The Introduction To Artificial Intelligence

Yuni Zeng yunizeng@zstu.edu.cn

2024-2025-1

The Introduction to Artificial Intelligence

- Part I Brief Introduction to AI & Different AI tribes
- Part II Knowledge Representation & Reasoning
- Part III AI GAMES and Searching
- Part IV Model Evaluation and Selection
- Part V Machine Learning
- Part VI Neural Networks

Neural Networks

- *Brief review*
- Feedforward Neural Networks
- Recurrent Neural Networks
- The Learning of Neural Networks
- Model Performance: Cost Function
- Steepest Descent Method
- Backpropagation

Brief review

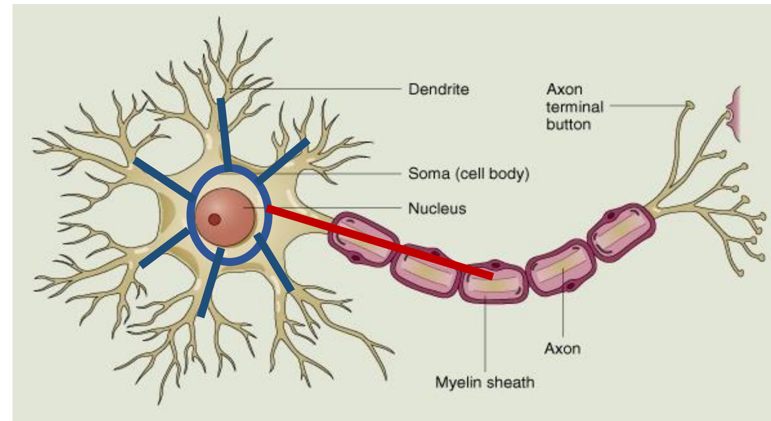
Artificial Neuron

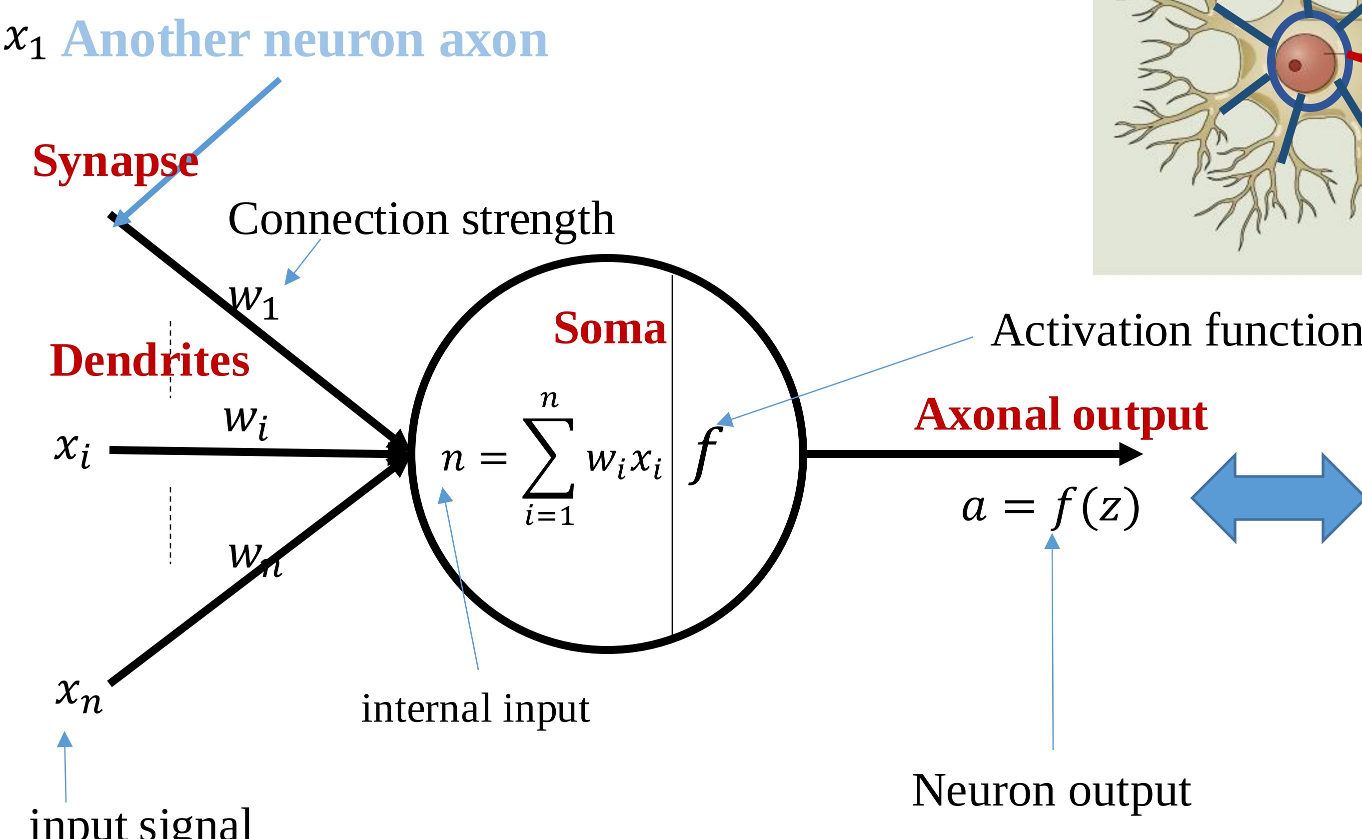
Biological neural network → **Artificial neural networks**

Biological neural network		Artificial neural networks
Neurons		Neuron model
Synaptic Connection	Abstract	Weights
Biological Neural Networks	Build a computable mathematical model	Neural network models
Learning		Learning algorithm

Brief review

Artificial Neuron





$$a = f\left(\sum_{i=1}^{R} w_i x_i\right)$$

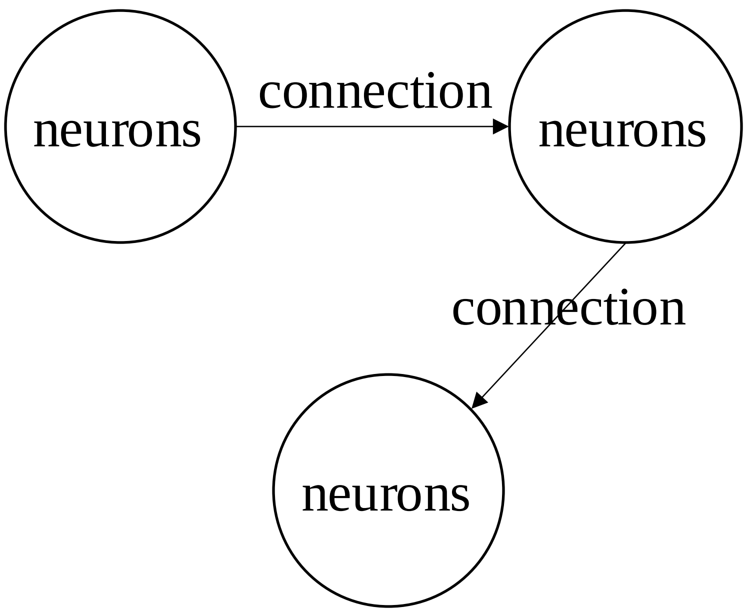
Computational Model of Neural Network

Neural Networks

Feedforward neural network

=

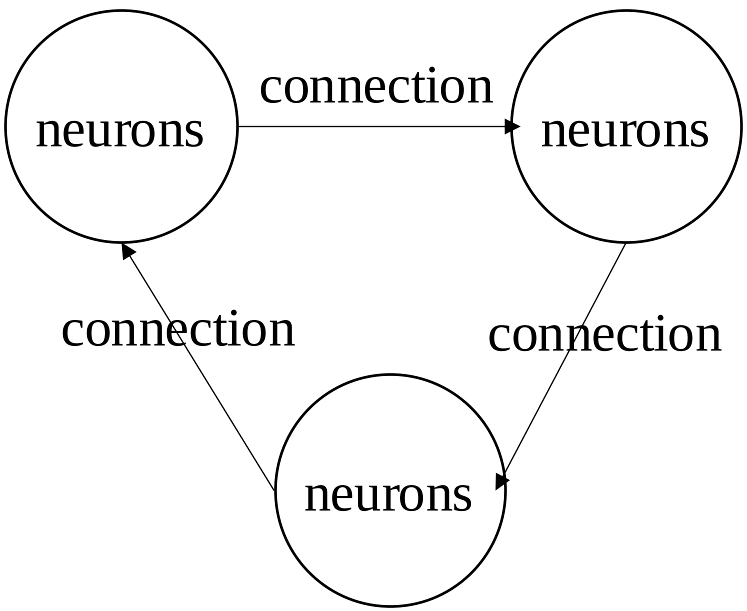
neurons + feedforward connections



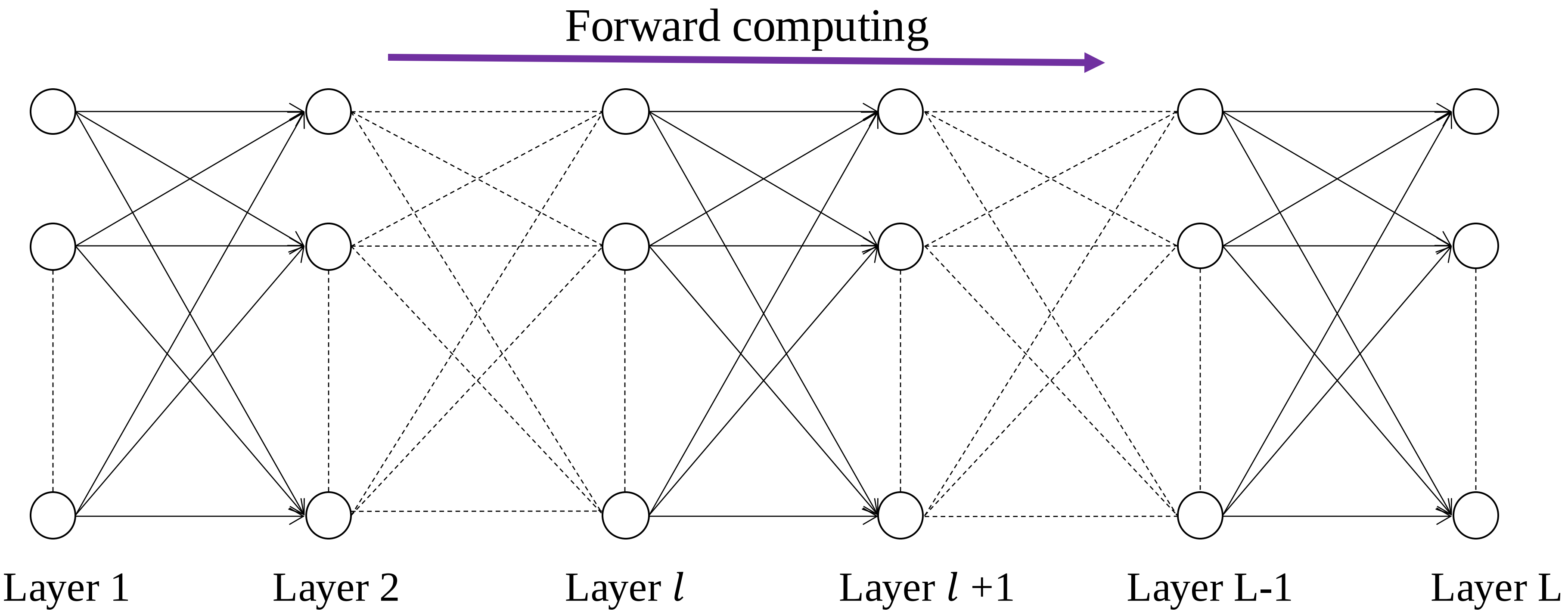
Recurrent neural network

=

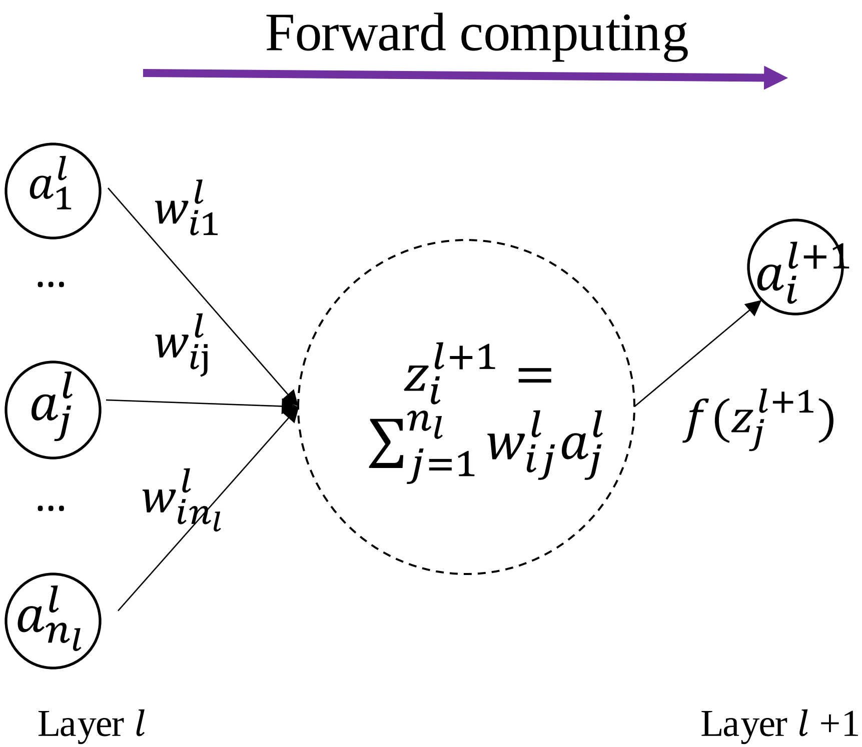
neurons + recurrent connections

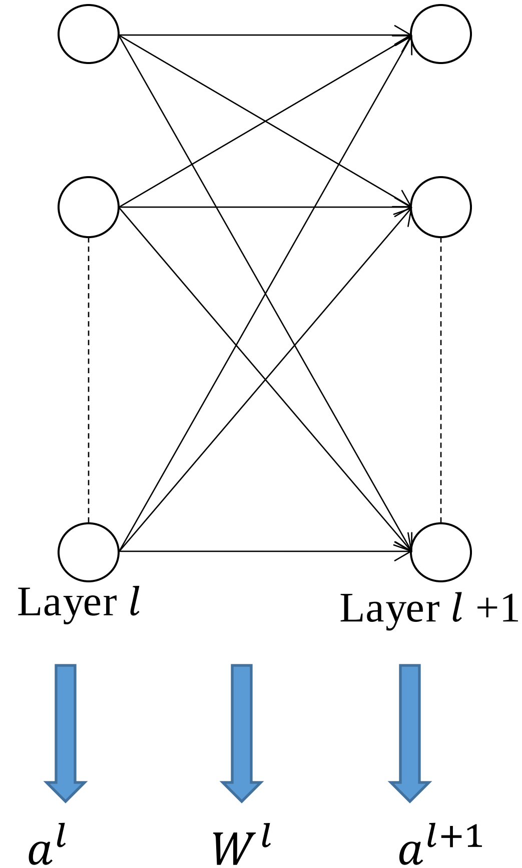


Feedforward Neural Network



Feedforward Neural Network





Component form

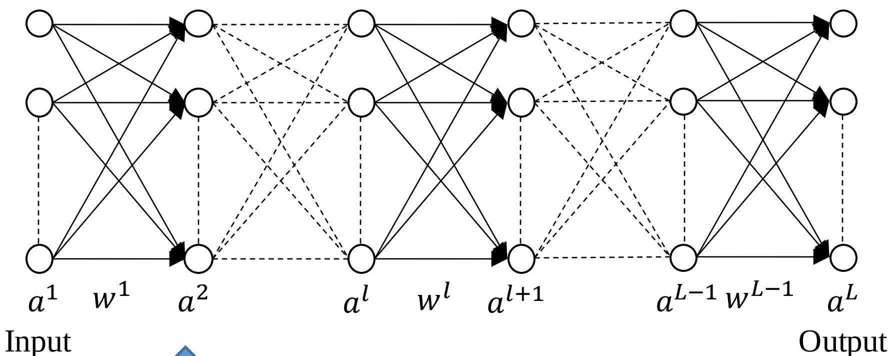
$$\begin{cases} a_i^{l+1} = f(z_i^{l+1}) \\ z_i^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l a_j^l \end{cases}$$

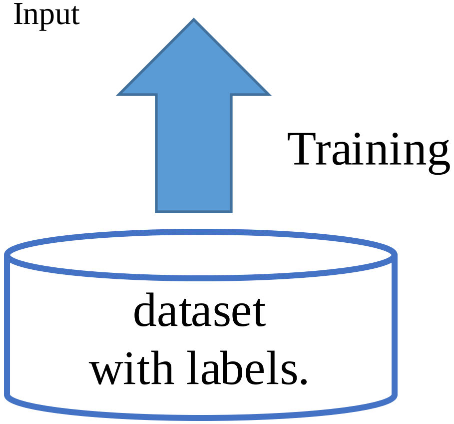
Vector form

$$\begin{cases} a^{l+1} = f(z^{l+1}) \\ z^{l+1} = W^l a^l \end{cases}$$

Model Performance: Cost Function

Cost Function





The goal of Learning:

Network output ≈ Target output

Cost Function $J(a^L, y^L)$:

- describe the distance between network output a^L and target output y^L
- $J(a^L, y^L)$ is a function related to $(w^1, \cdots, w^{L-1})$

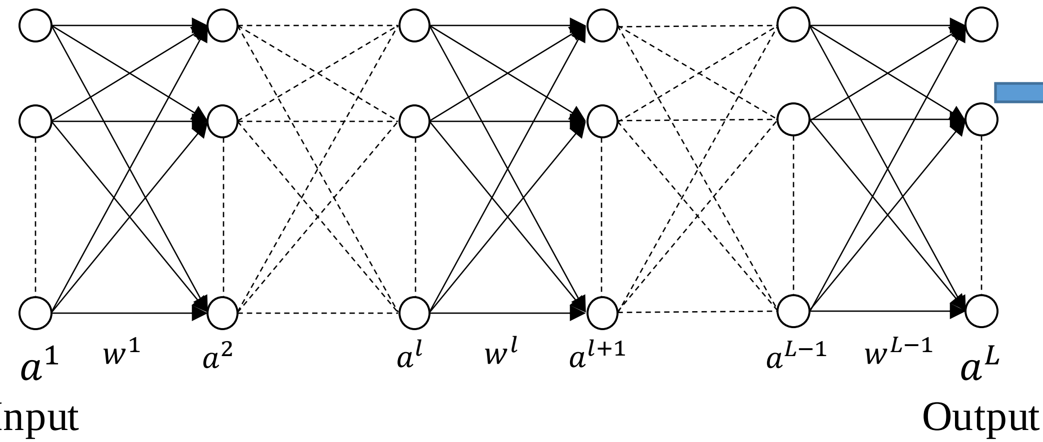
$$J = J(w^1, \cdots, w^{L-1})$$

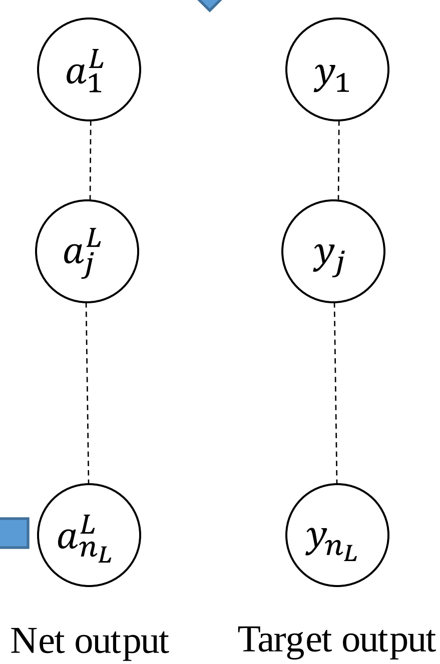
Steepest Descent Method

Deep learning

Steepest Descent Algorithm:

$$w^{k+1} = w^{k} - \alpha_k \cdot \left.\frac{\partial F}{\partial w}\right|_{w^k}$$





Construct cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L}\left(y_j - a_j^L\right)^2$$

Computing gradient

$$\frac{\partial J}{\partial w_{ji}^l}$$

Updating weights

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

Backpropagation

Conclusion: BP for FNN

Forward computing: $y = f(\sum_{i=1}^{n} w_i x_i)$

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

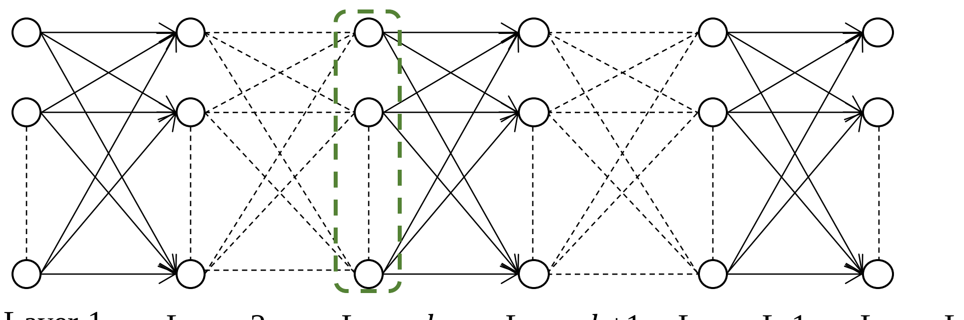
Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

Define δ: $\delta_i^l = \frac{\partial J}{\partial z_i^l}$

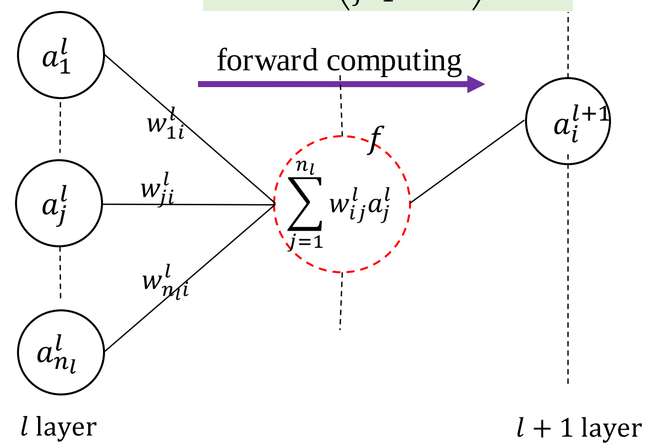
Find the relation: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$

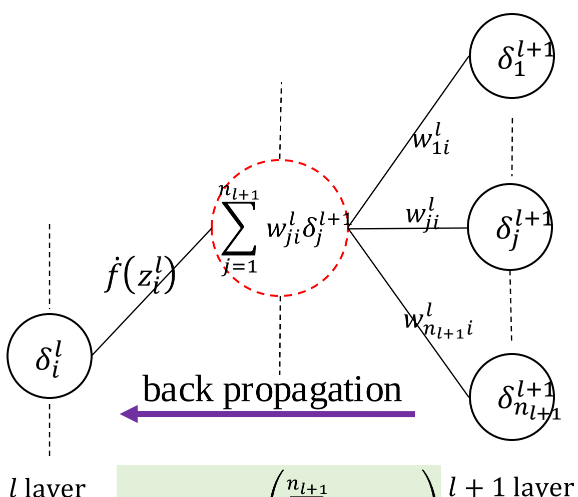
Back propagation: $\delta_i^L = \frac{\partial J}{\partial z_i^L} = (a_i^L - y_i^L) \cdot \dot{f}(z_i^L)$

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



$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l \right)$$



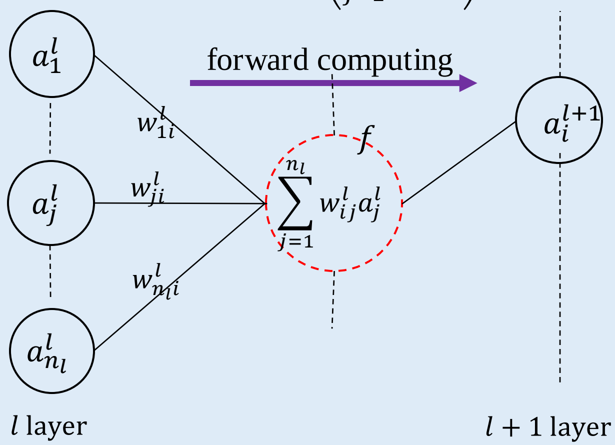


$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

Backpropagation

Conclusion: BP for FNN

$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right)$$



function $fc(W^l, a^l)$
$for\ i = 1{:}\,n_{l+1}$

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

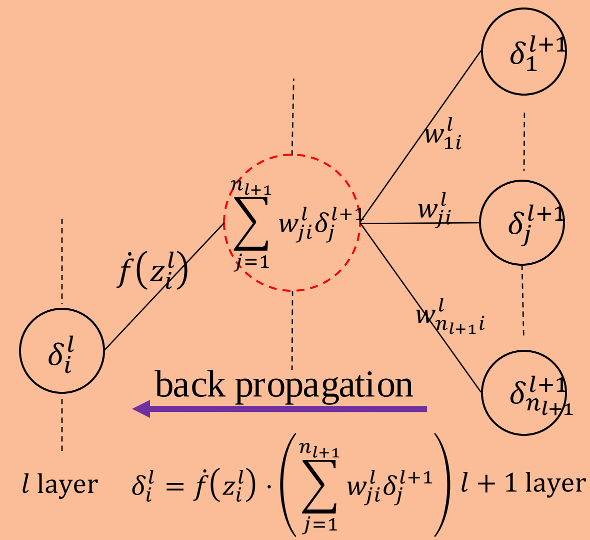
$$a_i^{l+1} = f(z_i^{l+1})$$

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



Backpropagation

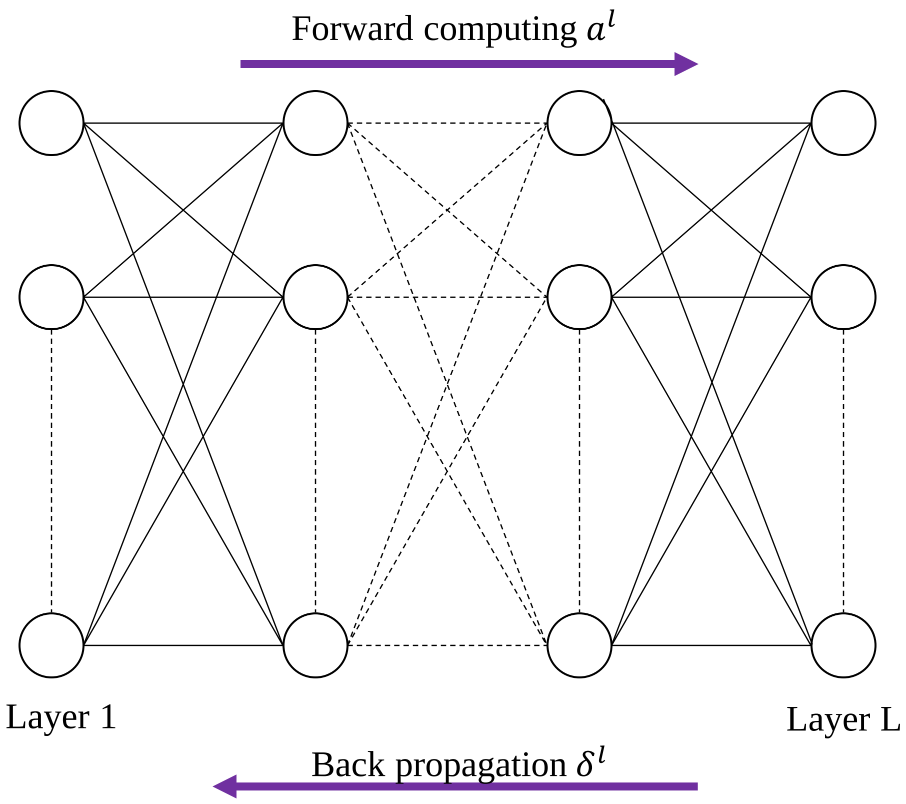
Algorithm

The training data set
$D = \{(x, y) | m\ samples\}$

x: $input\ sample$
y: $target\ output$

There are two ways to train the network.

1. Online training: For each sample $(x, y) \in D$, define a cost function, for example, as
$$J(x, y) = \frac{1}{2}\sum_{j=1}^{n_L}(a_j^L - y_j^L)^2$$
2. Batch training: Define cost function as
$$J = \frac{1}{m}\sum_{(x,y)\in D} J(x, y)$$



Backpropagation

Algorithm

Batch 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 all m samples $(x, y) \in D$, set $a^1 = x$

for $l = 1: L-1$

$a^{l+1} \leftarrow fc(w^l, a^l)$

end

$$\delta^L \leftarrow \frac{\partial J}{\partial z^L}$$

for $l = L-1: 1$

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

end

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

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 $i = 1: n_l$

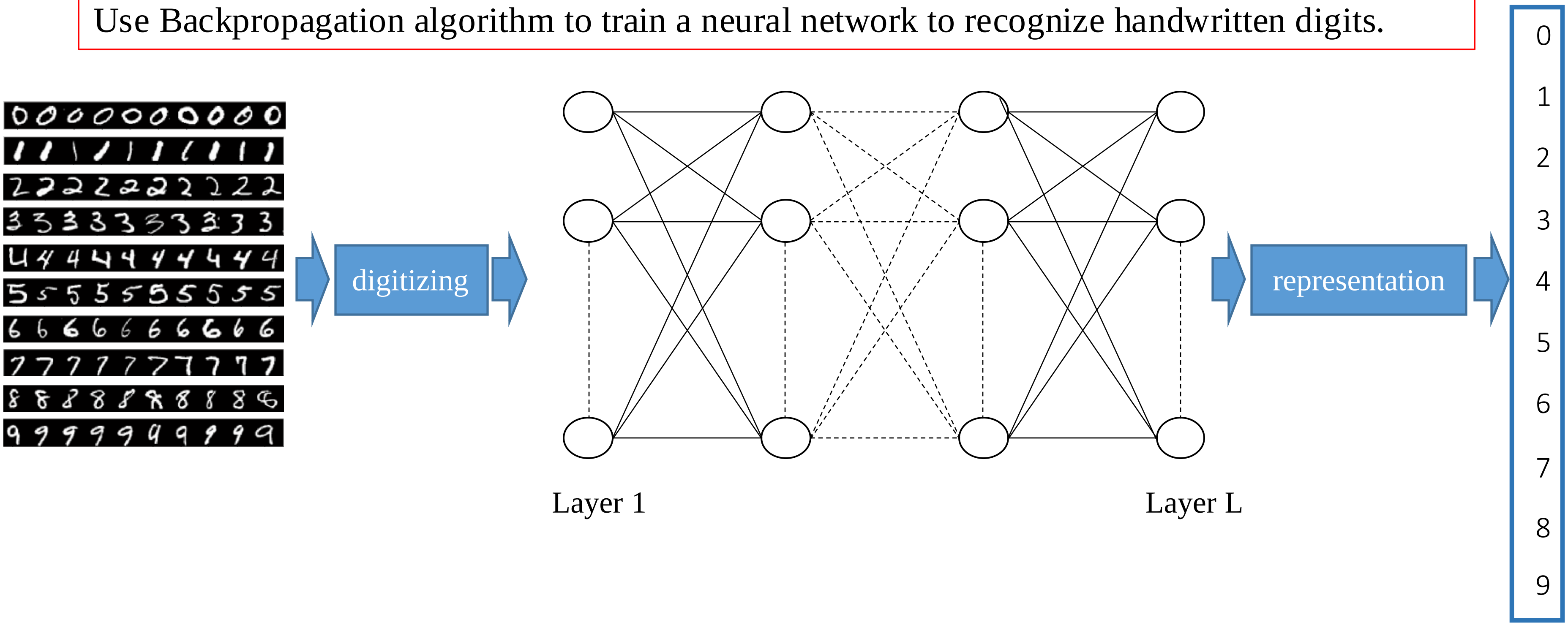
$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^{l} \delta_j^{l+1} \right)$$

end

Backpropagation

Example

Task:
Use Backpropagation algorithm to train a neural network to recognize handwritten digits.



Backpropagation

Example – Step 1: prepare data

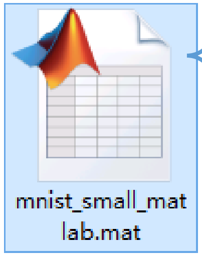
Dataset: **MNIST_small**

MNIST is a database of handwritten digits created by "re-mixing" the samples from MNIST's original datasets. It contains digits written by high school students and employees of the United States Census Bureau. The digits have been size-normalized and centered in 28×28 images.

MNIST_small dataset is a subset of MNIST containing 10000 training samples and 2000 testing samples.

New York University

Download link:
MNIST http://yann.lecun.com/exdb/mnist/
MNIST_small:
https://github.com/kswersky/nnet/blob/master/mnist_small.mat

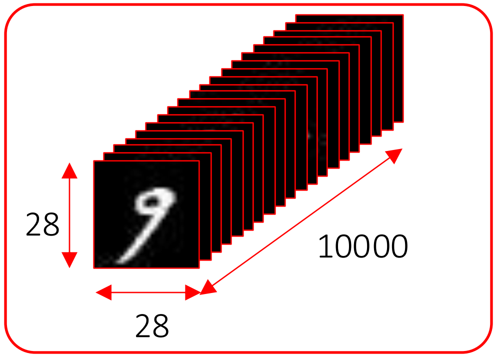


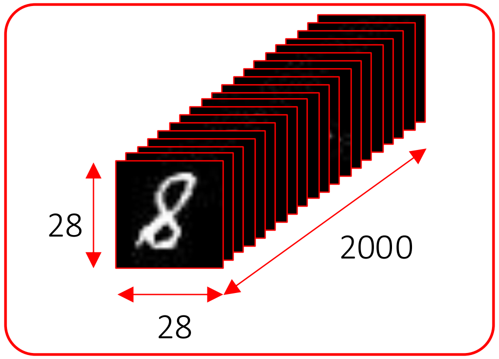
Training set
- Used for training network
- 10000 samples

Testing set
- Used for evaluating network performance
- 2000 samples

The input image is a vector

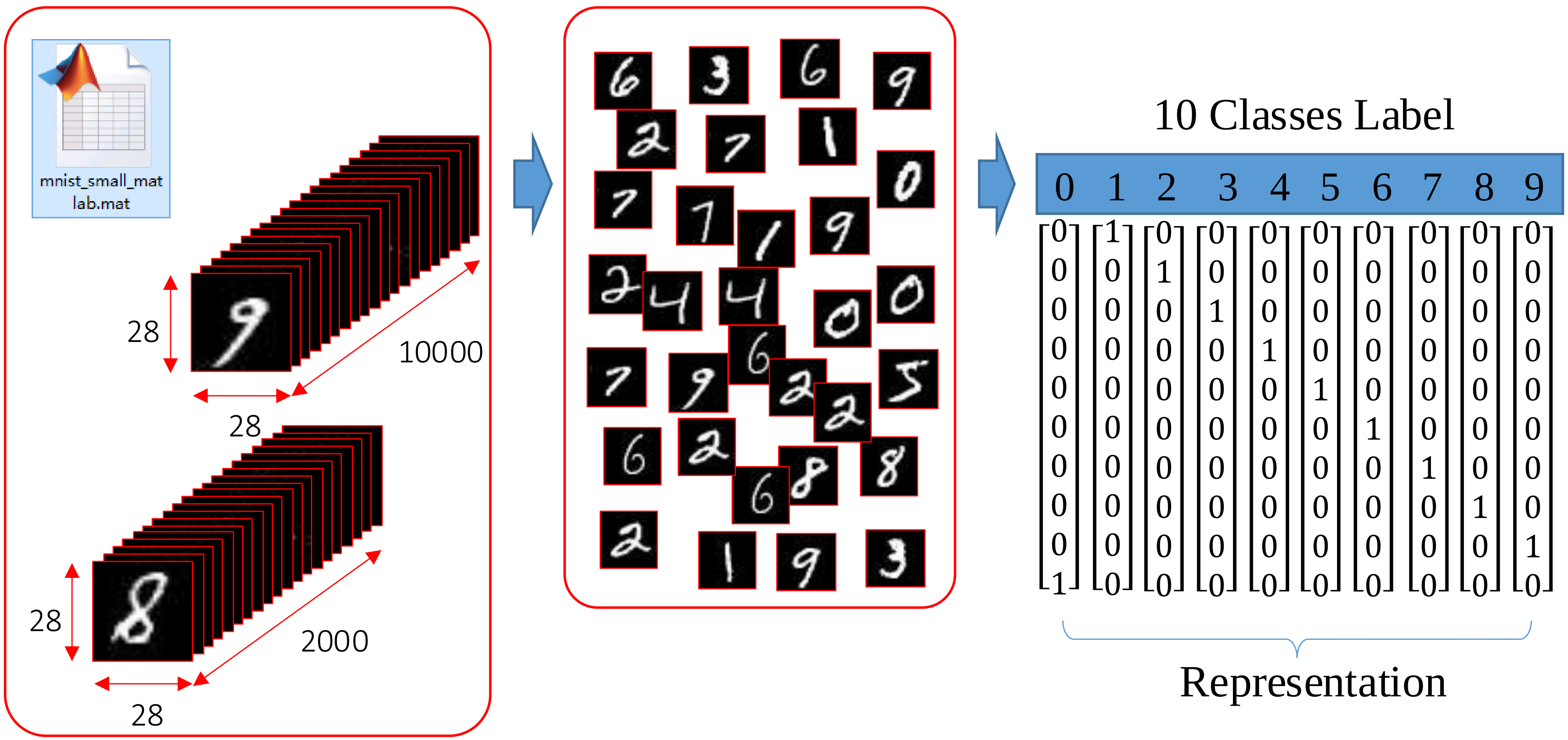
Data





Backpropagation

Example – Step 1: prepare data



Backpropagation

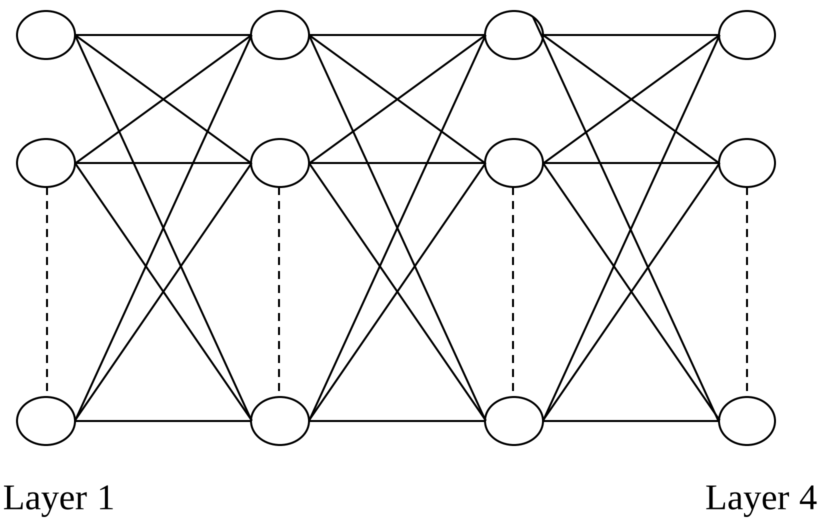
Example --- step 2: Design network architecture

$$a_i^l = f\left(z_i^l\right)$$

Activation function

Network architecture design:

1. Number of layers
2. Number of neurons in each layer
3. Activation function



Number of neurons in the 1st layer = Dimension of an input data

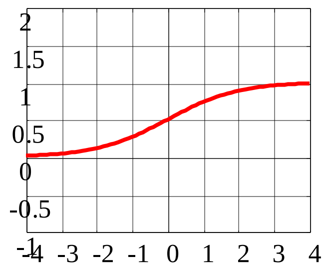
Backpropagation

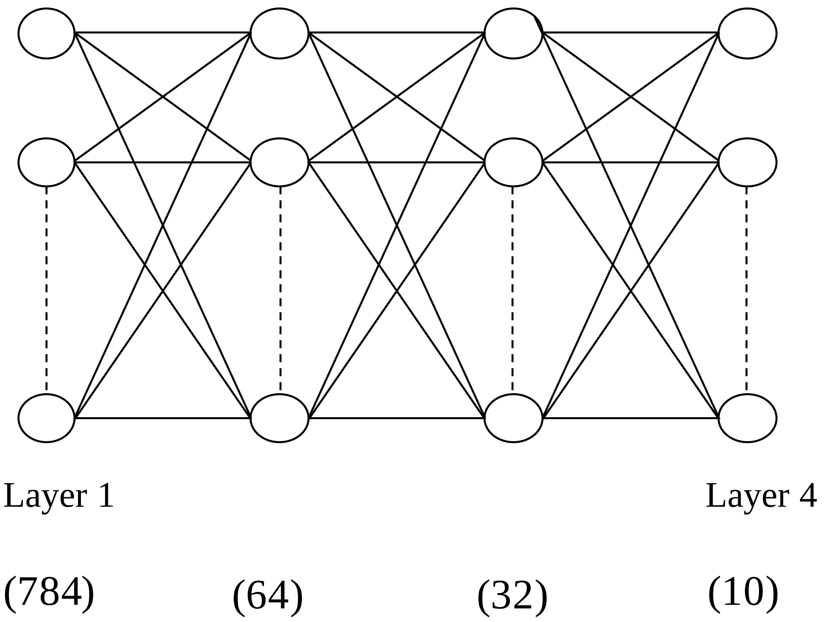
Example --- step 2: Design network architecture

$$a_i^l = f\left(z_i^l\right)$$

Sigmoid function

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





Backpropagation

Example --- step 3: Initial Weights and Learning Rate

Initialize Weight Connections

Random initialization:

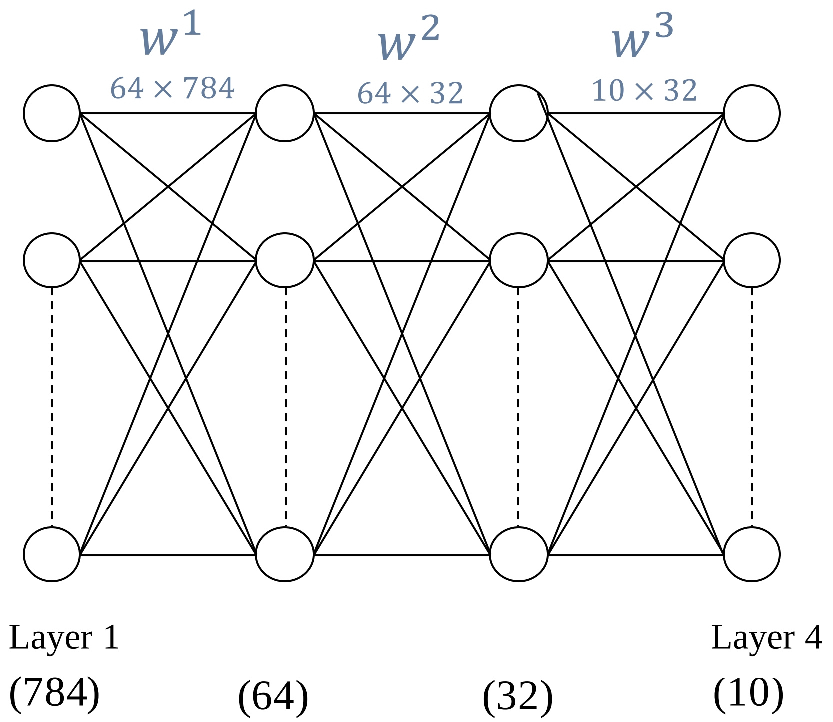
Method 1: Gaussian distribution: $w_{ij}^{l} \sim N(0,1)$

Method 2: Uniform distribution: $w_{ij}^{l} \sim U(-r^{l}, r^{l})$

$$r^{l} = \sqrt{\frac{6}{p^{l} + q^{l+1}}}$$

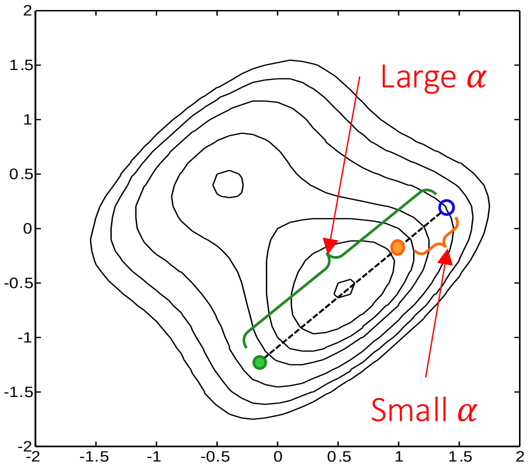
p^{l}: number of neurons in l layer

q^{l+1}: number of internal neurons in $l+1$ layer



Backpropagation

Example --- step 3: Initial Weights and Learning Rate

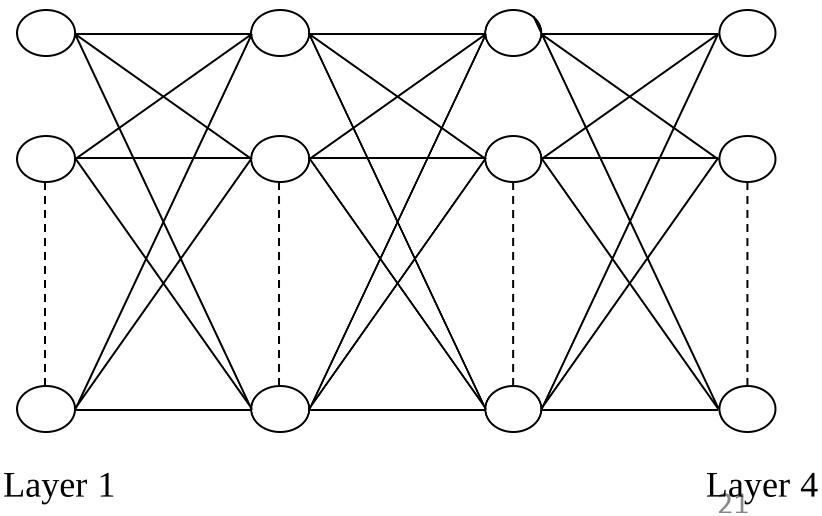


Learning rate:

- Small: slow learning, long learning time.
- Large: fast learning, possibly not converge to minima.

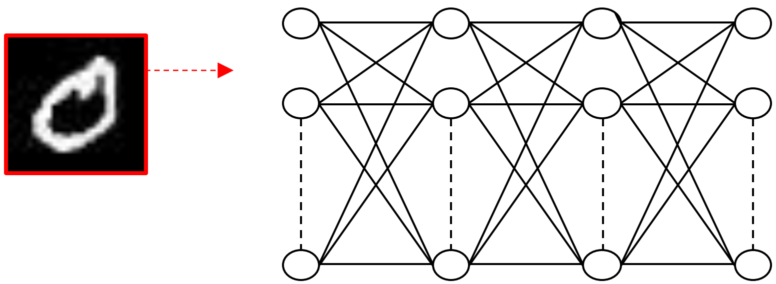
$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \boldsymbol{\alpha} \cdot \frac{\partial J}{\partial w_{ji}^{l}}$$

$$\alpha = \cdots, 0.5, 1, 2, 4, \cdots$$



Backpropagation

Example --- step 4: Cost function



Network output

$$a^L = \begin{bmatrix} 0 \\ 0 \\ 0.1 \\ 0 \\ 0 \\ 0 \\ 0.2 \\ 0 \\ 0 \\ 0.7 \end{bmatrix}$$

Cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2$$

$$e_j = a_j^L - y_j^L$$

Label

0 → representation

Target output

$$y^L = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$e = \begin{bmatrix} 0 \\ 0 \\ 0.1 \\ 0 \\ 0 \\ 0 \\ 0.2 \\ 0 \\ 0 \\ -0.3 \end{bmatrix}$$

Backpropagation

Example --- step 5: Evaluation

$$\text{Accuracy} = \frac{\text{number of correct prediction}}{\text{number of samples}}$$

An example

Tested data

Prediction 7 9 0 4 8 8 8 3 1

Correct prediction 7

Incorrect prediction 2

$$\text{Accuracy} = \frac{7}{9} = 77.78\%$$

Test on training set:

- Reflect the progress of training.
- Evaluate the ability of the model to fit given data.

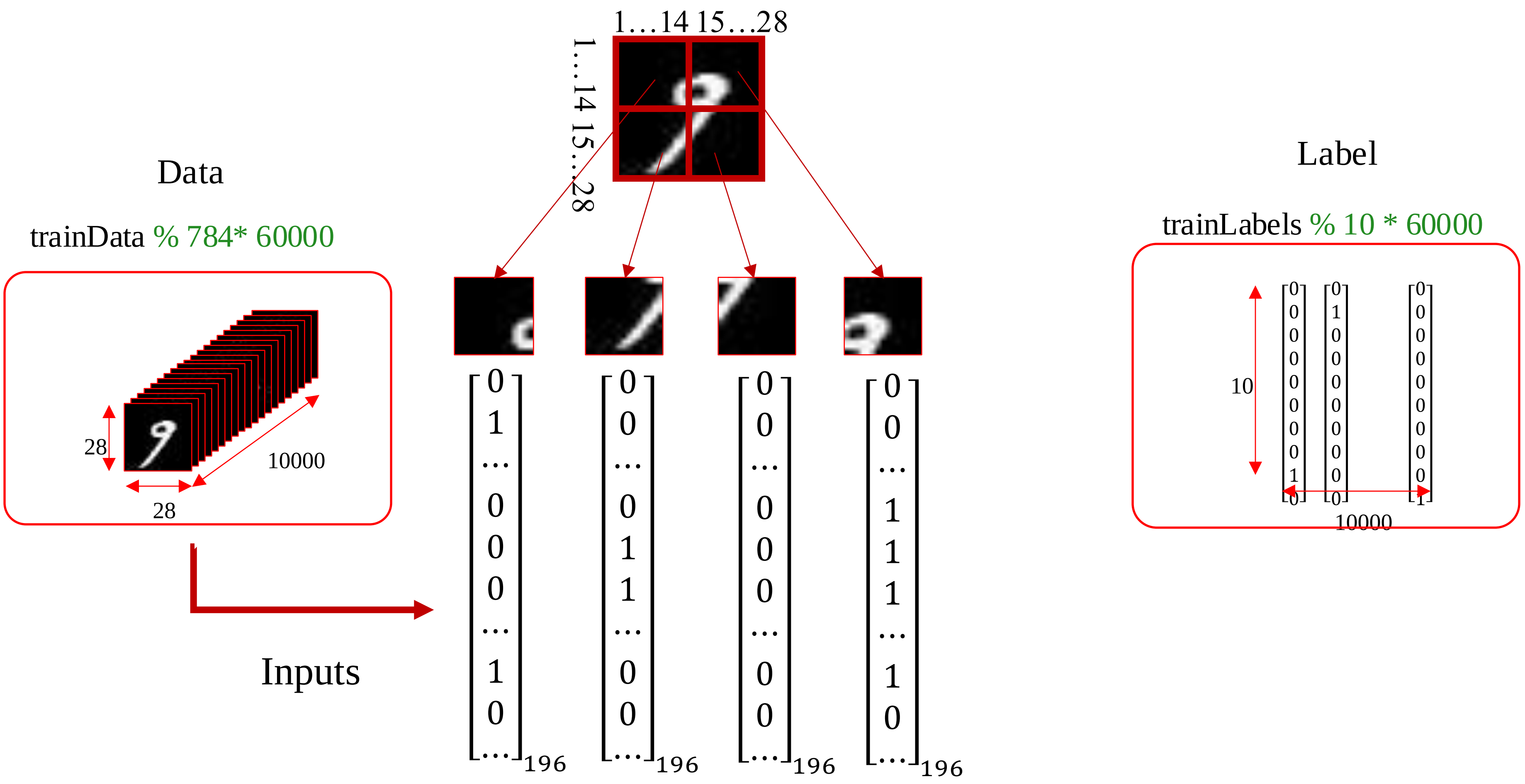
Test on testing set:

- Evaluate the ability of the model to generalize the knowledge.

Backpropagation

Example --- Experiments

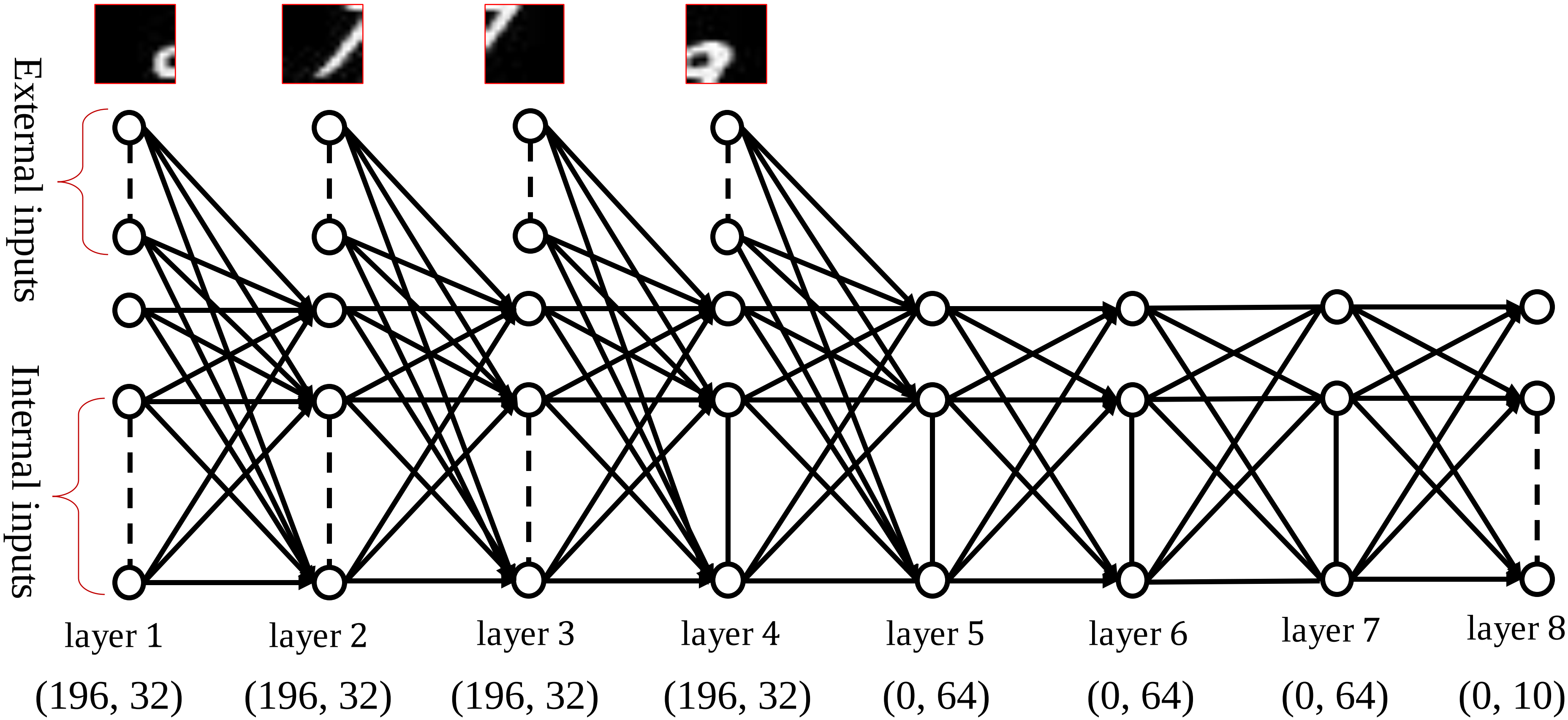
```
% prepare the data set
load ./mnist_small_matlab.mat
```



Backpropagation

Example --- Experiments

```
% define network architecture
L = 8;
```



Backpropagation

Example --- Experiments: Initialize Weights

Gaussian distribution: $w_{ij}^{l} \sim N(0,1)$

```
% initialize weights
for l = 1:L-1
    w{l} = randn(layer_size(l+1,2), sum(layer_size(l,:)));
end
```

Uniform distribution: $w_{ij}^{l} \sim U\left(-r^{l}, r^{l}\right)$

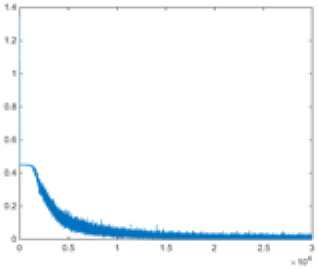
```
% initialize weights
for l = 1:L-1
    % a tricky, but effective, initialization
    w{l} = (rand(layer_size(l+1,2), sum(layer_size(l,:))) * 2 -1)
                * sqrt(6/(layer_size(l+1,2)+sum(layer_size(l,:))));
end
```

Backpropagation

Example --- Experiments: plotting

Cost function

$$J = \frac{1}{2}\sum_{j=1}^{n_L}\left(a_j^L - y_j^L\right)^2$$

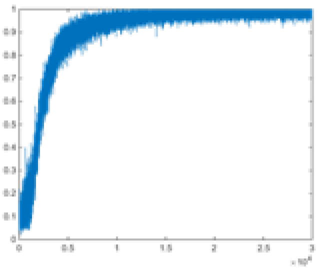


```
% cost function
J = [J 1/2/mini_batch*sum((a{L}(:)-y(:)).^2)];
figure
plot(J);
```

Accuracy

$$\text{Acc} = \frac{number\ of\ correct\ prediction}{number\ of\ samples}$$

Use max output as prediction



```
% accuary on training batch
[~,ind_train] = max(y);
[~,ind_pred] = max(a{L});
Acc= [Acc sum(ind_train == ind_pred) /
mini_batch];
figure
plot(Acc);
```

Neural Networks

- Brief review
- Feedforward Neural Networks
- Recurrent Neural Networks
- The Learning of Neural Networks
- Model Performance: Cost Function
- Steepest Descent Method
- Backpropagation