Artificial Neural Networks

Jen-Ing Hwang
Department of Computer Science and
Information Engineering
Fu Jen Catholic University

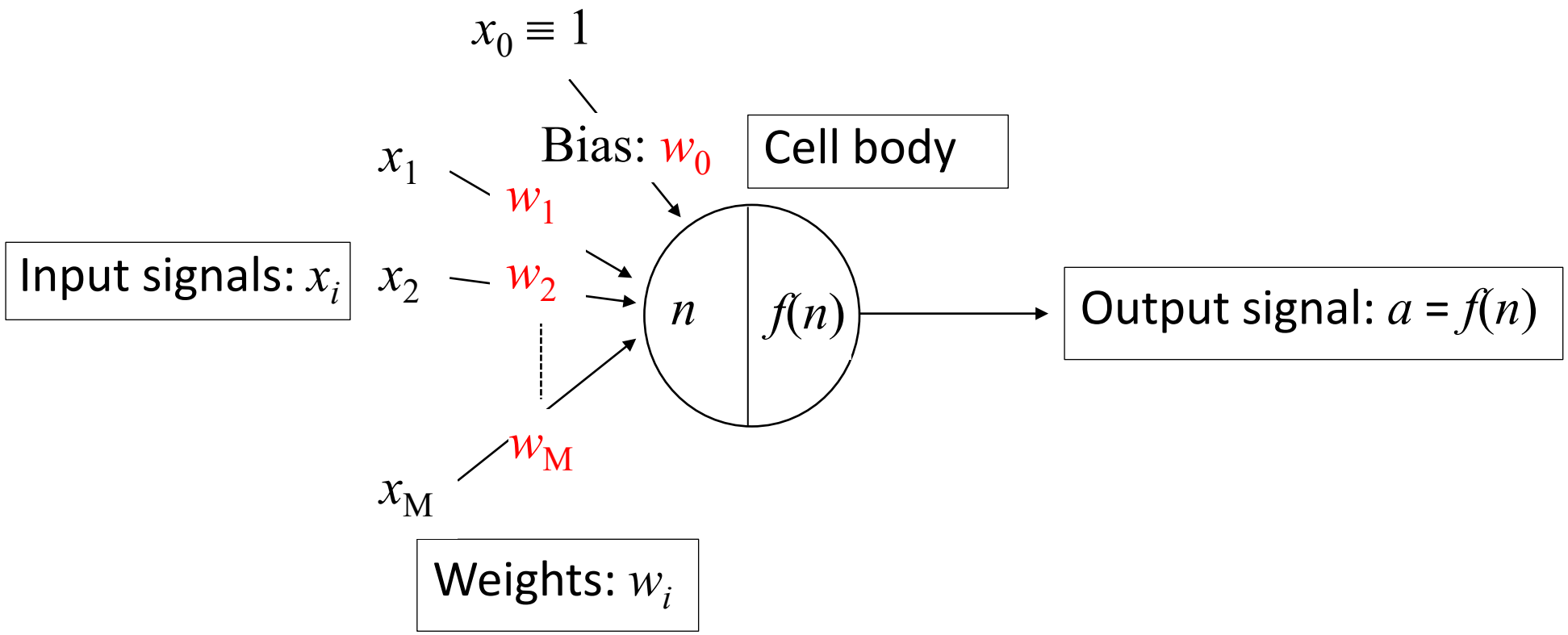
Outline

- Network Architectures
- Backpropagation Algorithm
- Remarks on Backpropagation

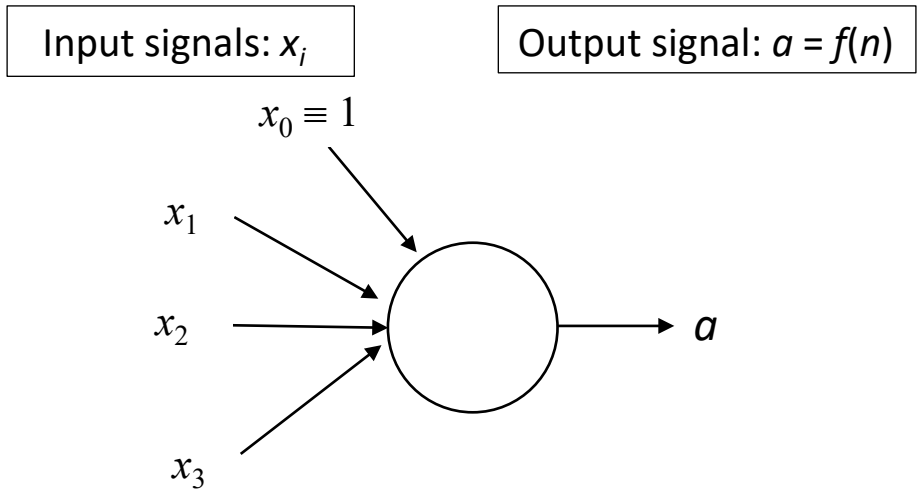
Network Architectures

1. A Single Neuron
2. A Layer of Neurons
3. Multilayer Networks

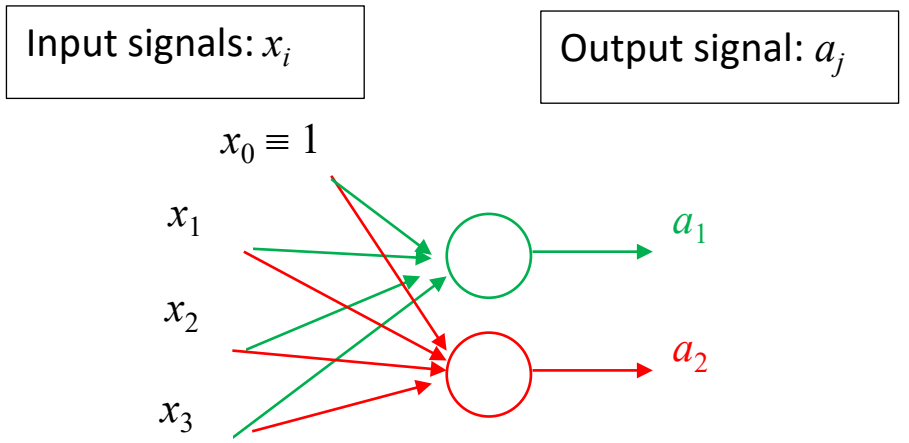
A Single Neuron



A Layer of Neurons (1/3)



A Single Neuron



A Layer of 2 Neurons

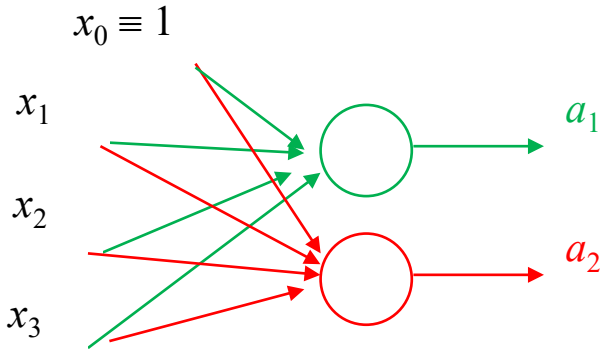
A Layer of Neurons (2/3)

同一層大多是做相同的事情
所以觸發函數是相同的

W10
1是後面的註標
0是前面的註標

Input signals: x_i

Output signal: a_j



$$\mathbf{Wx} = \begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} = \mathbf{n}$$

$$a_1 = f(n_1) = f(w_{10}x_0 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3)$$

$$a_2 = f(n_2) = f(w_{20}x_0 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3)$$

$$\mathbf{a} = f(\mathbf{n}) = \begin{bmatrix} f(n_1) \\ f(n_2) \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

A Layer of Neurons (3/3)

- Input vector **x** is a vector of length 4
  - 4: number of input elements (including bias x_0)
    雖然輸入只有3個，但有加bias
- Output vector **a** is a vector of length 2
  - 2: number of neurons in a layer
- Weight matrix **W** is a 2x4 matrix
  - $\mathbf{W} = \begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix}$ Augmented(擴充) Matrix
    表示有加bias
  - Entries in row 1 are the weights of neuron 1
  - Entries in row 2 are the weights of neuron 2

先寫後面的
再寫前面的

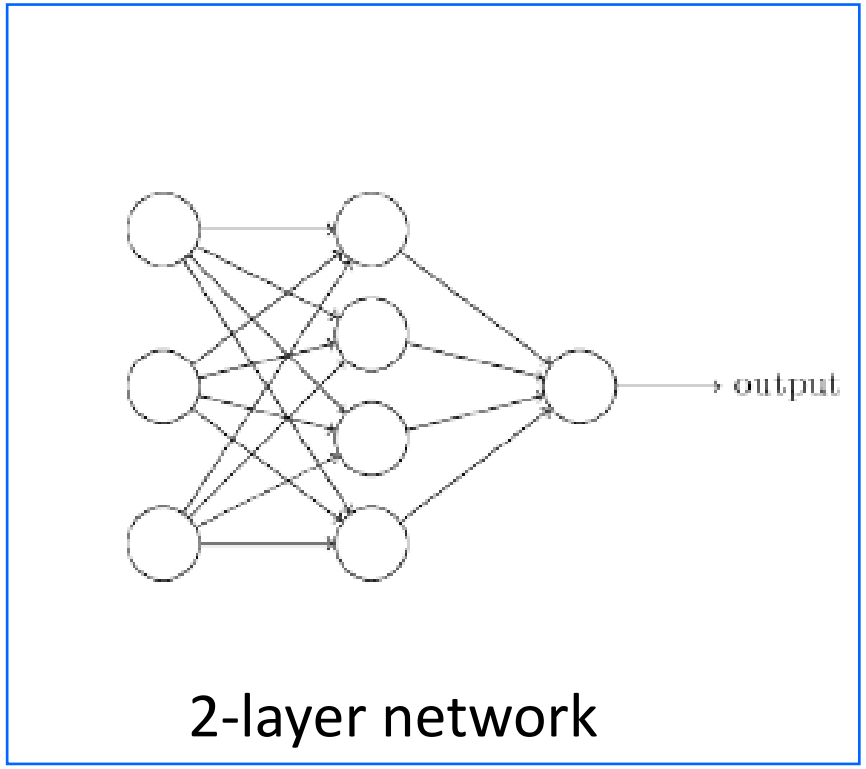
Multilayer Networks (1/4)

- A multilayer feedforward network is composed of:
  - An input layer
  - One or more hidden layers
  - An output layer
- "Layers": counted in weight layers
  - e.g., 1 hidden layer ≡ 2-layer network, 2 hidden layer ≡ 3-layer network
- Only hidden and output layers contain neurons

幾層 = 隱藏層 + 輸出層

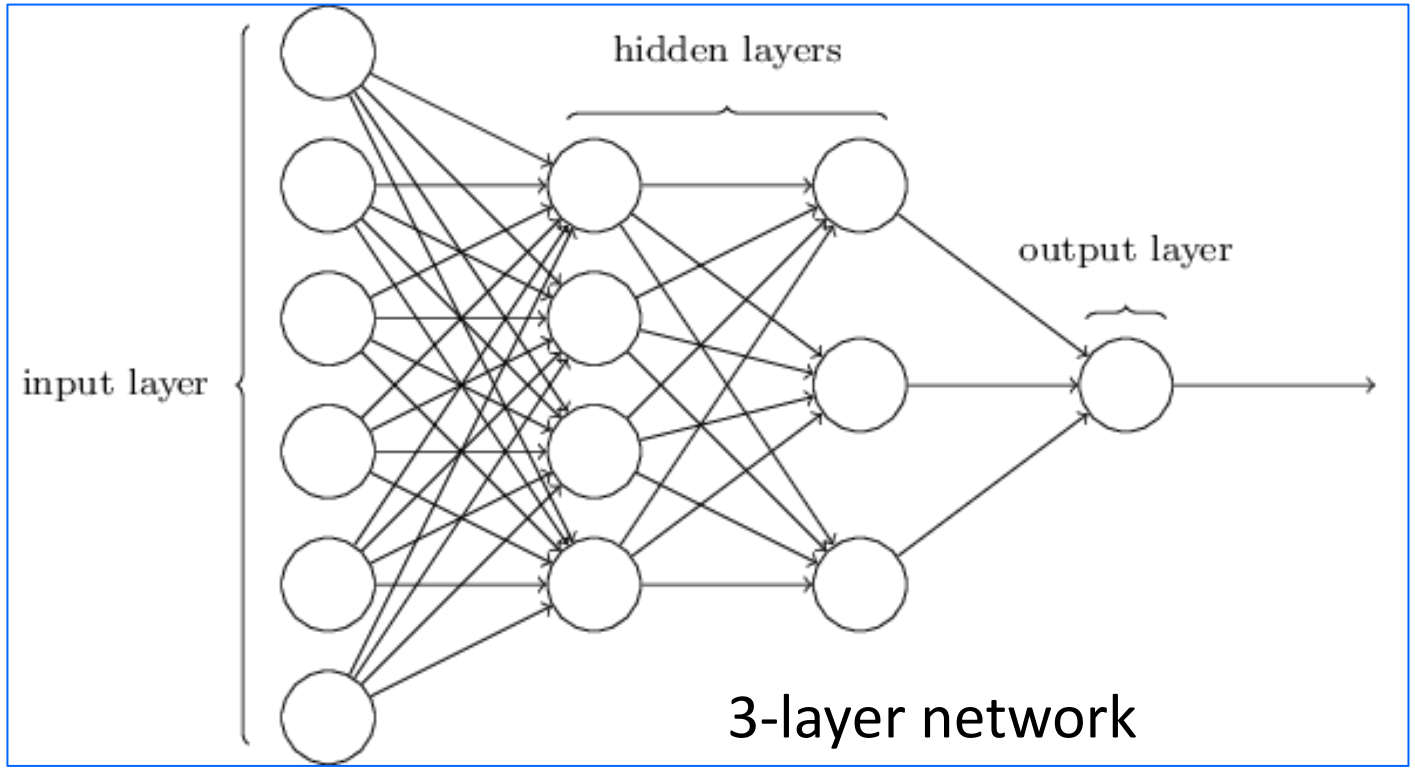
Multilayer Networks (2/4)

為了簡化，所以沒有畫bias



2-layer network

一個hidden layer + 一個output layer



3-layer network

兩個hidden layer + 一個output layer

Figures adapted from Ref. [4]

Multilayer Networks (3/4)

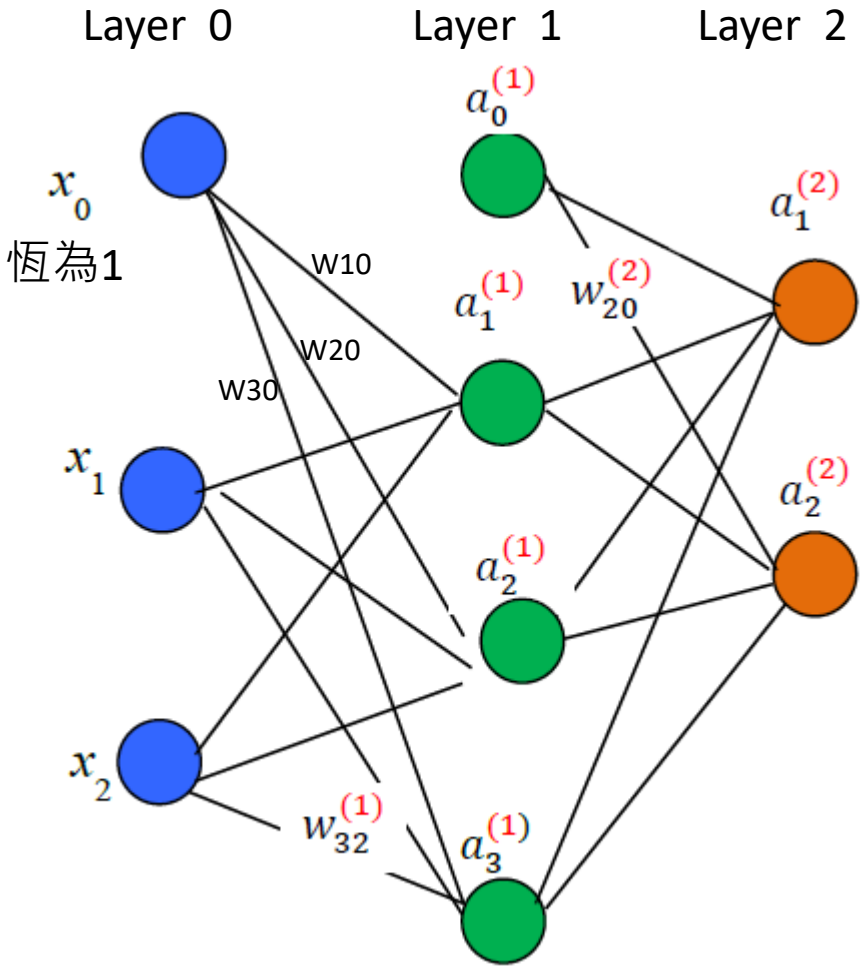
- The **input layer(輸入層)** is called layer 0(第0層)
- The **outputs** of layer k are the **inputs** of layer $(k+1)$
- Each layer has its weight matrix **W**, net-input vector **n** and output vector **a**
- Need some additional notation to distinguish between different layers
  - Use superscripts to identify the layers

Multilayer Networks (4/4)

上註標說明第幾層

- Use superscripts(上標) to identify the layers -> 上標表示第幾層
- **Define $a^{(0)}$ = x** for convenience; that is:
- 第0層表示 x

- $\mathbf{a^{(0)}} = \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \mathbf{x}$



Backpropagation Algorithm 反傳導

1. Stochastic Gradient Descent: a Single Neuron
2. Train a Two-Layer Neural Network
3. Stochastic Backpropagation: Notation
4. Backpropagation Procedure

在輸送訊息時 ，就會向前前進，把訊息送到大腦，大腦就會去做認知

Stochastic (Incremental) Gradient Descent

Do until satisfied // Epoch

For each training example

Compute the gradient $\nabla E(\mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \eta\,(-\nabla E(\mathbf{w}))$ // update **w** per training example

Batch Gradient Descent

Do until satisfied // Epoch

Initialize delta_**w** to zero

FOR each training example // Find $\nabla E(\mathbf{w})$ for each training example

Compute the gradient $\nabla E(\mathbf{w})$

delta_**w** $\leftarrow$ delta_**w** + $\eta\,(-\nabla E(\mathbf{w}))$

w $\leftarrow$ **w** + delta_**w** // update **w** per epoch

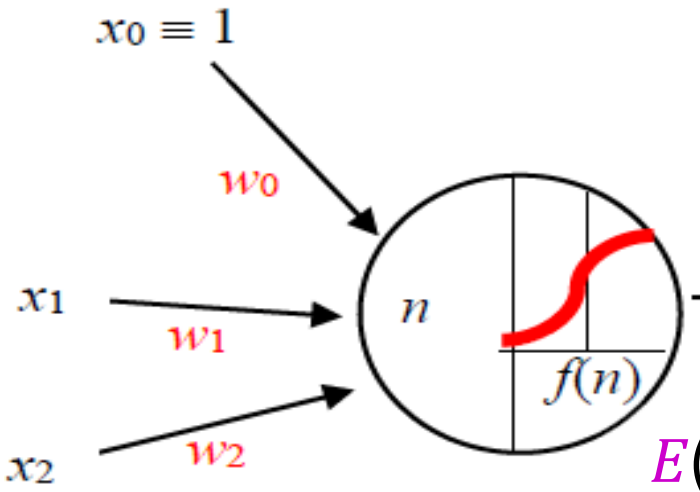
(Some other variation version of update W:

w $\leftarrow$ **w** + (1/N) delta_**w**

// N training examples; the average of delta_**w**)

Error Gradient for a Sigmoid Neuron: Half of Squared Error

$$h(\mathbf{x}) = f(n) = \sigma(w_0 x_0 + w_1 x_1 + w_2 x_2) = a = \hat{y}$$



$h(\mathbf{x}) = a = \hat{y}$? $\cong$ y

$E(w_0, w_1, w_2) = \frac{1}{2}(y - \hat{y})^2$

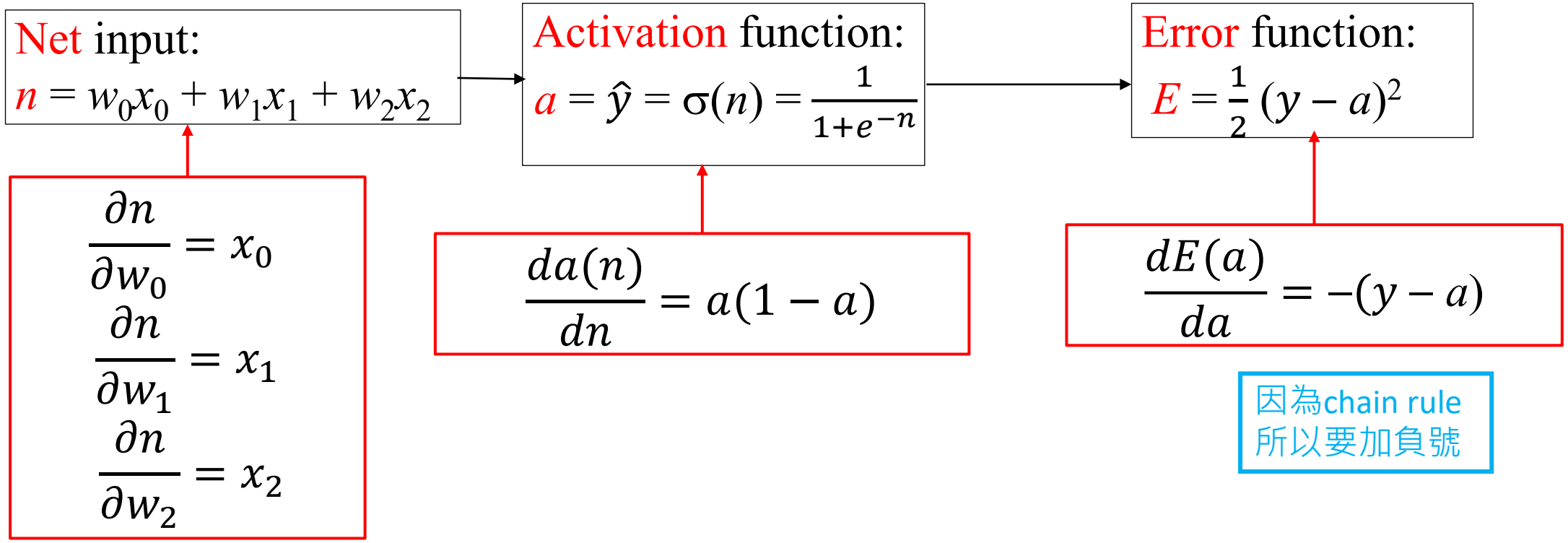
Gradient Descent

$\mathbf{w} \leftarrow \mathbf{w} + \eta(-\nabla E(\mathbf{w}))$

Where $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$

How to find $\nabla E(\mathbf{w})$? (1/2)

First, find derivatives of *n*, *a*, and *E*.



How to find $\nabla E(\mathbf{w})$? (2/2)

Next, use the chain rule to find $\frac{\partial E}{\partial w_0}$, $\frac{\partial E}{\partial w_1}$, $\frac{\partial E}{\partial w_2}$:

$$\frac{\partial E}{\partial w_0} = \frac{dE}{da}\frac{da}{dn}\frac{\partial n}{\partial w_0} = -(y-a)\,a(1-a)x_0$$

$$\frac{\partial E}{\partial w_1} = \frac{dE}{da}\frac{da}{dn}\frac{\partial n}{\partial w_1} = -(y-a)\,a(1-a)x_1$$

$$\frac{\partial E}{\partial w_2} = \frac{dE}{da}\frac{da}{dn}\frac{\partial n}{\partial w_2} = -(y-a)\,a(1-a)x_2$$

Stochastic Gradient Descent: a Single Neuron

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix} = \begin{bmatrix} -(y-a)a(1-a)x_0 \\ -(y-a)a(1-a)x_1 \\ -(y-a)a(1-a)x_2 \end{bmatrix} = -(y-a)a(1-a)\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = -(y-a)a(1-a)\mathbf{x}$$

Gradient Descent Rule

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\,(-\nabla E(\mathbf{w}))$$

$$\Rightarrow \mathbf{w} \leftarrow \mathbf{w} + \eta\,(y-a)a(1-a)\mathbf{x}$$

: Vector Form

Or component form:

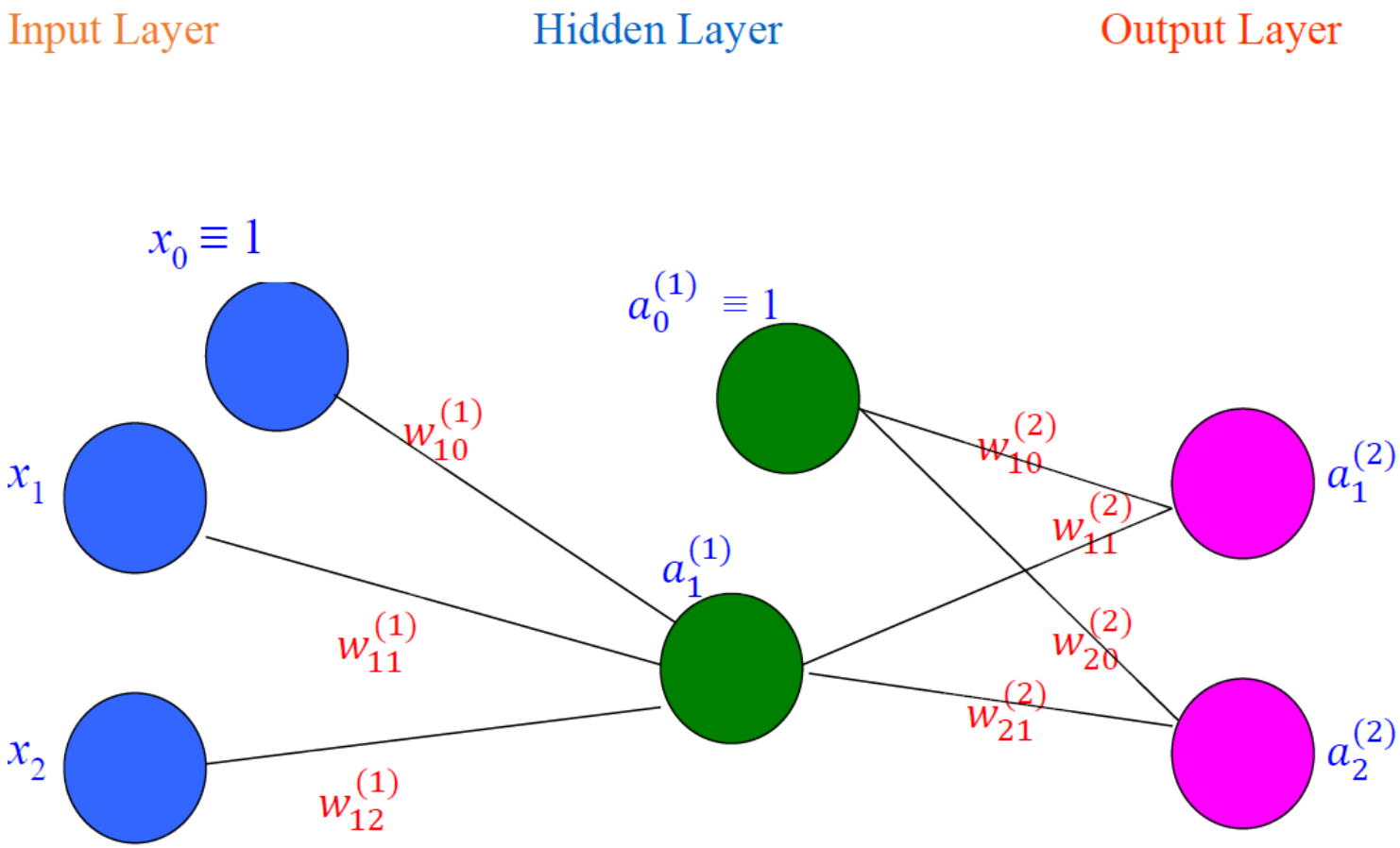
$$w_0 \leftarrow w_0 + \eta\,(y-a)a(1-a)x_0$$
$$w_1 \leftarrow w_1 + \eta\,(y-a)a(1-a)x_1$$
$$w_2 \leftarrow w_2 + \eta\,(y-a)a(1-a)x_2$$

Illustrative Example of Backpropagation Algorithm

Example:

- A two-layer neural network
- Activation Function: Sigmoid Function
- Error Function: Half of Squared Error
- Learning Algorithm: Stochastic Gradient Descent (SGD)

Two-Layer Neural Network: Example



How to Train a Two-Layer Neural Network?

- Recall that the SGD for a single neuron

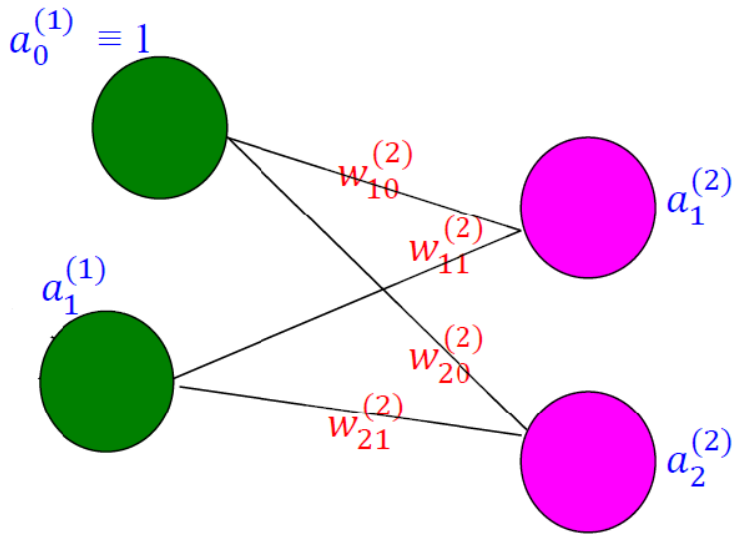
$$\mathbf{w} \leftarrow \mathbf{w} + \eta (y - a)a(1 - a)\mathbf{x}$$

- Let $\delta = (y - a)a(1 - a)$ be the error term
- Then the SGD:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \delta \mathbf{x}$$

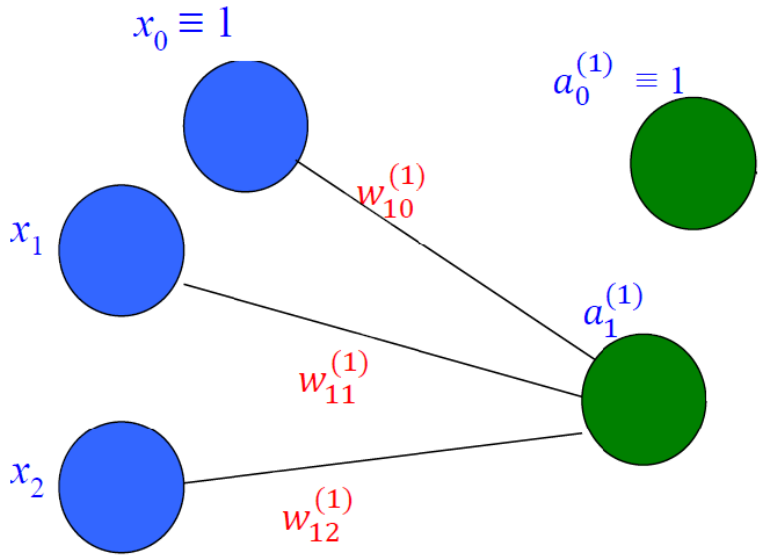
Error Term of the Output Layer

- Let $\delta_j^{(2)} = \left(y_j^{(2)} - a_j^{(2)}\right)\left[a_j^{(2)}\left(1 - a_j^{(2)}\right)\right], j = 1, 2$
- Then $w_{ji}^{(2)} \leftarrow w_{ji}^{(2)} + \eta\delta_j^{(2)} a_i^{(1)}, j = 1, 2$ and $i = 0, 1$



Error Term of the Hidden Layer (1/3)

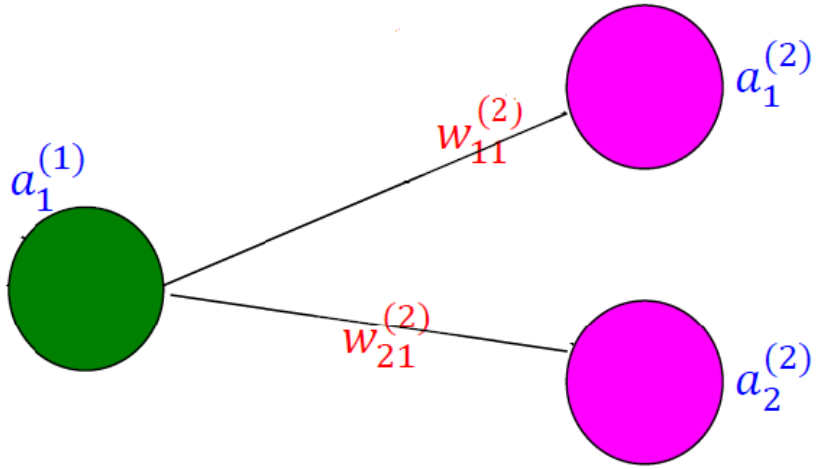
- We can not find $\delta_1^{(1)}$ using: $\delta_1^{(1)} = \left(y_1^{(1)} - a_1^{(1)}\right)\left[a_1^{(1)}\left(1 - a_1^{(1)}\right)\right]$
  - Because $y_1^{(1)}$ is unknown



Error Term of the Hidden Layer (2/3)

- The value of $\delta_1^{(1)}$ can be determined by:

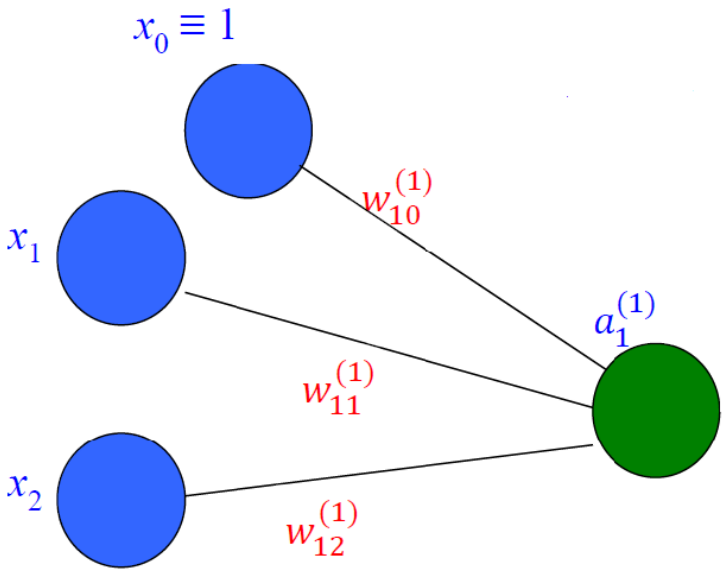
$$\delta_1^{(1)} = \boxed{\left(w_{11}^{(2)}\delta_1^{(2)} + w_{21}^{(2)}\delta_2^{(2)}\right)}\left[a_1^{(1)}\left(1 - a_1^{(1)}\right)\right]$$



Error Term of the Hidden Layer (3/3)

- **Let a$^{(0)}$** = $\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$ = **x**

- $w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} + \eta \delta_j^{(1)} a_i^{(0)}$, j = 1 and i = 0, 1, 2



Stochastic Backpropagation: Notation (1/4)

(**x**, y): training example

- $\mathbf{x} = \begin{bmatrix} x_0 \equiv 1 \\ x_1 \\ x_2 \end{bmatrix}$, where x_1, and x_2 are input attributes

- $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$.

Stochastic Backpropagation: Notation (2/4)

$\boldsymbol{a}^{(l)}$: the l^{th} layer output vector, $l = 0, 1, 2$

- $\boldsymbol{a}^{(0)} = \begin{bmatrix} a_0^{(0)} \equiv 1 \\ a_1^{(0)} \\ a_2^{(0)} \end{bmatrix} = \begin{bmatrix} x_0 \equiv 1 \\ x_1 \\ x_2 \end{bmatrix} = \mathbf{x}$, ← 最一開始的輸入
- $\boldsymbol{a}^{(1)} = \begin{bmatrix} a_0^{(1)} \equiv 1 \\ a_1^{(1)} \end{bmatrix}$, and $\boldsymbol{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix}$.

Stochastic Backpropagation: Notation (3/4)

$\delta_j^{(l)}$: the error term for the j^{th} neuron in the l^{th} layer

- Output layer: $\delta_j^{(2)} = \left(y_j^{(2)} - a_j^{(2)}\right)\left[a_j^{(2)}\left(1 - a_j^{(2)}\right)\right], j = 1, 2$
- Hidden layer: $\delta_1^{(1)} = (w_{11}^{(2)}\delta_1^{(2)} + w_{21}^{(2)}\delta_2^{(2)})\left[a_1^{(1)}\left(1 - a_1^{(1)}\right)\right]$

Stochastic Backpropagation: Notation (4/4)

$w_{ji}^{(l)}$: the weight between unit *i* in layer *l*-1 and unit *j* in layer *l*

- $w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} + \eta\delta_j^{(1)} a_i^{(0)}$ $i = 0, 1, 2; j = 1$
- $w_{ji}^{(2)} \leftarrow w_{ji}^{(2)} + \eta\delta_j^{(2)} a_i^{(1)}, j = 1, 2$ and $i = 0, 1$

Stochastic Backpropagation (1/3)

// Build a two-layer neural network

// Use sigmoid neurons in the hidden layer and the output layer

// For each neuron in the output layer,

// use Half of Squared Error as the loss function

Stochastic Backpropagation (2/3)

Initialize all network weights to small random numbers
UNTIL one of the termination conditions is met, DO
  FOR each (**x**, **y**) in the training dataset, DO
    Step 1. Feedforward:
      // Compute the output for each neuron in the network
      Input the instance **x** **(=** $\boldsymbol{a}^{(0)}$**)**
      For each $l = 1, 2$
      Compute $\boldsymbol{a}^{(l)}$

Stochastic Backpropagation (3/3)

Step 2. Backward:

Step 2.1 Calculate the output layer error layer,

$$\delta_j^{(2)} = \left(y_j^{(2)} - a_j^{(2)}\right)\left[a_j^{(2)}\left(1 - a_j^{(2)}\right)\right], j = 1, 2$$

Step 2.2 Backpropagate the error for the hidden layer, and compute

$$\delta_1^{(1)} = \left(w_{11}^{(2)}\delta_1^{(2)}\, w_{21}^{(2)}\delta_2^{(2)}\right)\left[a_1^{(1)}\left(1 - a_1^{(1)}\right)\right]$$

Step 2.3 Update all of weights

For each $l = 1, 2$

Compute $w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} + \delta_j^{(l)} a_i^{(l-1)}$

Remarks on Backpropagation

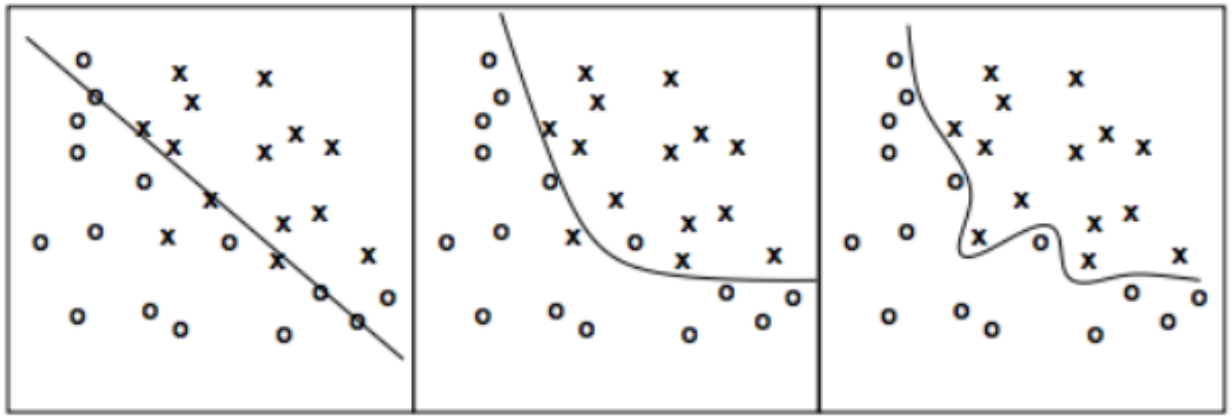
1. Convergence of Backpropagation
2. Overfitting
3. Error functions
4. Hidden Layer Representations

Convergence of Backpropagation

- Note that gradient descent may converge to some local minimum.
  - Adding momentum may help to avoid some local minima.
  - Stochastic gradient descent or mini-batch gradient descent may also help to avoid some local minima.
  - Train multiple networks with different initial weights and learning rates

Overfitting

- Definition of Overfitting
  - *h'* worse than *h* on D_{train}, better on D_{test}
  - Learned hypothesis *h* may fit the training data very well, even outliers (noise) but fail to generalize to new examples (test data)



Error functions

- For real-valued regression, we might use the squared loss:
  - $(t - a)^2$
- For binary classification using a sigmoid function, we use the binary cross-entropy loss function:
  - $-(t \log a + (1 - t) \log(1 - a))$
- For softmax over *K* classes, we use the cross-entropy loss function:
  - $-\sum_{j=1}^{K} t_j \log a_j$

Hidden Layer Representation

- Hidden Units and Feature Extraction
  - Training procedure: hidden unit representations that minimize error *E*
  - Sometimes backpropagation will define new hidden features that are not explicit in the input representation, but which capture properties of the input instances that are most relevant to learning the target function
  - Hidden units express *newly constructed features*

References

[1] Neural Network Design (2nd Edition), Martin T Hagan , Howard B Demuth , Mark H Beale , and Orlando De Jesús, Martin Hagan, 2014.

[2] Pattern Recognition and Machine Learning, Christopher Bishop, Springer-Verlag New York, 2006.

[3] Machine Learning, Tom M. Mitchell, McGraw-Hill, 1997.

[4] Neural Networks and Deep Learning, Michael Nielson, https://neuralnetworksanddeeplearning.com