	1.5198	1.7902	0.9937	0.0873	0.8747



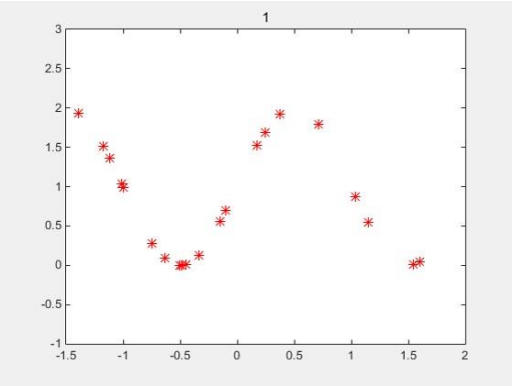


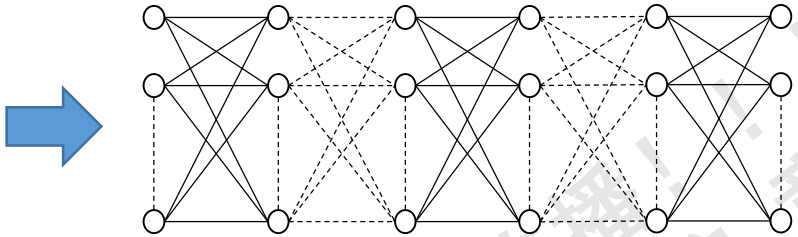
Sample data

	1	2	3	4	5	6
x	-0.2000	-1.9000	1.9000	0.4000	-1.9000	0.8000
	1.4000	-1.9000	-1.5000	-0.5000	0.3000	-0.1000
y	2.0000	7.2200	5.8600	0.4100	3.7000	0.6500

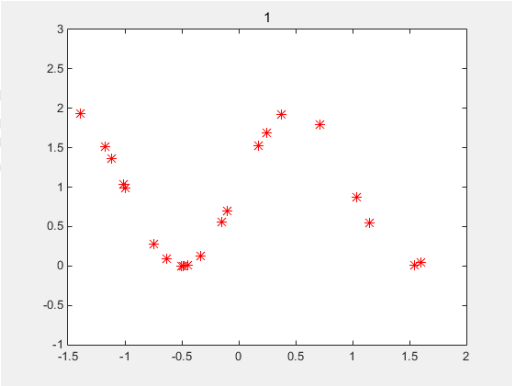


On the Target Output

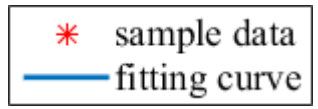


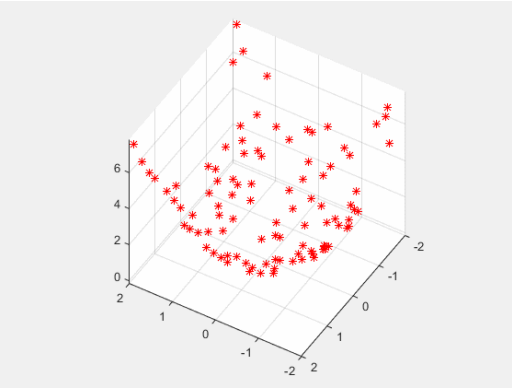


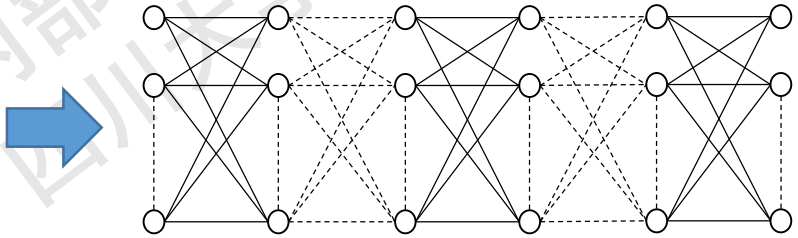
Training sample(x, y)



The target output is the value of y corresponding to x of each sample.

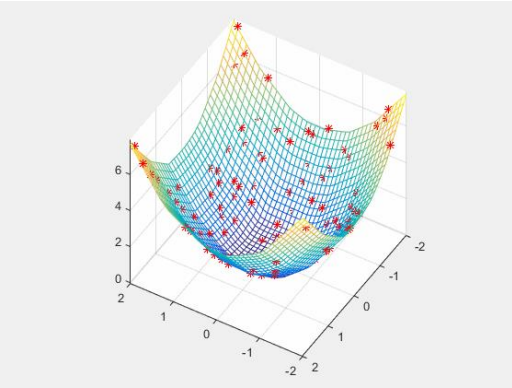






Target Output

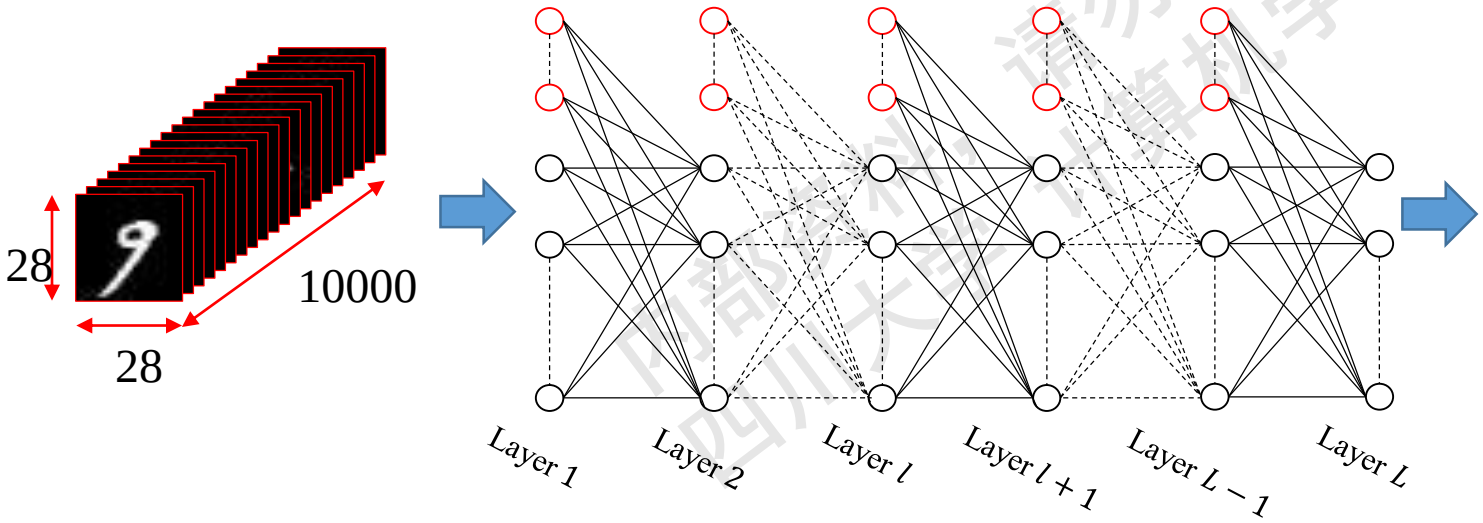
$y^L = y$

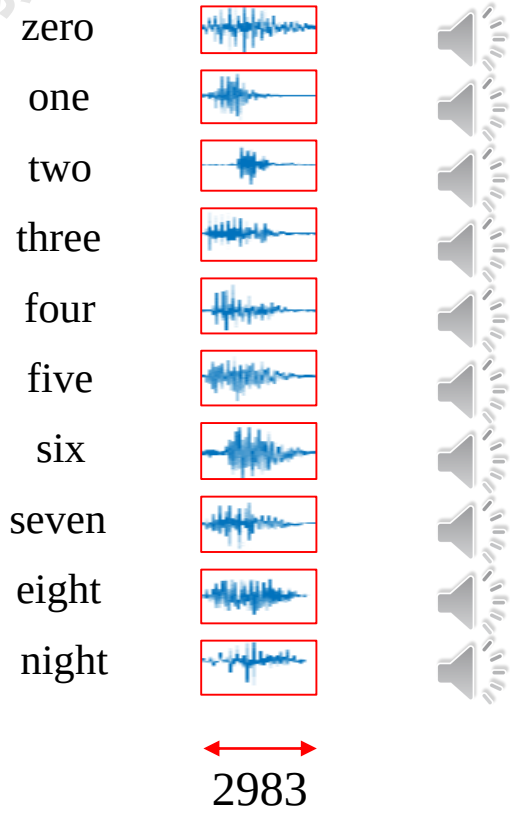


On the Target Output

Target Output

$$y^L = \begin{bmatrix} y_1^L \\ y_2^L \\ \vdots \\ y_{2983}^L \end{bmatrix}$$

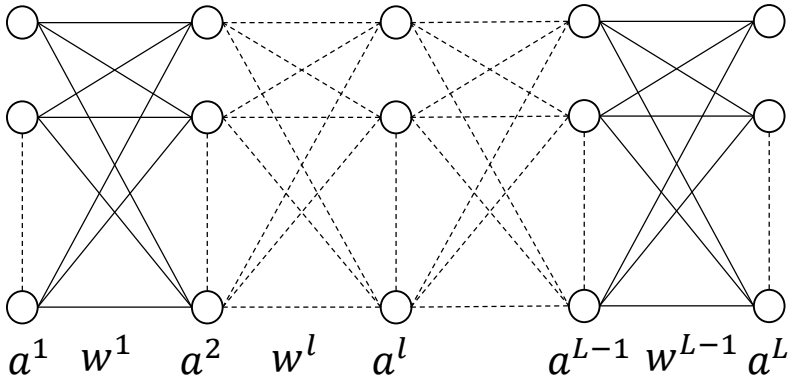




Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

On the Network Output



Network Output

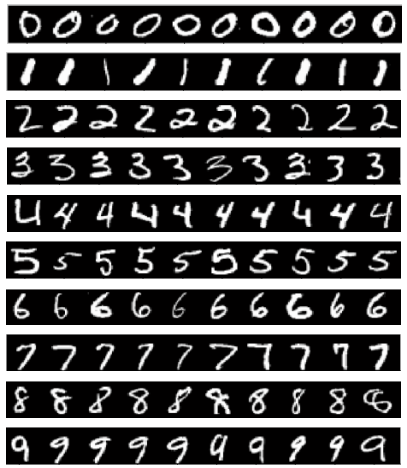
$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

Define the last layer activation function f so that the network output a^L can match the target output y^L. Note that f should be differentiable.

$$a_i^L = f(z_i^L)$$

Target Output

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

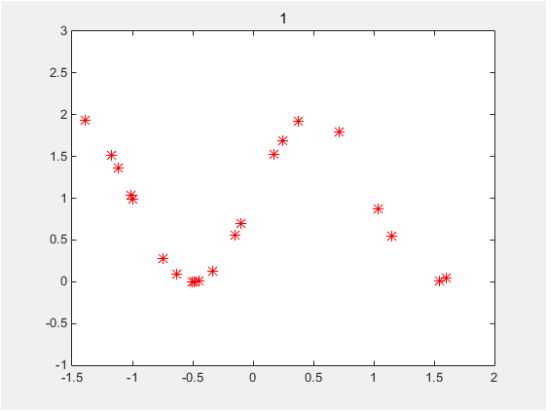


Sigmoid function

$$f(s) = \frac{1}{1+e^{-s}} \in (0,1)$$

$$\begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix} \quad \text{Threshold } \theta \quad \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

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



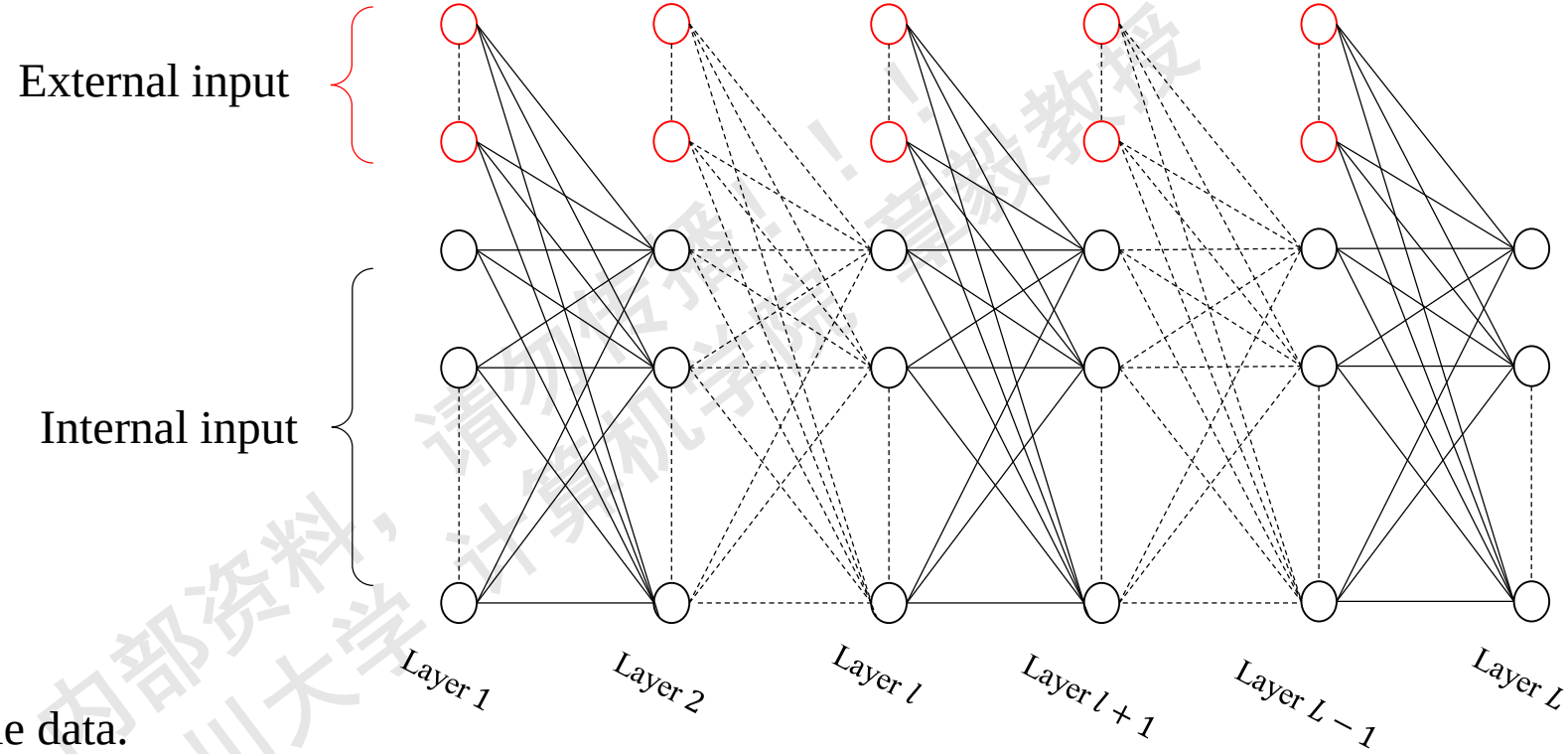
Linear function

$$f(s) = s$$

Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

On the Network Input



External input:

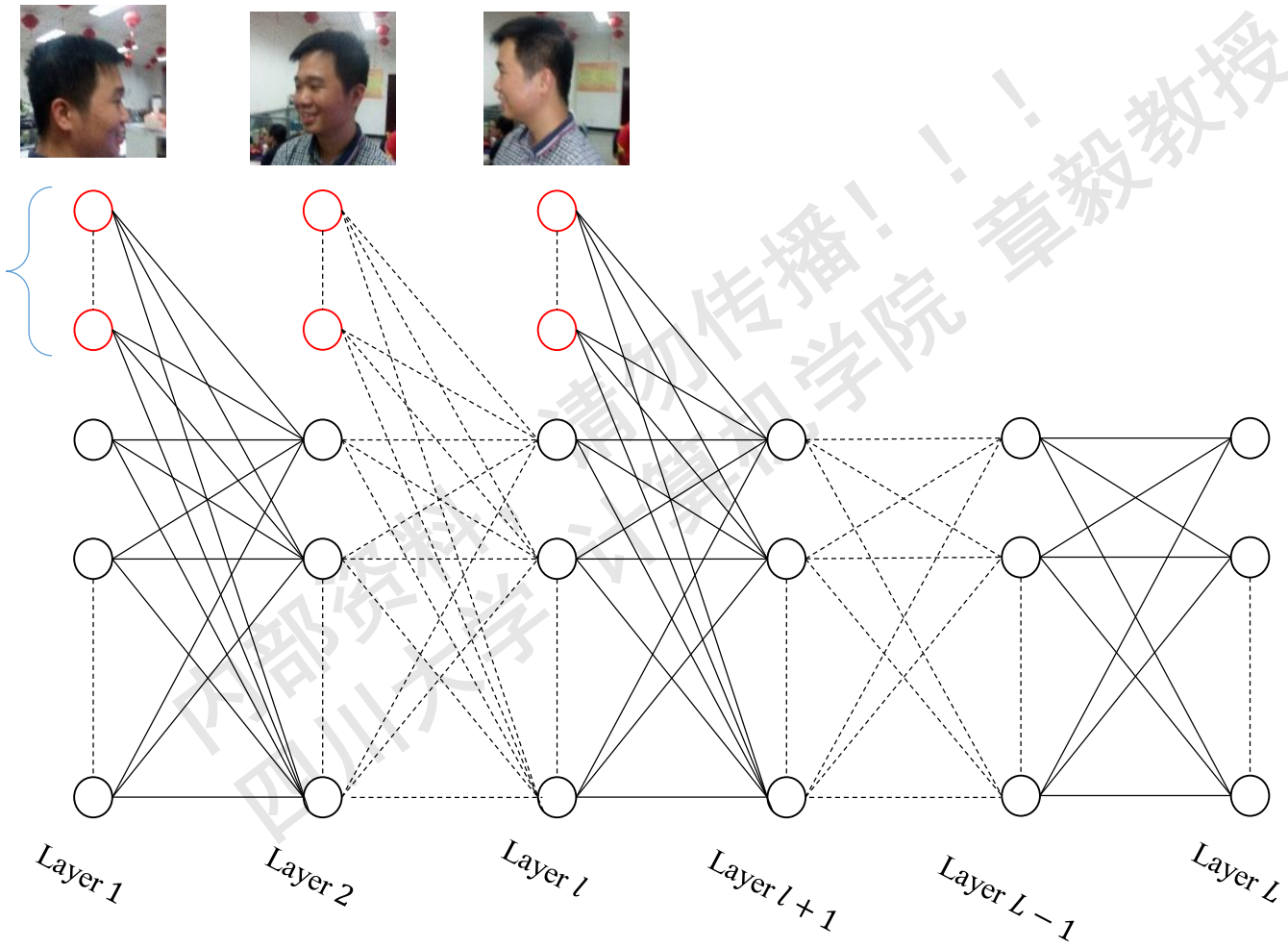
- Directly from sample data.

Internal input:

- Generated by former layer
- Maintain a working memory for the neural network
- The first layer internal input is generated by user

On the Input

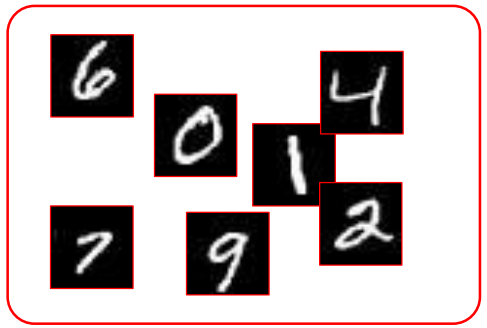
Sequence Input

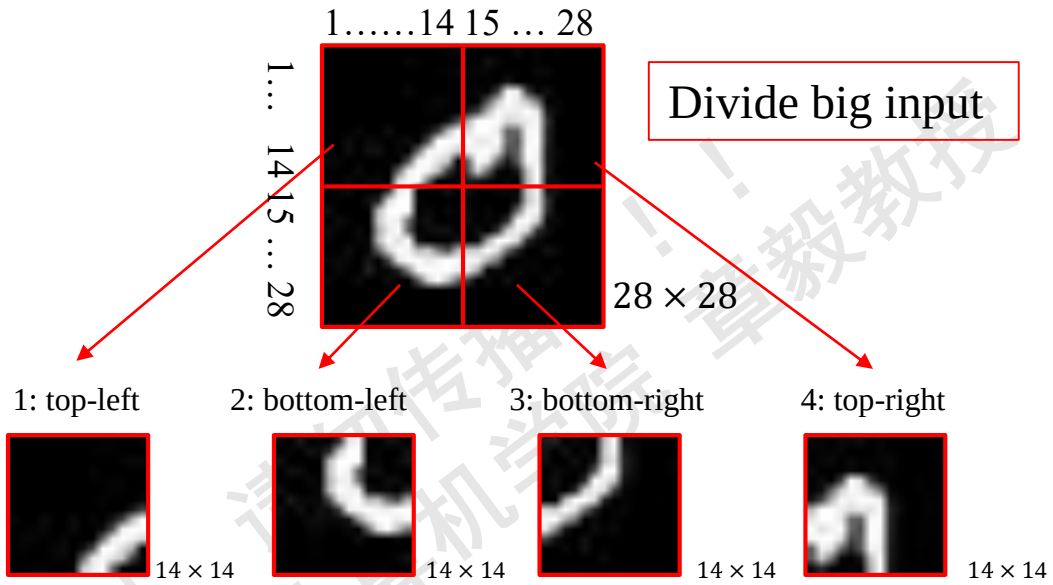


Recognize the identity

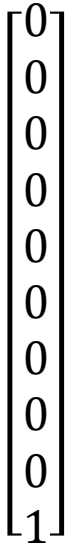
On the Input

If the dimension of the input data is too large, it can be divided into small ones.



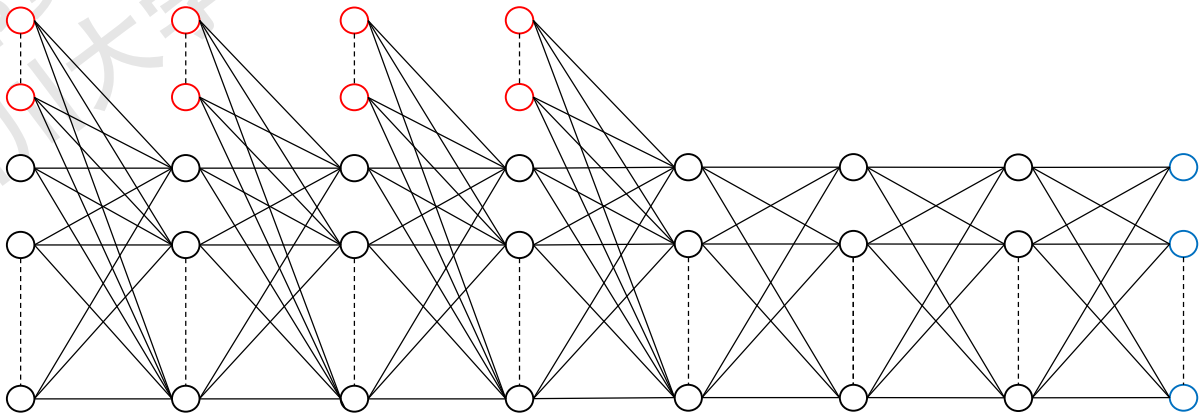


Representation

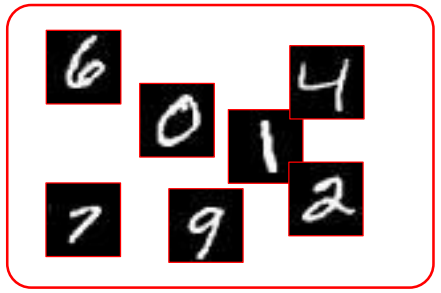


0

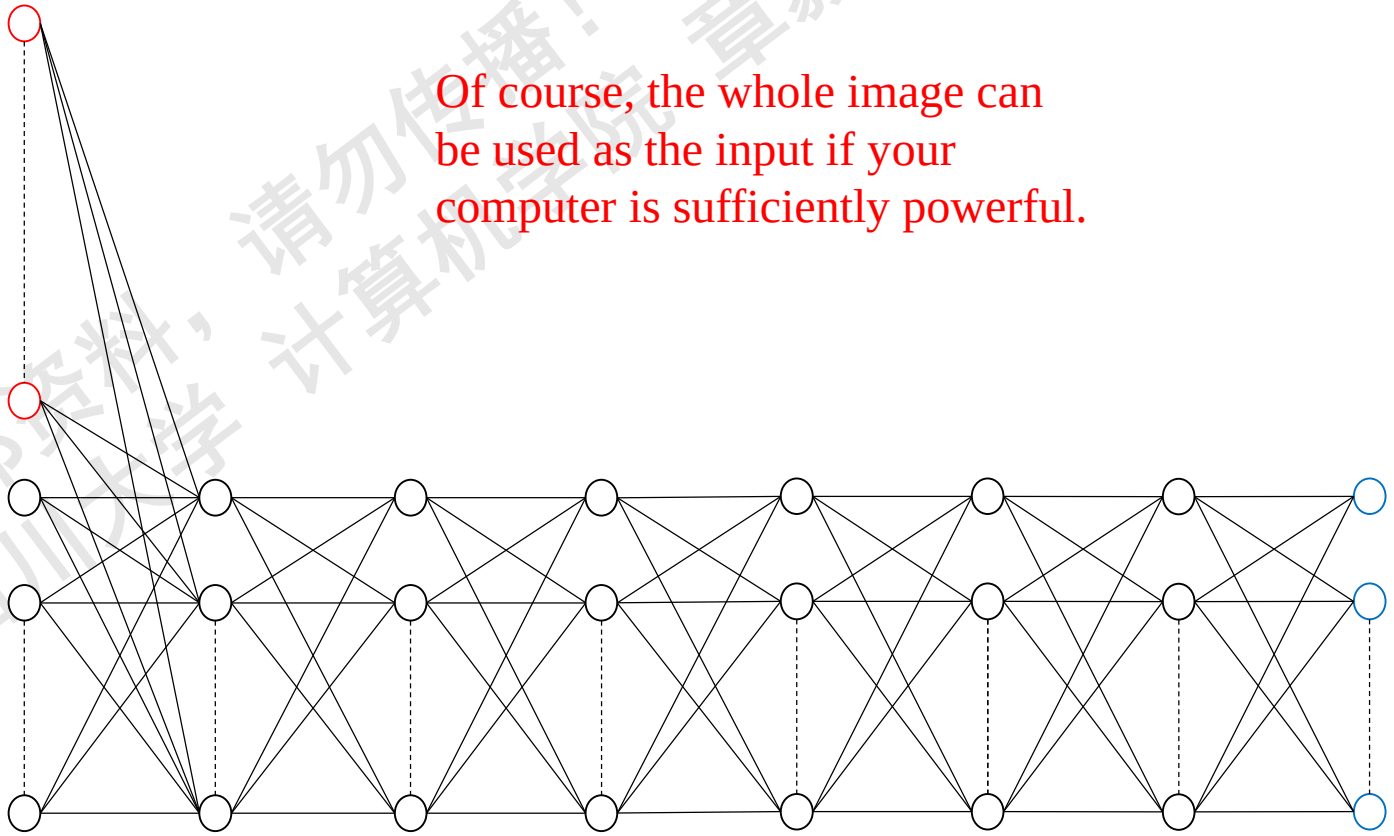
196-dimension



On the Input

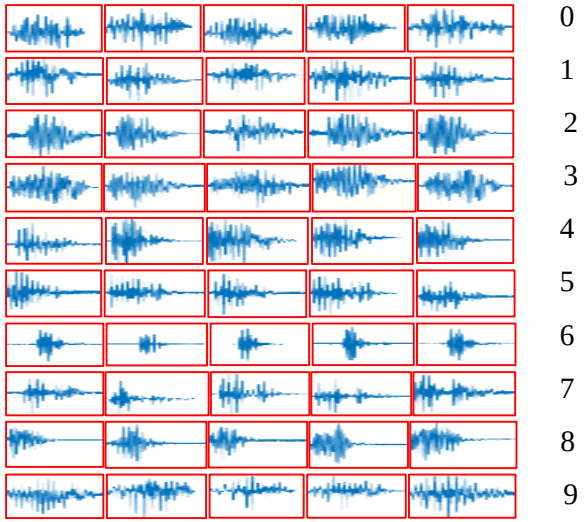


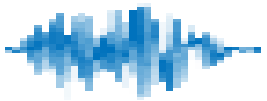
784-dimension



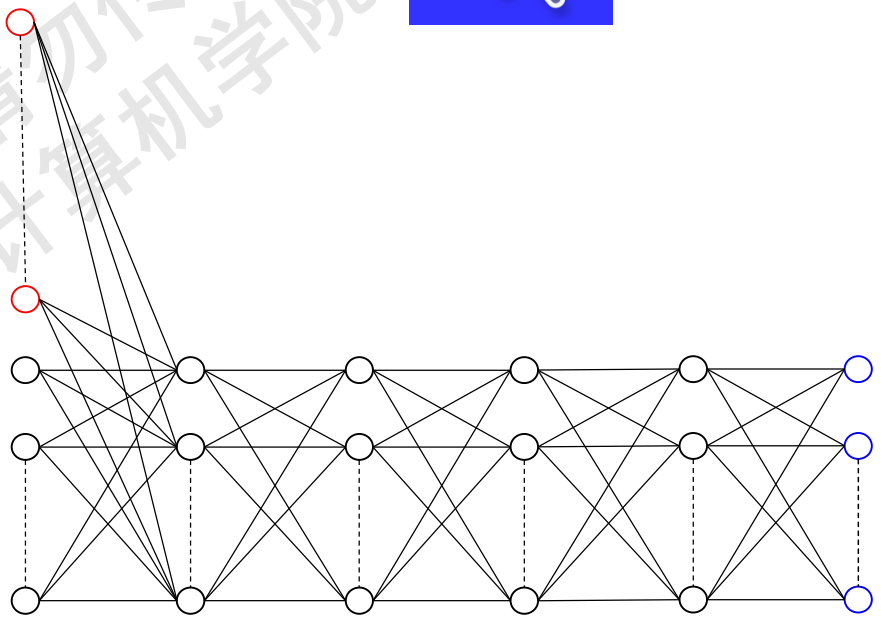
Of course, the whole image can be used as the input if your computer is sufficiently powerful.

On the input





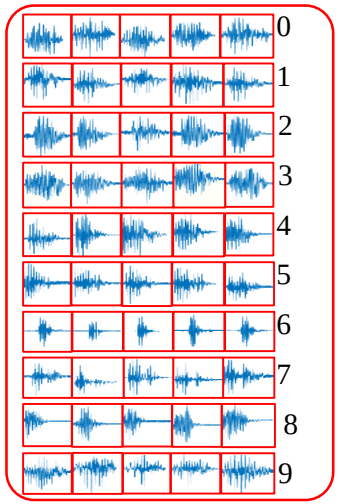
4000-dimension

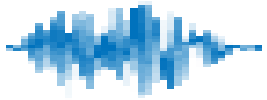


3

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

On the input

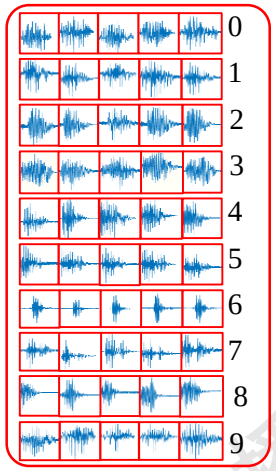




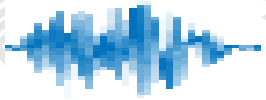
4000-dimension

784-dimension

On the Input

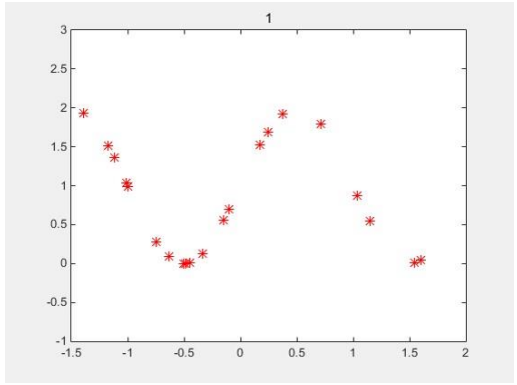


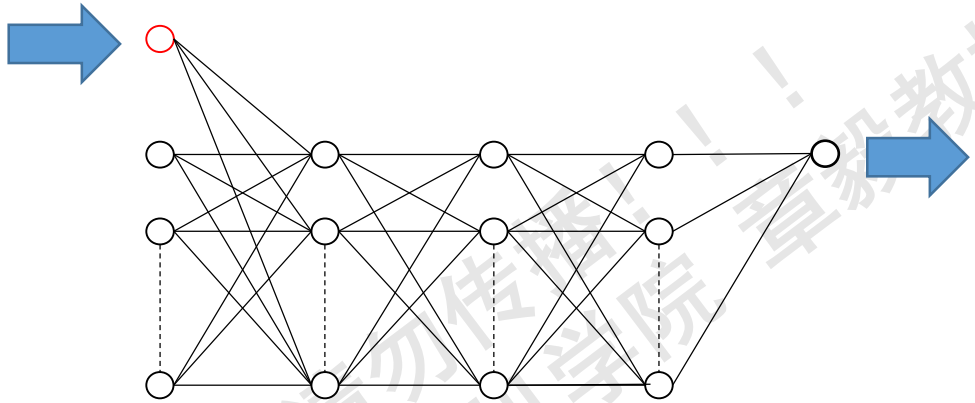
784-dimension

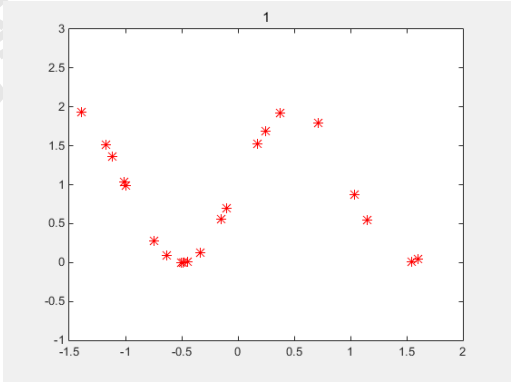


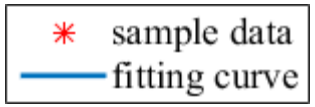
4000-dimension

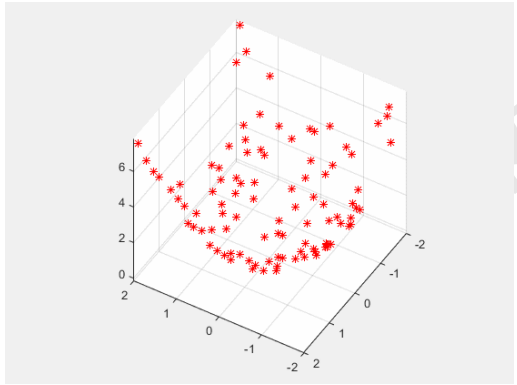
On the input

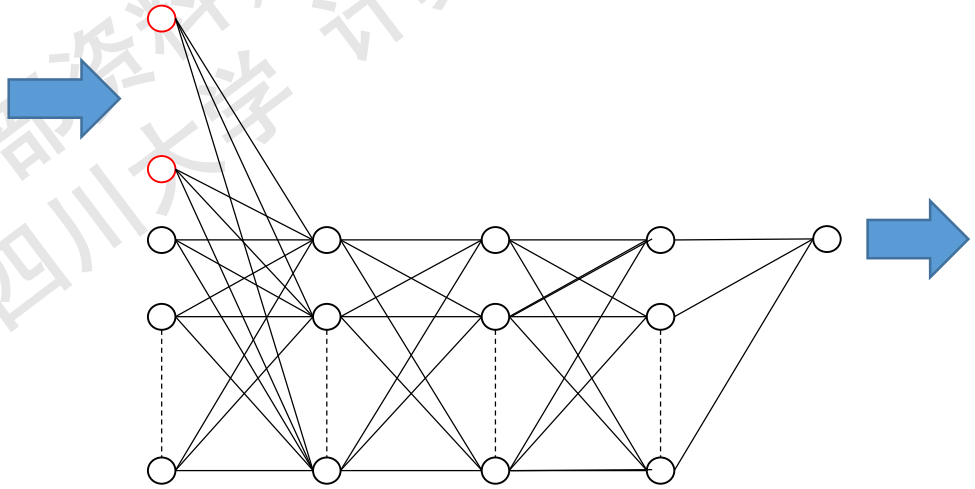


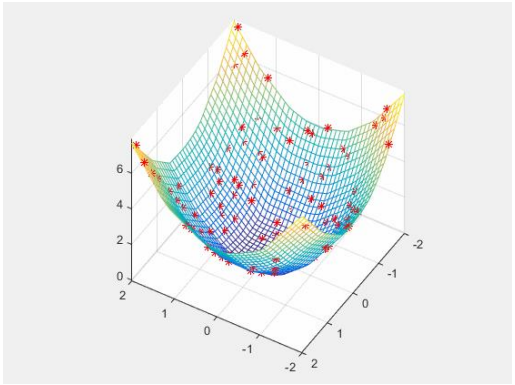








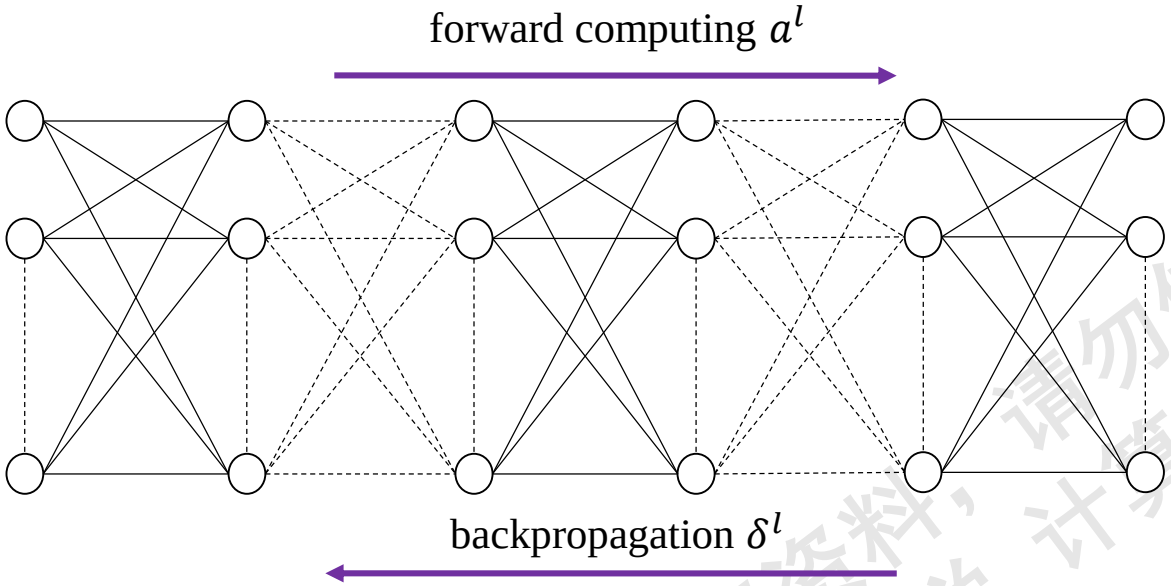




Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

On the Cost Function



Network Output

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

Target Output

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

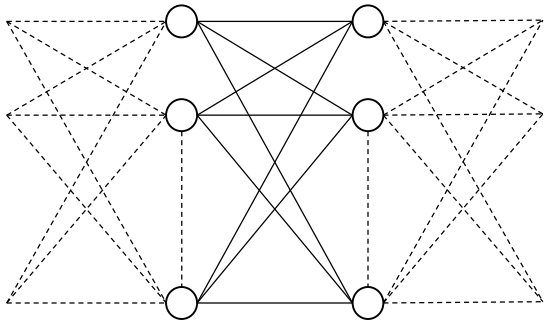
$J(a^L, y^L)$

Cost function $J(a^L, y^L)$ is used to describe the closeness between a^L and y^L, $J(a^L, y)$ is indeed a function of $(w^1, \cdots, w^{L-1})$, i. e. ,

$$J = J(w^1, \cdots, w^{L-1}).$$

On the Cost Function

forward computing a^l



backpropagation δ^l

$$0 \le y_i^L \le 1 \ (i = 1, \cdots, n_L)$$

$$a_i^L = f(z_i^L) = \frac{1}{1 + e^{-z_i^L}}$$

Sigmoid function

Square Error

$$\begin{cases} J = \frac{1}{2}\sum_{j=1}^{n^L}(a_j^L - y_j^L)^2 \\ \delta_i^L = \frac{\partial J}{\partial z_i^l} = (a_i^L - y_i^L) \cdot \dot{f}(z_i^L) \end{cases}$$

Network Output

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

Target Output

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

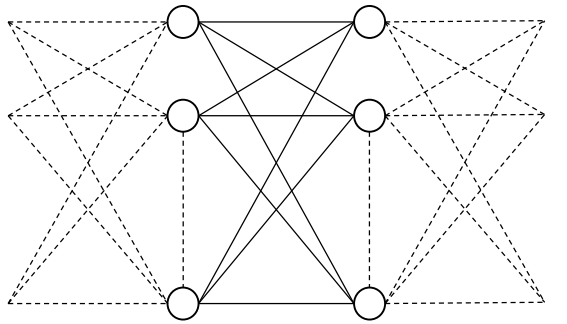
$J(a^L, y^L)$

Cost function $J(a^L, y^L)$ is used to describe the closeness between a^L and y^L, $J(a^L, y)$ is indeed a function of $(w^1, \cdots, w^{L-1})$, i. e.,

$$J = J(w^1, \cdots, w^{L-1}).$$

On the Cost Function

forward computing a^l



backpropagation δ^l

$$\sum_{j=1}^{n_L} y_i^L = 1$$

$$a_j^L = \frac{e^{z_j^L}}{e^{z_1^L} + \cdots + e^{z_{n_L}^L}}$$

Softmax function

Network Output

$$a^L = \begin{bmatrix} a_1^L \\ \vdots \\ a_{n_L}^L \end{bmatrix}$$

Target Output

$$y^L = \begin{bmatrix} y_1^L \\ \vdots \\ y_{n_L}^L \end{bmatrix}$$

$J(a^L, y^L)$

Cost function $J(a^L, y^L)$ is used to describe the closeness between a^L and y^L, $J(a^L, y)$ is indeed a function of $(w^1, \cdots, w^{L-1})$, i. e.,

$$J = J(w^1, \cdots, w^{L-1}).$$

Cross Entropy

$$J = -\sum_{j=1}^{n_L} y_j^L \cdot \log(a_j^L) + \lambda \cdot \sum (w_{ij}^l)^2$$

$$a_j^L = \frac{e^{z_j^L}}{\sum_{i=1}^{n_L} e^{z_i^L}}$$

$$\delta_i^L = a_i^L - y_i^L$$

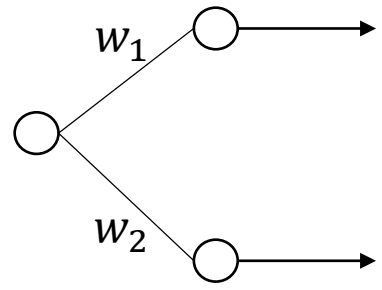
On the Cost Function

An example

Sample data

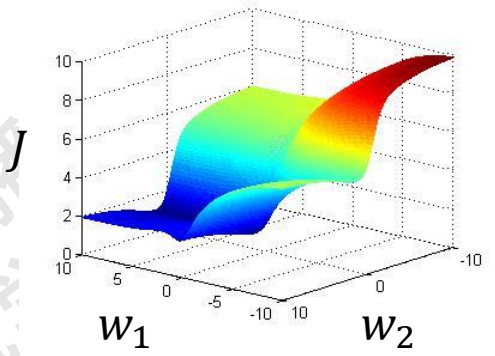
	1	2
x	0.8000	0.2000
y	0	1

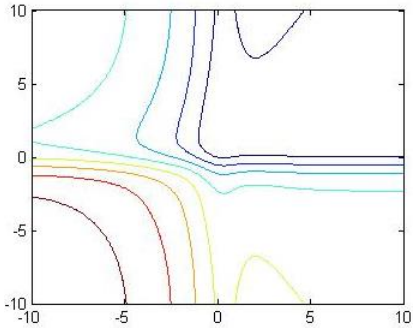
Network



Square Error

$$\begin{cases} J = \frac{1}{2}\sum_{j=1}^{2}(a_j - y_j)^2 \\ a_j = \frac{1}{1+\exp(-z_j)} \\ z_j = w_j \cdot x \end{cases}$$

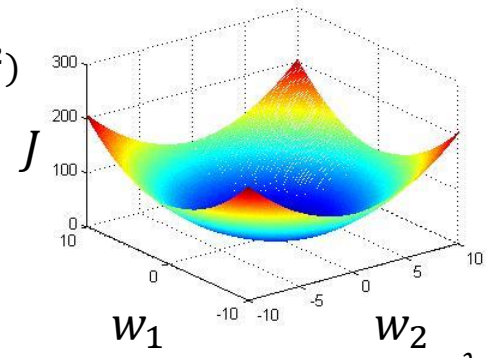


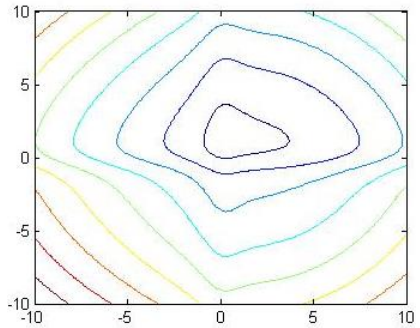


contour

Cross Entropy

$$\begin{cases} J = -\sum_{j=1}^{2} y_j \cdot \log(a_j) + \lambda(w_1^2 + w_2^2) \\ a_j = \frac{e^{z_j}}{\sum_{i=1}^{2} e^{z_j}} \\ z_j = w_j \cdot x \end{cases}$$





$\lambda = 0.05$

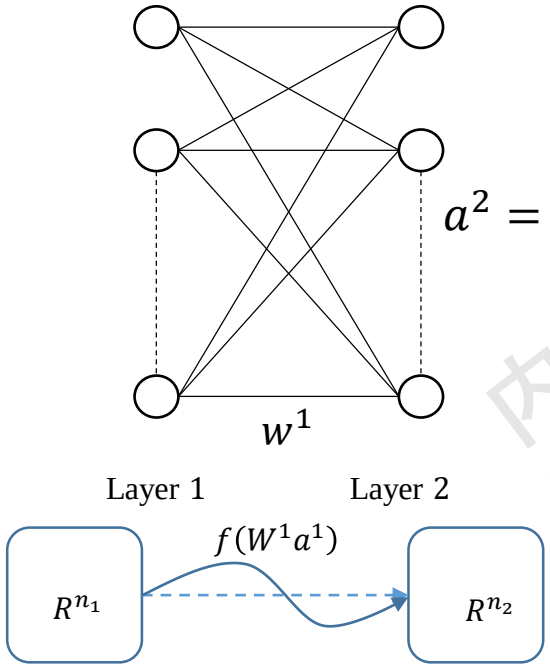
Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

On the Depth of the Networks

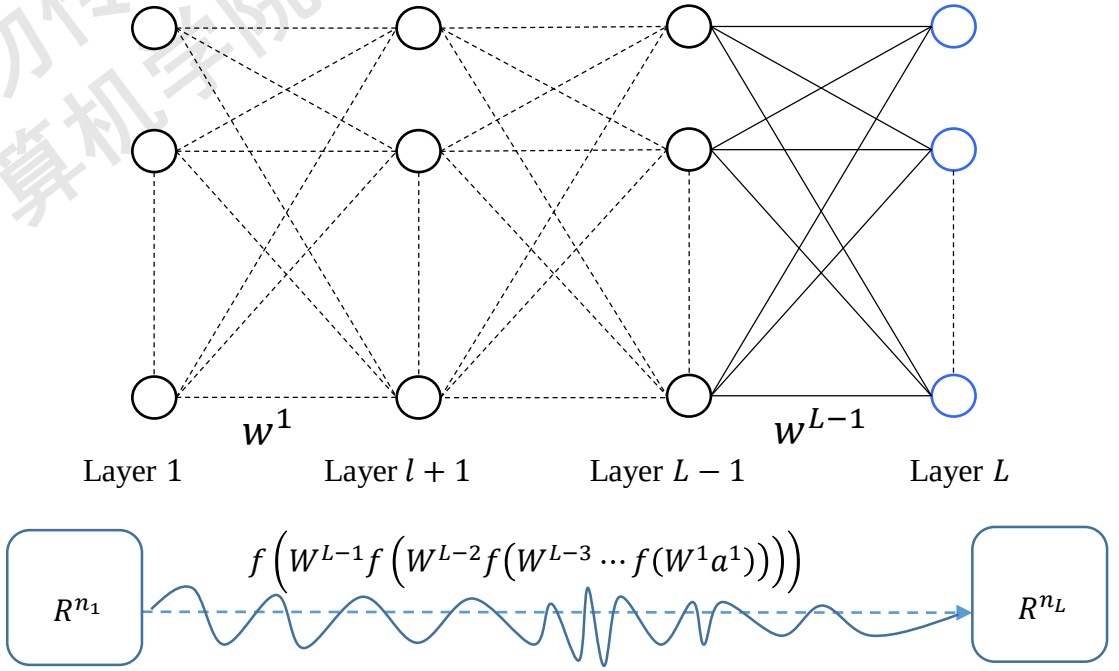
Shallow neural network

- $L = 2$
- too shallow to learn complex mappings

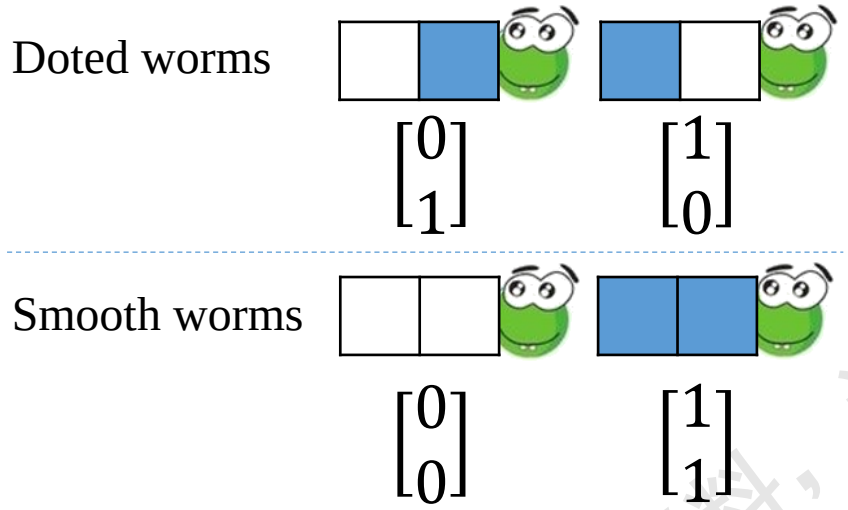


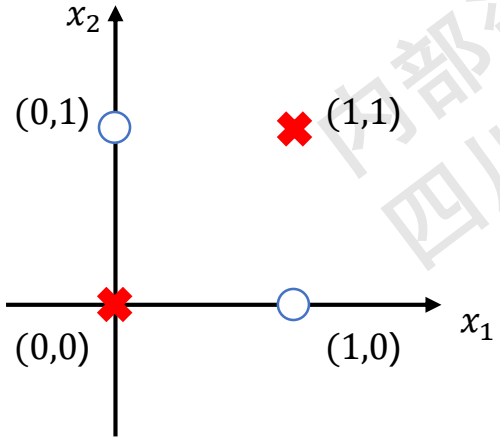
Deep neural network

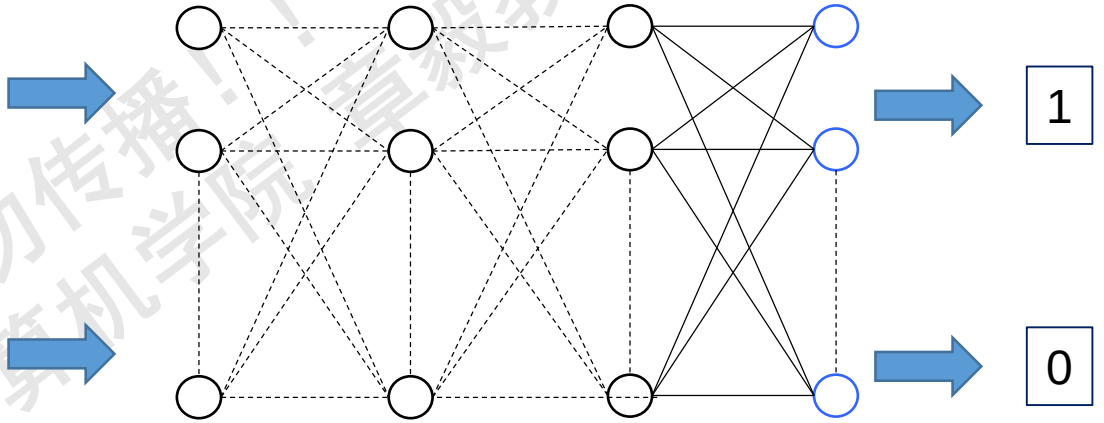
- $L > 2$
- can approximate any nonlinear mappings in any precise provided sufficient neurons in the networks



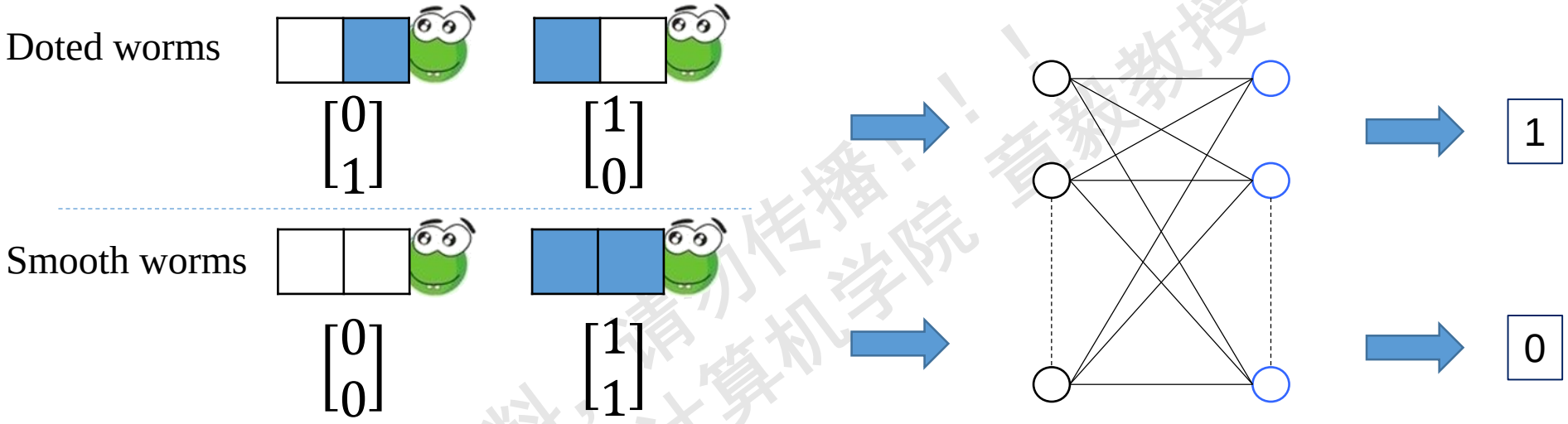
An example: XOR problem

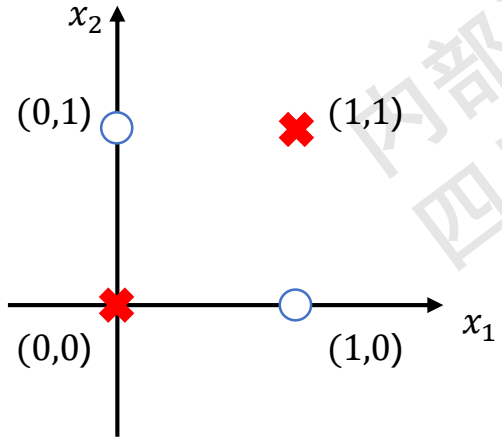






An example: XOR problem



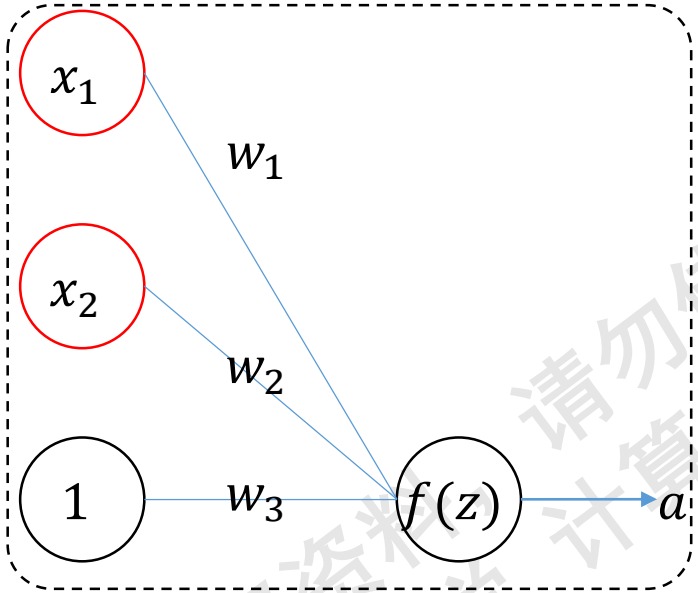


The classification task CANNOT be completed by using two layers network.

An example: XOR problem

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

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

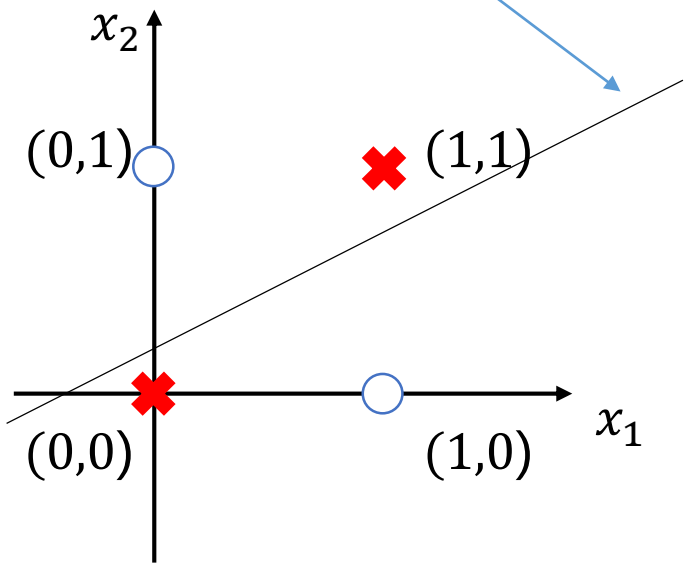


Equivalent

$a = f(w_1x_1 + w_2x_2 + w_3)$

Decision line

$w_1x_1 + w_2x_2 + w_3 = 0$

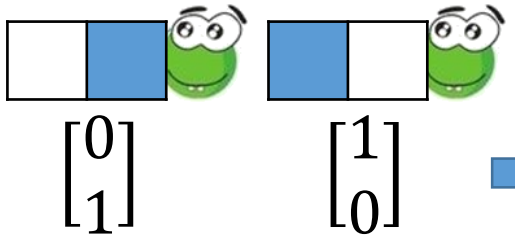


CANNOT use a line to separate the two classes!

An example: XOR problem

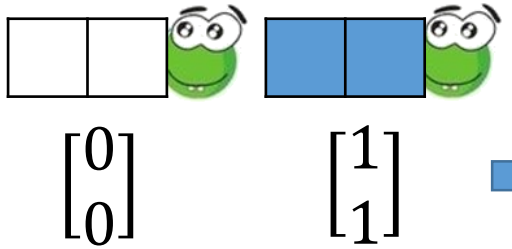
At least three layers are required for XOR problem.

Doted worms

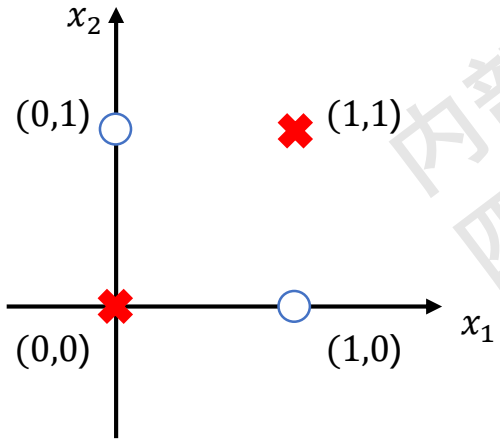


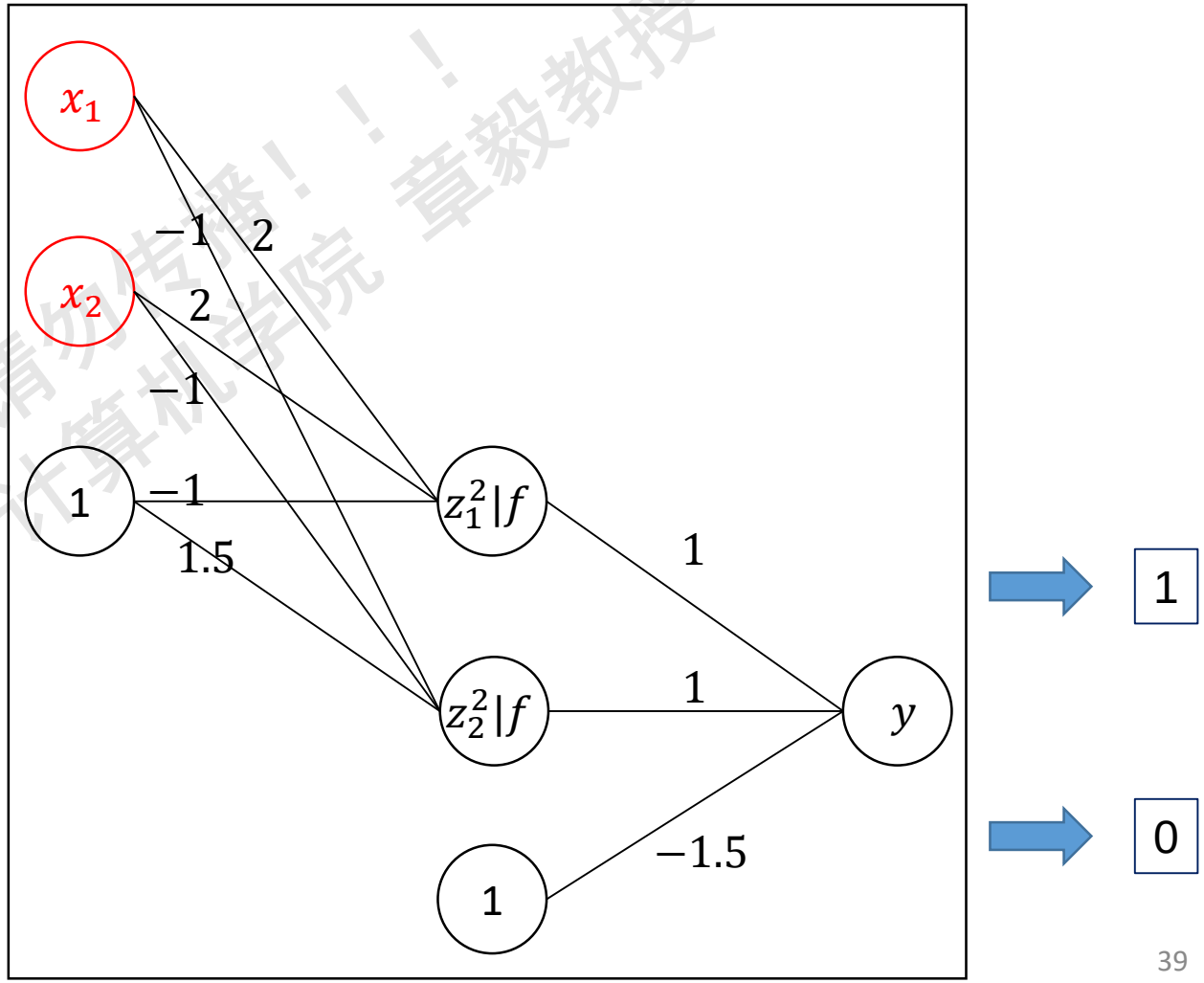
$\begin{bmatrix}0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\end{bmatrix}$

Smooth worms



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





On the Depth of the Networks

Gradient Vanishing Problem

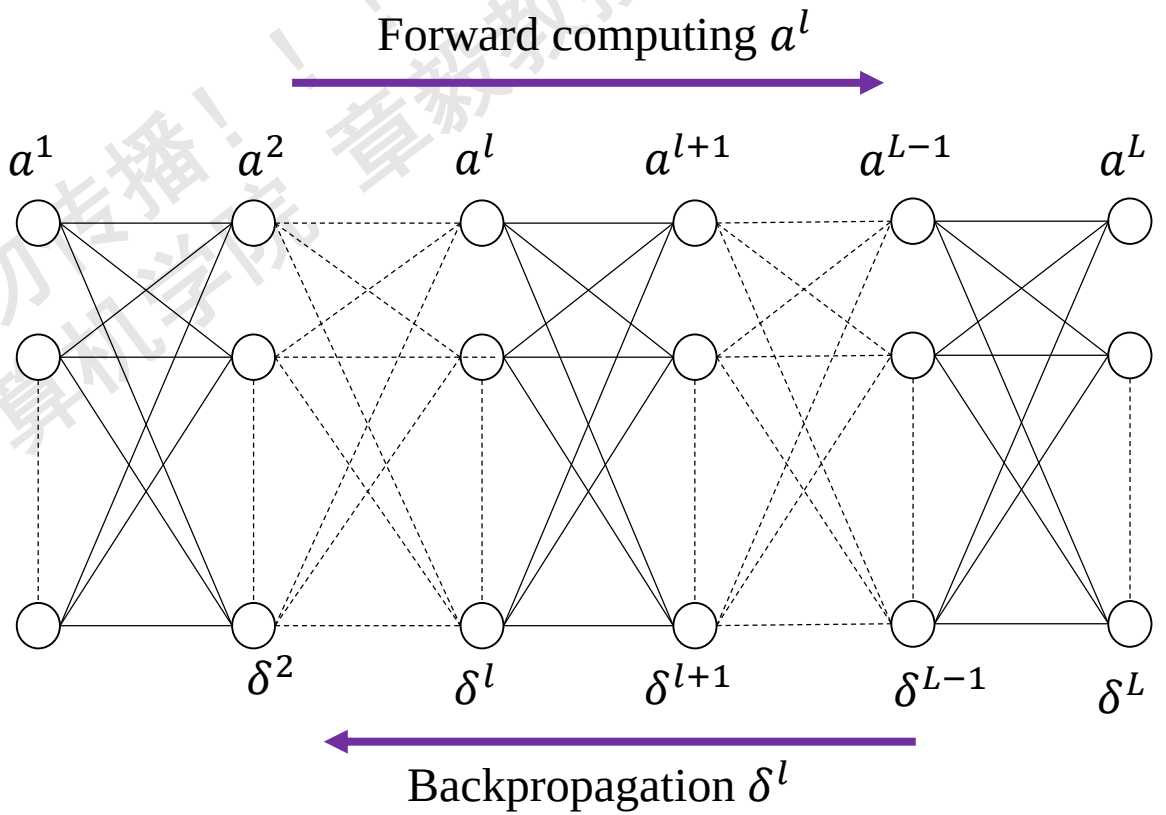
Cost function: $J(w^1, \cdots, w^{L-1})$

Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

Relationship: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$

key:

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$



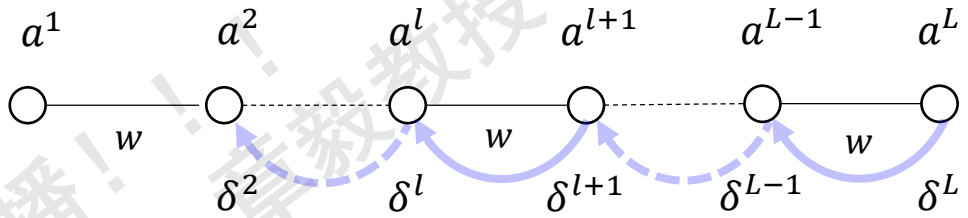
On the Depth of the Networks

Gradient Vanishing Problem

a simple example

$$w = w^{l}$$

$$\delta^{l} = \dot{f}(z^{l}) \cdot w \cdot \delta^{l+1}$$

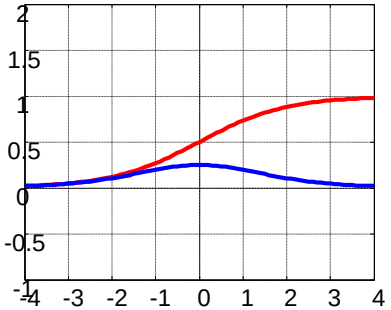


$$\delta^{l} = \dot{f}(z^{l}) \cdot w \cdot \delta^{l+1}$$
$$= \dot{f}(z^{l}) \cdot w \cdot \dot{f}(z^{l+1}) \cdot w \cdot \delta^{l+2}$$
$$= w \cdot \dot{f}(z^{l}) \cdot w \cdot \dot{f}(z^{l+1}) \cdots w \cdot \dot{f}(z^{L-1}) \cdot \delta^{L}$$
$$= \prod_{m=L-1}^{l} \left(w \cdot \dot{f}(z^{m})\right) \cdot \delta^{L}$$

$$\left|\frac{\partial \delta^{l}}{\partial \delta^{L}}\right| = \prod_{m=L-1}^{l} \left|w \cdot \dot{f}(z^{m})\right| \leq |w|^{L-l+1} \cdot (0.25)^{L-l+1}$$

$\dot{f}(z^{m}) \leq 0.25$

Sigmoid

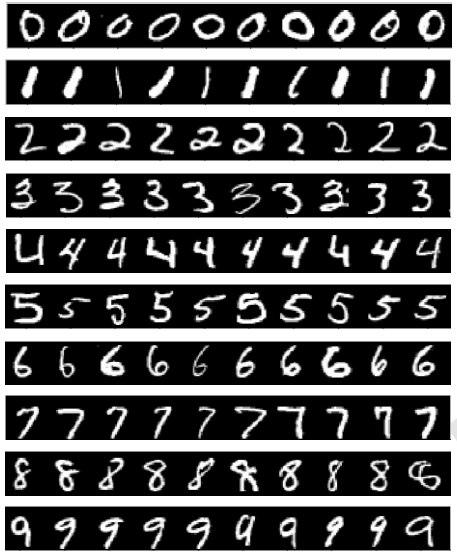


Notes：

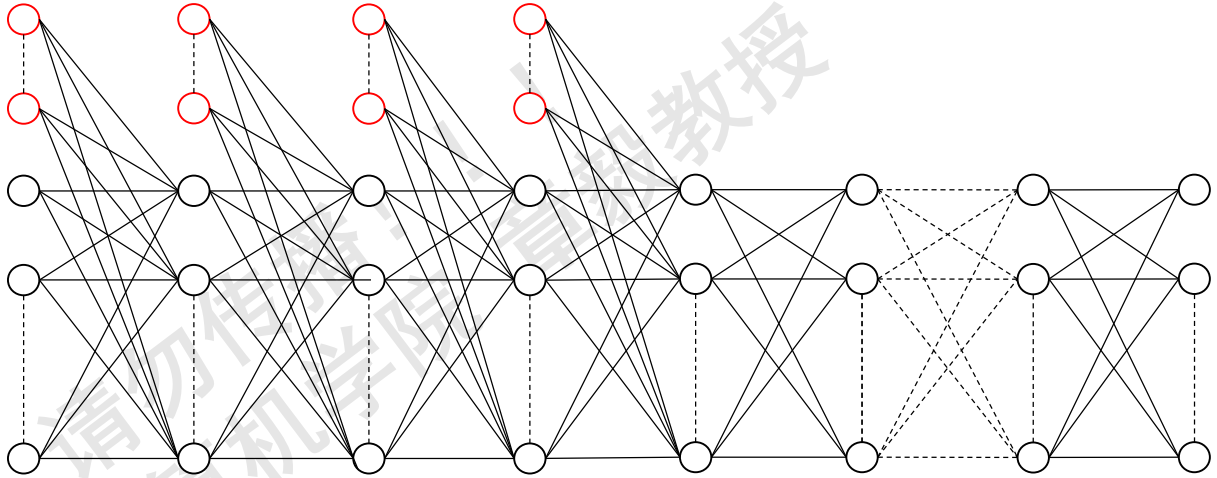
The exponential descent of δ^{l} causes the gradient vanish problem.

On the Depth of the Networks

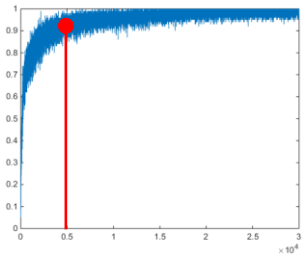
The depth of the network is correlated to the problem.



Handwritten digits recognition problem

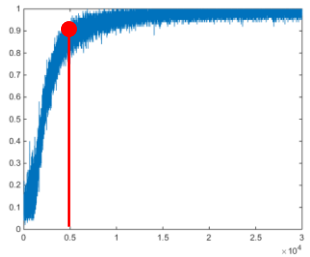


5 layers



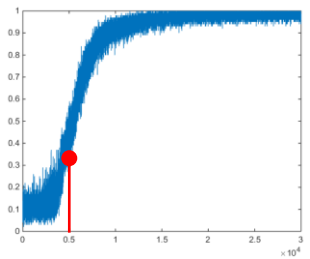
Accuracy
- Training=97.55%
- Testing=95.25%

8 layers



Accuracy
Training=98.65%
Testing=95.10%

9 layers



Accuracy
- Training=98.45%
- Testing=93.20%

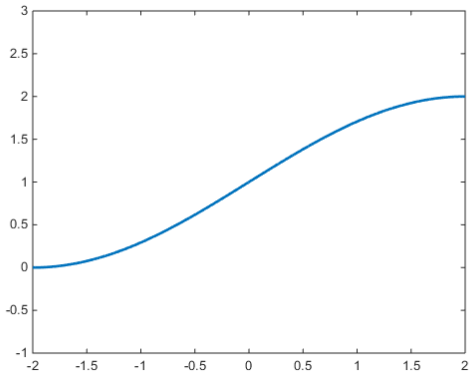
Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

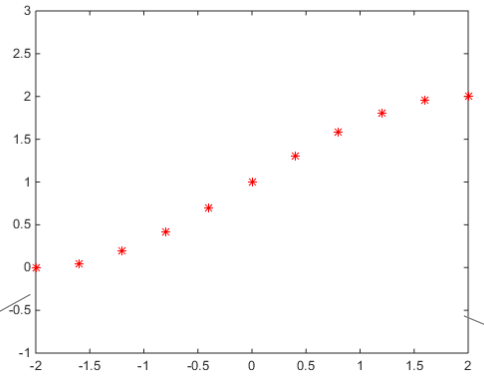
On the Training Data

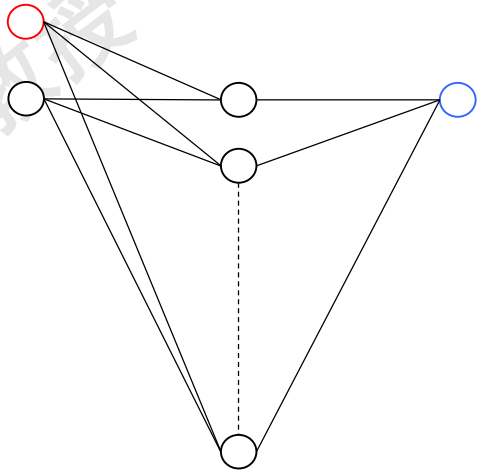
Using a 2-9-1 network to fit a partial sin curve

$$y = g(x) = 1 + \sin\left(\frac{\pi}{4}x\right), x \in [-2,2]$$



Samples





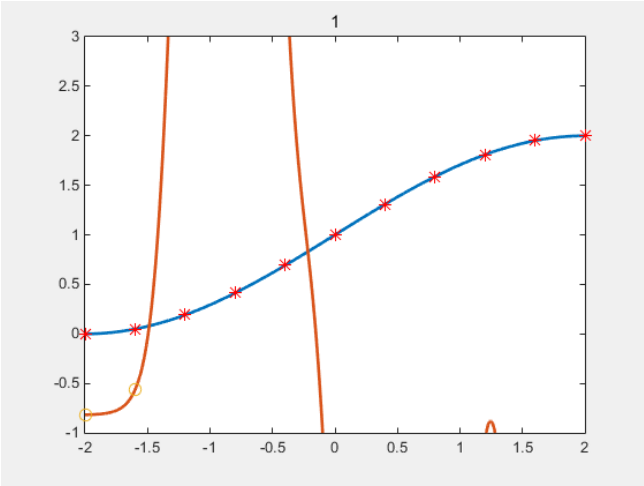
(1,1) → (0,9) → (0,1)

	1	2	3	4	5	6	7	8	9	10	11
x	-2	-1.6000	-1.2000	-0.8000	-0.4000	0	0.4000	0.8000	1.2000	1.6000	2
y	0	0.0489	0.1910	0.4122	0.6910	1	1.3090	1.5878	1.8090	1.9511	2

11 samples

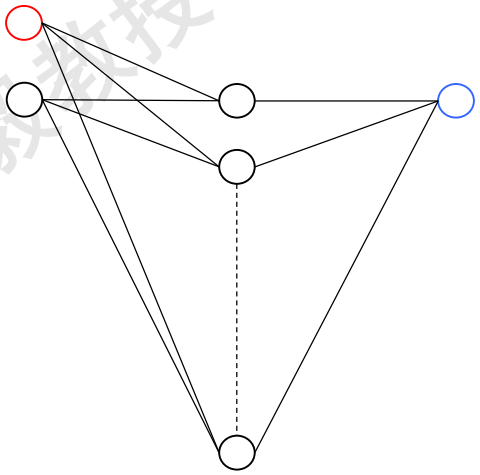
On the Training Data

Overfitting



11 data samples

Using a 2-9-1 network to fit a partial sin curve

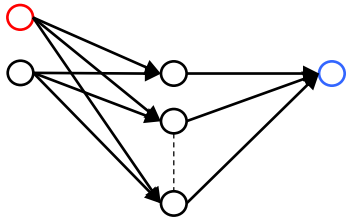


2-9-1 network has 27 weights to be tuned.
In general, we need more samples than the number of unknown parameters in a system.

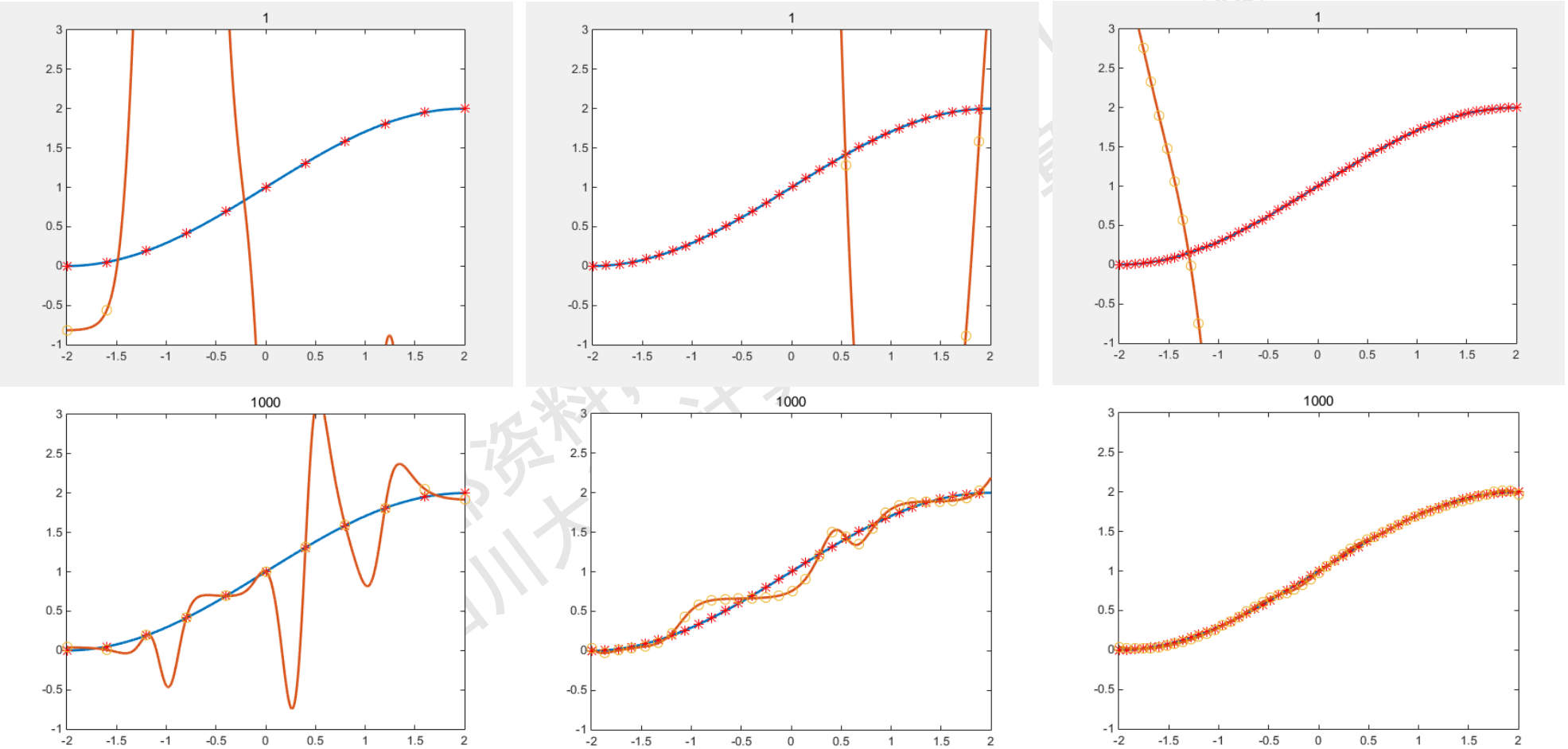
The network fit the data sample properly, but nowhere else on the curve! Overfitting!

- Fit training data well
- Cannot fit testing data
- We need **MORE** data!

On the Training Data



(1,1) → (0,9) → (0,1)

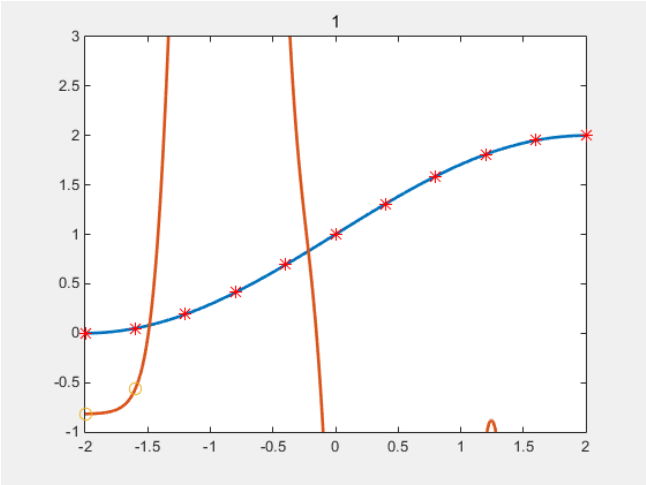


11 data samples

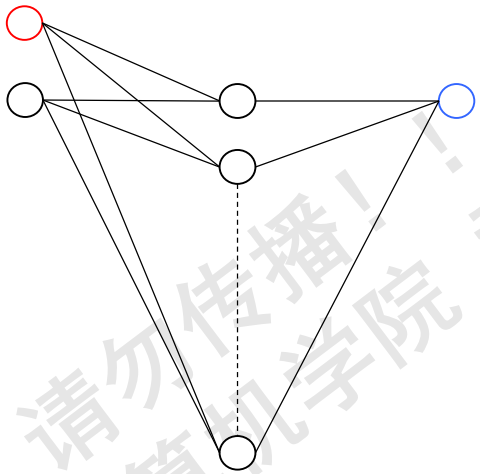
23 data samples

51 data samples

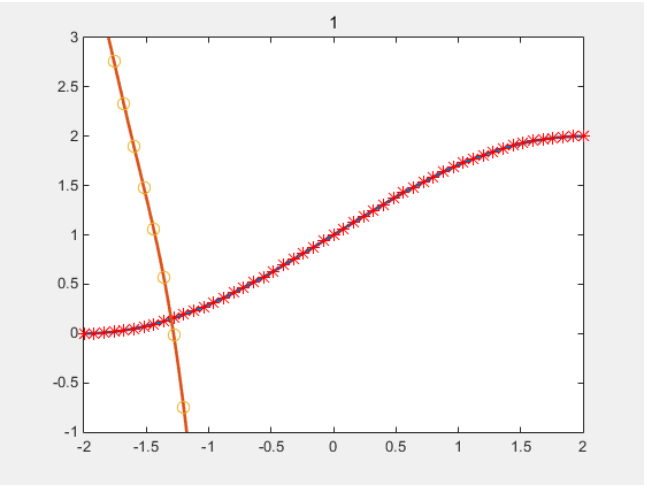
On the Training Data



11 data samples



For a network to be able to generalize, it should have fewer parameters than there are data points in the training set.



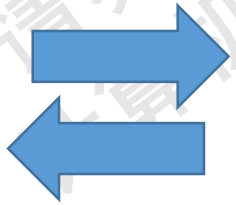
51 data samples

On the Training Data

Big data

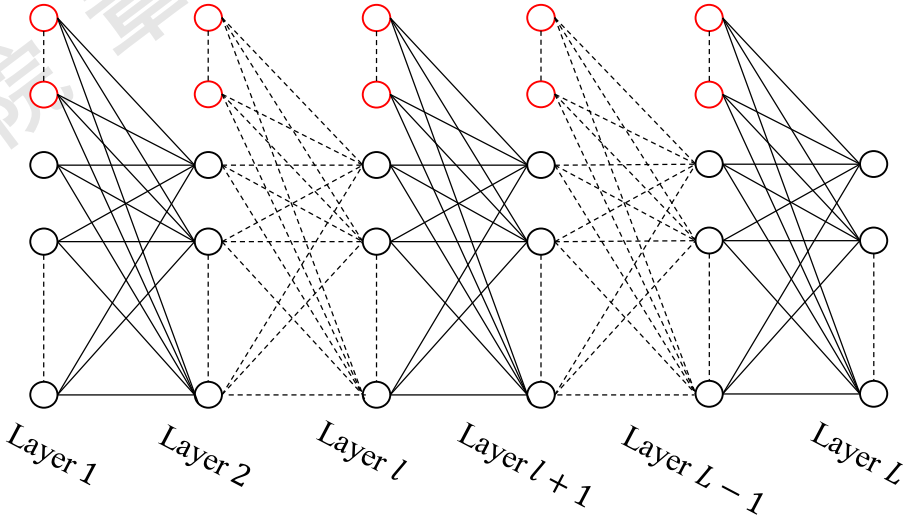
Complex patterns in **big data** need complex model to deal with.

Abundant data sample for training model (samples)



Highly nonlinear, flexible, and trainable model (complexity)

DNN



Huge number of parameters in **DNN** models need to be determined.

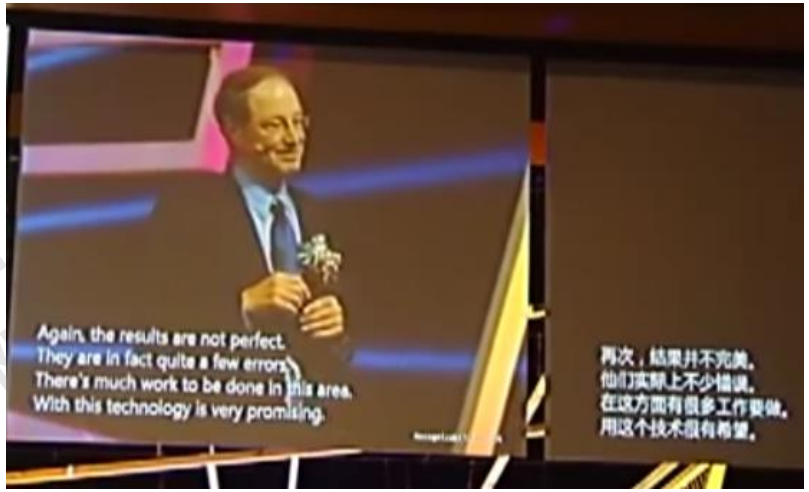
Big data + DNN Example

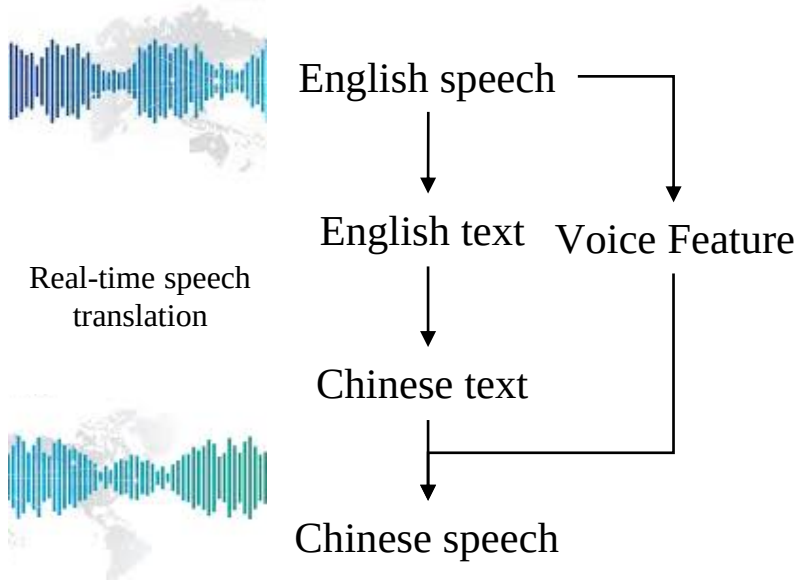
Speech Recognition

1950s Wave of speech + pattern recognition = few words

1970s Gaussian Mixture Model + Hidden Markov Model = ~80% recognition rate

2011 Deep neural network for modeling speech = awesome real-time recognition!



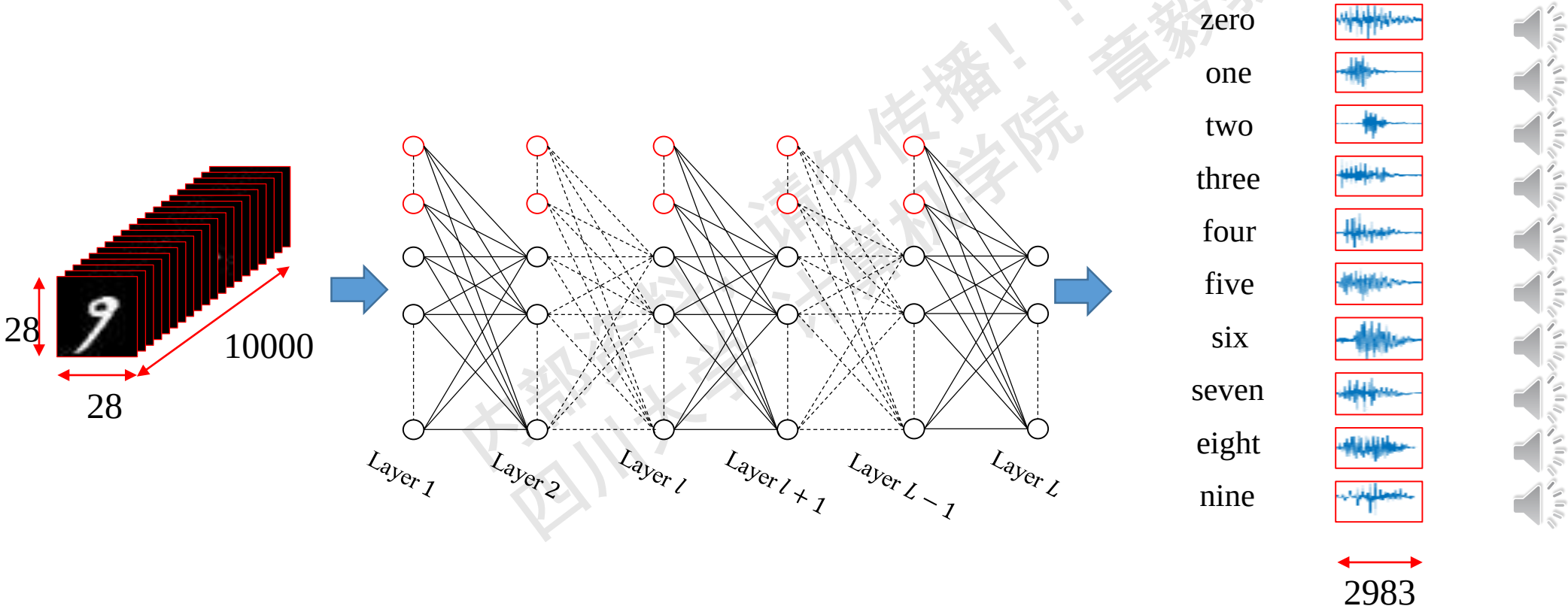


Outline

- Brief Review of Backpropagation Algorithm
- On Some Problems of BP
  - On the Target Output
  - On the Network Output
  - On the Input
  - On the Cost Function
  - On the Depth of the Network
  - On the Training Data
- Assignment

Assignment

Implement the handwritten digits to speech convertor by MATLAB.

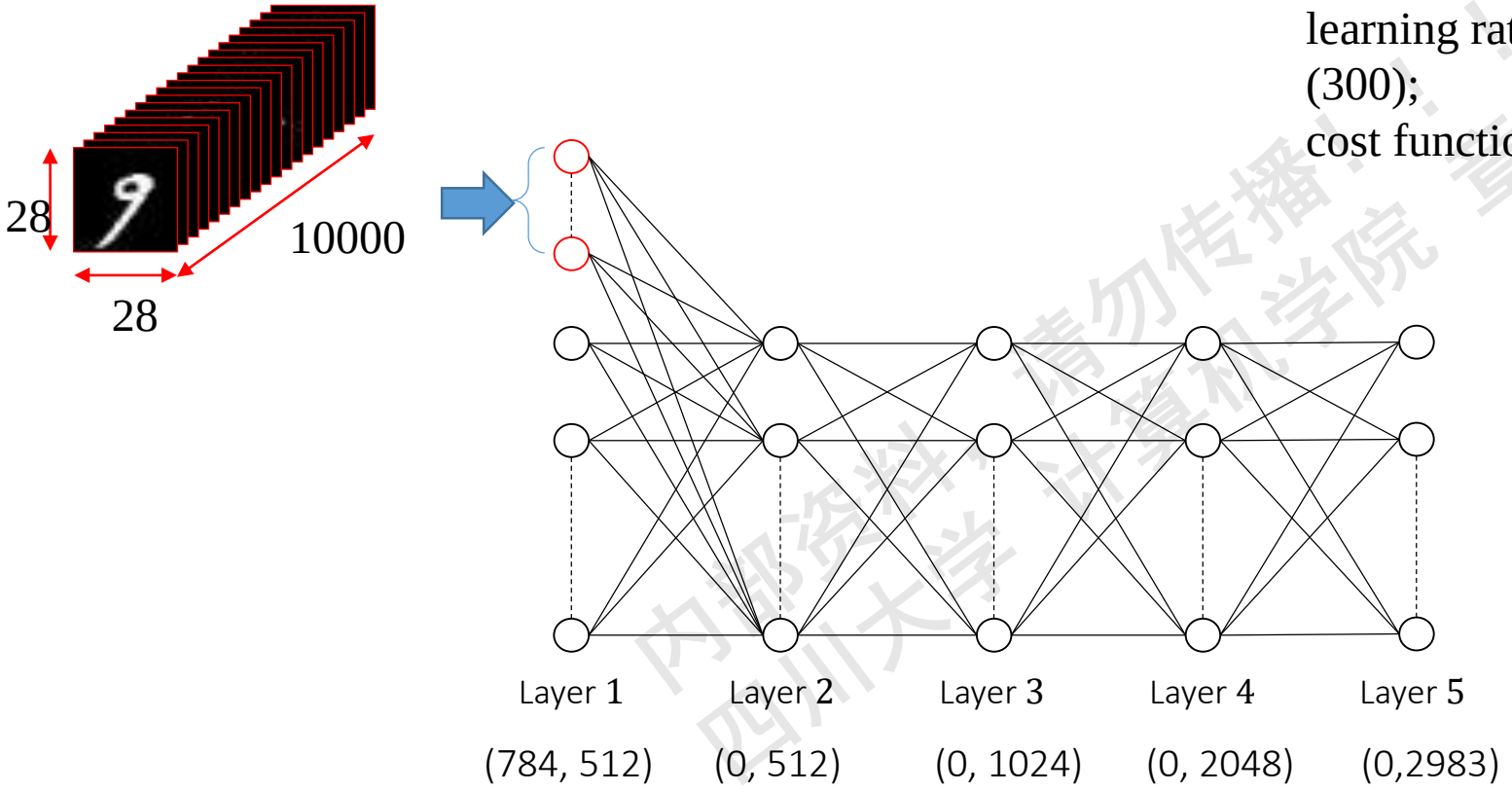


Thanks

Assignment: an example

Hint: One of my student used the following parameters for the network and successfully trained the network.

learning rate (0.1); mini batch (100); iteration (300);
cost function (mean square error).



Assignment: codes

fc.m %%forward computation file

```
function [a_next, z_next] = fc(w, a, x, f)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Your code BELOW
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % forward computing (either component or vector form)
    z_next = w * [x; a];
    a_next = f(z_next);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Your code ABOVE
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

Assignment: codes

bc.m %%backward computation file

```
function delta = bc(w, z, delta_next, df)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Your code BELOW
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % backward computing (either component or vector form)
    delta = df(z) .* (w(:, end-size(z,1)+1:end)' * delta_next);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Your code ABOVE
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

Assignment: codes

get_audio.m %%the function to get audio file

```
function [ audio_y ] = get_audio( y, audio )

audio_y = zeros(size(audio,2), size(y,2));
for i = 1:size(y,2)
    audio_y(:,i) = audio(find(y(:,i)==1),:);
end
```

Assignment: codes

lab5.m %%the main training function

```
% clear workspace and close plot windows
clear;
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your code BELOW
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare the data set
load mnist_small_matlab.mat
input_size = 28 * 28; % size of each patch
% prepare training data
train_size = size(trainLabels,2);
X_train{1} = reshape(trainData,[],train_size);% top-left
X_train{2} = zeros(0, train_size);
X_train{3} = zeros(0, train_size);
X_train{4} = zeros(0, train_size);
X_train{5} = zeros(0, train_size);
```

```
% prepare testing data
test_size = size(testLabels,2);
X_test{1} = reshape(trainData,[],test_size);% top-left
X_test{2} = zeros(0, test_size);
X_test{3} = zeros(0, test_size);
X_test{4} = zeros(0, test_size);
X_test{5} = zeros(0, test_size);

% prepare standard speech audio
sample_rate = 4000; % shall assert they all have a same sample rate
audio = zeros(2983, 10); % we checked with the audio file and know its 2983-dim
input
for i = 1:10
    [audio(:,i), sample_rate] = audioread(fullfile('audio',sprintf('%d.wav',i-1)));
    soundsc(audio(:,i), sample_rate);
    pause(1)
end
audio = (audio+1)/2;

% choose parameters
alpha = 0.1; % learning rate
max_iter = 300;
mini_batch = 100;
```

```
layer_size = [input_size 512    % layer 1
                         0 512    % layer 2
                         0 1024    % layer 3
                         0 2048    % layer 4
                         0 2983];  % layer 5
L = size(layer_size, 1);
% define function
sigm = @(s) 1 ./ (1 + exp(-s));
dsigm = @(s) sigm(s) .* (1 - sigm(s));
lin = @(s) s;
dlin = @(s) 1;
fs  = {[],  sigm,  sigm,  sigm,  sigm,  sigm,  sigm,  sigm};
dfs = {[], dsigm, dsigm, dsigm, dsigm, dsigm, dsigm, dsigm};
% initialize weights
w = cell(L-1, 1);
for l = 1:L-1
    %w{l} = randn(layer_size(l+1,2), sum(layer_size(l,:)));
    % a tricky, but effective, initialization
    w{l} = (rand(layer_size(l+1,2), sum(layer_size(l,:))) * 2 -1) *
sqrt(6/(layer_size(l+1,2)+sum(layer_size(l,:))));
end

% train
J = [];
x = cell(L, 1);
a = cell(L, 1);
z = cell(L, 1);
delta = cell(L, 1);
```

```
for iter = 1:max_iter
    ind = randperm(train_size);
    % for each mini-batch
    for k = 1:ceil(train_size/mini_batch)
        % prepare internal inputs
        a{1} = zeros(layer_size(1,2),mini_batch);
        % prepare external inputs
        for l=1:L
            x{l} = X_train{l}(:,ind((k-1)*mini_batch+1:min(k*mini_batch, train_size)));
        end
        % prepare labels
        [~, ind_label] = max(trainLabels(:,ind((k-1)*mini_batch+1:min(k*mini_batch, train_size))));
        % prepare targets
        y = audio(:,ind_label);

        % batch forward computation
        for l=1:L-1
            [a{l+1}, z{l+1}] = fc(w{l}, a{l}, x{l}, fs{l+1});
        end

        % cost function and error
        J = [J 1/2/mini_batch*sum((a{L}(:)-y(:)).^2)];
        delta{L} = (a{L} - y).* dfs{L}(z{L});

        % batch backward computation
        for l=L-1:-1:2
            delta{l} = bc(w{l}, z{l}, delta{l+1}, dfs{l});
        end
        % update weight
        for l=1:L-1
            gw = delta{l+1} * [x{l};a{l}]' / mini_batch;
            w{l} = w{l} - alpha * gw;
        end
    end
```

```
 % end loop
    if mod(iter,1) == 0
        fprintf('%i/%i epochs: J=%.4f\n', iter,
max_iter, J(end));
    end
end

% save model
save model.mat w layer_size J
```