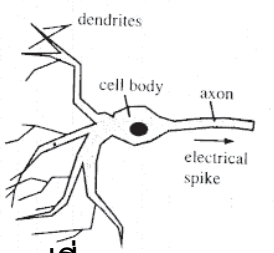
5.9 Artificial Neural Networks

- Artificial neural networks (ANN) เป็นการจำลองการทำงานบางส่วนของสมองมนุษย์
- Neuron เป็นเซลที่ประกอบด้วย nucleus, cell body, dendry, synapse axon ดังแสดงในรูปที่ 5.9.1



รูปที่ 5.9.1 neuron

- สมองของมนุษย์มี neuron ประมาณ 10^{11} แต่ละ neuron เชื่อมต่อกับ neuron อื่นประมาณ 10^{4} และมี stwiching time ประมาณ 10^{-3} วินาที ซึ่งช้ามากเมื่อเทียบกับคอมพิวเตอร์ (10^{-10} วินาที) แต่ทำงานบางอย่างได้ดีกว่ามาก

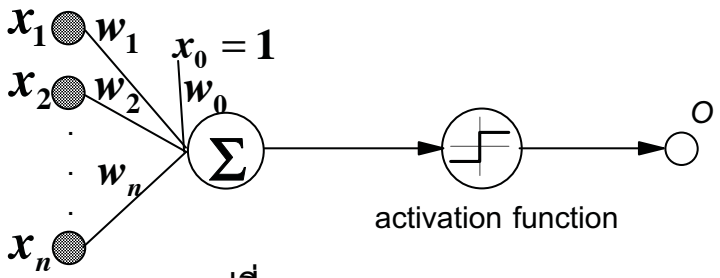
Perceptrons

- Perceptron เป็น ANN ชนิดหนึ่ง ซึ่งประกอบด้วยยูนิตเดี่ยว(รูป 5.9.2)
- input เป็น real-valued vector
- Perceptron คำนวณ linear combination ของ input(x) และให้ output(o) เป็น 1 ถ้าค่าที่ได้เกิน threshold และเป็น -1 ถ้าค่าไม่เกิน

$$o(x_1,...,x_n)=\begin{cases}1 & if\ \ w_0+w_1x_1+\cdots+w_nx_n>0\\ -1 & if\ \ w_0+w_1x_1+\cdots+w_nx_n<0\end{cases}$$

โดยที่ -w0 คือ ค่า threshold, wi เป็น real-valued constant หรือ weight

- wi เป็นตัวกำหนดความสำคัญของ input xi ที่มีต่อ output



รูปที่ 5.9.2 Perceptron

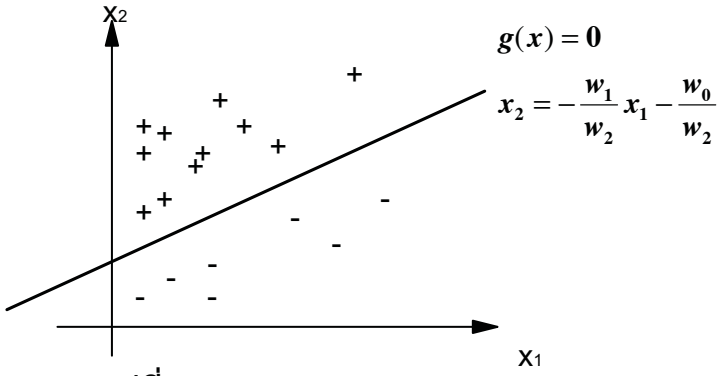
- ให้ $g(x)=\sum_{i=0}^{n}w_ix_i$ หรือในรูปของเวกเตอร์ $g(x)=\vec{w}\cdot\vec{x}$

$$o(x_1,...,x_n)=\begin{cases}1 & if\ \ g(x)>0\\ -1 & if\ \ g(x)<0\end{cases}$$

o ใช้ activation function เป็น stepwise bipolar (output = 1 หรือ -1)

- ในกรณีของ 2 inputs ดังแสดงในรูป 5.9.3

$$g(x)=w_0+w_1x_1+w_2x_2$$



รูปที่ 5.9.3 decision surface

- เราสามารถมอง perceptron เป็น hyperplane decision surface (g(x)=0)ใน n-dimensional surface
- เราสามารถสอน perceptron ให้แยกตัวอย่าง (คู่ลำดับ input-output)
- คุณสมบัติหนึ่งของ perceptronคือ ทุก function ที่ perceptron สามารถคำนวณได้ มันก็สามารถที่จะถูกสอนให้เรียน function นั้นได้

Perceptron Learning Rule

- จงหาค่าของเวกเตอร์น้ำหนัก $\vec{w}$ ที่ทำให้ perceptron เอ๊าท์พุตเป็น +1 หรือ -1 ได้ถูกต้องสำหรับทุกตัวอย่างที่สอน
- Perceptron learning rule
  - เริ่มต้นจากการสุ่มค่าของน้ำหนัก w_i
  - เทียบ perceptron กับทุกตัวอย่างที่สอนทีละตัว และแก้ไขน้ำหนักเมื่อ perceptron แยกตัวอย่างผิดพลาด
  - วนทำซ้ำกับตัวอย่างที่สอน จนกระทั่ง perceptron แยกตัวอย่างได้ถูกต้องทั้งหมด
  - น้ำหนักถูกปรับตาม $w_i = w_i + \Delta w_i$

    โดยที่ $\Delta w_i = \alpha\ (t-o)x_i$ = α × error × input

    t เป็น target output,

    o เป็น output จาก perceptron, α เป็นค่าคงที่แสดง learning rate

-ในกรณีที่ perceptron แยกตัวอย่างได้ถูกต้อง

(t-o) จะมีค่าเป็น 0 Δw ไม่เปลี่ยนแปลง

-ในกรณีที่ perceptron ให้เอ๊าท์พุตเป็น -1 แต่ target output = 1

เพื่อที่จะทำให้ perceptron เอ๊าท์พุตเป็น 1 น้ำหนักต้องถูกปรับให้สามารถเพิ่มค่าของ $\vec{w} \cdot \vec{x}$

-- ถ้า x_i>0, w_i จะเพิ่มขึ้นและจะทำให้ perceptron เอ๊าท์พุตได้ถูกต้องยิ่งขึ้น ($\Delta w_i = \alpha(t-o)x_i > 0$)

-- ถ้า x_i<0, w_i จะลดลงและจะทำให้ perceptron เอ๊าท์พุตได้ถูกต้องยิ่งขึ้น

-ในกรณีที่ perceptron ให้เอ๊าท์พุตเป็น 1 แต่ target output = -1

w_i ของ x_i ที่เป็นค่าบวก จะลดลง,

w_i ของ x_i ที่เป็นค่าลบ จะเพิ่มขึ้น

ตารางที่ 5.9.1 การเรียนรู้ของ Perceptron กับฟังก์ชัน AND

(binary activation function (output = 0 หรือ 1))

Perceptron Learning Example

Bias Input X0 = ±1 Alpha = 0.05

Input X1	Input X2	1.0*W0	X1*W1	X2*W2	Net Sum Input	Target Out	Actual Out	Alpha *Error	W0	W1	W2
									0.1	0.1	0.1
0	0	0.10	0.00	0.00	0.10	0	1	−0.05	0.05	0.10	0.10
0	1	0.05	0.00	0.10	0.15	0	1	−0.50	−0.45	0.10	−0.40
1	0	−0.45	0.10	0.00	−0.35	0	0	0.00	−0.45	0.10	−0.40
1	1	−0.45	0.10	−0.40	−0.75	1	0	0.50	0.05	0.60	0.10
0	0	0.05	0.00	0.00	0.05	0	1	−0.50	−0.45	0.60	0.10
0	1	−0.45	0.00	0.10	−0.35	0	0	0.00	−0.45	0.60	0.10
1	0	−0.45	0.60	0.00	0.15	0	1	−0.50	−0.95	0.10	0.10
1	1	−0.95	0.10	0.10	−0.75	1	0	0.50	−0.45	0.60	0.60
0	0	−0.45	0.00	0.00	−0.45	0	0	0.00	−0.45	0.60	0.60
0	1	−0.45	0.00	0.60	0.15	0	1	−0.50	−0.95	0.60	0.10
1	0	−0.95	0.60	0.00	−0.35	0	0	0.00	−0.95	0.60	0.10
1	1	−0.95	0.60	0.10	−0.25	1	0	0.50	−0.45	1.10	0.60
0	0	−0.45	0.00	0.00	−0.45	0	0	0.00	−0.45	1.10	0.60
0	1	−0.45	0.00	0.60	0.15	0	1	−0.50	−0.95	1.10	0.10
1	0	−0.95	1.10	0.00	0.15	0	1	−0.50	−1.45	0.60	0.10
1	1	−1.45	0.60	0.10	−0.75	1	0	0.50	−0.95	1.10	0.60
0	0	−0.95	0.00	0.00	−0.95	0	0	0.00	−0.95	1.10	0.60
0	1	−0.95	0.00	0.60	−0.35	0	0	0.00	−0.95	1.10	0.60
1	0	−0.95	1.10	0.00	0.15	0	1	−0.50	−1.45	0.60	0.60
1	1	−1.45	0.60	0.60	−0.25	1	0	0.50	−0.95	1.10	1.10
0	0	−0.95	0.00	0.00	−0.95	0	0	0.00	−0.95	1.10	1.10
0	1	−0.95	0.00	1.10	0.15	0	1	−0.50	−1.45	1.10	0.60
1	0	−1.45	1.10	0.00	−0.35	0	0	0.00	−1.45	1.10	0.60
1	1	−1.45	1.10	0.60	0.25	1	1	0.00	−1.45	1.10	0.60
0	0	−1.45	0.00	0.00	−1.45	0	0	0.00	−1.45	1.10	0.60
0	1	−1.45	0.00	0.60	−0.85	0	0	0.00	−1.45	1.10	0.60
1	0	−1.45	1.10	0.00	−0.35	0	0	0.00	−1.45	1.10	0.60
1	1	−1.45	1.10	0.60	0.25	1	1	0.00	−1.45	1.10	0.60
0	0	−1.45	0.00	0.00	−1.45	0	0	0.00	−1.45	1.10	0.60
0	1	−1.45	0.00	0.60	−0.84	0	0	0.00	−1.45	1.10	0.60
1	0	−1.45	1.10	0.00	−0.35	0	0	0.00	−1.45	1.10	0.60
1	1	.45	1.10	0.60	0.25	1	1	0.00	−1.45	1.10	0.60

ตารางที่ 5.9.2 ความพยายามของ perceptron ที่จะเรียนฟังก์ชัน XOR

(binary activation function (output = 0 หรือ 1))

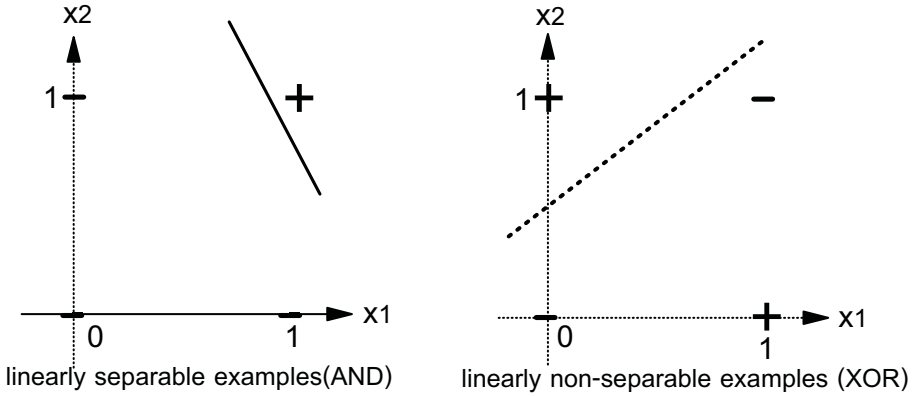
Perceptron Learning Example

Bias Input X0 = ±1 Alpha = 0.80

Input X1	Input X2	1.0*W0	X1*W1	X2*W2	Net Sum Input	Target Out	Actual Out	Alpha *Error	W0	W1	W2
									0.1	0.1	0.1
0	0	0.10	0.00	0.00	0.10	0	1	-0.20	-0.10	0.10	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	0.10	0.60
1	0	0.40	0.10	0.00	0.50	1	1	0.00	0.40	0.10	0.60
1	1	0.40	0.10	0.60	1.10	0	1	-0.50	-0.10	-0.40	0.10
0	0	-0.10	0.00	0.00	-0.10	0	0	0.00	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10
0	0	0.40	0.00	0.00	0.40	0	1	-0.50	-0.10	-0.40	0.10
0	1	-0.10	0.00	0.10	0.00	1	0	0.50	0.40	-0.40	0.60
1	0	0.40	-0.40	0.00	0.00	1	0	0.50	0.90	0.10	0.60
1	1	0.90	0.10	0.60	1.60	0	1	-0.50	0.40	-0.40	0.10

ข้อจำกัดของ Perceptron Learning Rule

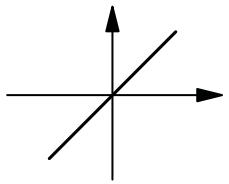
- concept หรือ function ที่สามารถเรียนได้โดย perceptron learning rule นั้น จะต้องเป็น linearly separable function
- ถ้าไม่เป็นแบบ linearly separable, perceptron จะไม่ลู่เข้า (รูปที่ 5.9.4)



รูปที่ 5.9.4 linearly and linearly non-separable function

Delta Rule (Gradient Descent)

- delta rule คล้ายกับ perceptron learning rule แต่จะลู่เข้าสู่ค่าที่ทำให้ error น้อยที่สุด ในกรณีที่ตัวอย่างไม่เป็นแบบ linearly separable
- ใช้หลักการของ gradient descent เพื่อหาคำตอบจาก space ของเวกเตอร์น้ำหนักที่เป็นไปได้
- เป็นพื้นฐานของอัลกอริทึม Backprogagation
- ใช้ activation function เป็น linear function ซึ่งหาอนุพันธ์ได้ (รูปที่ 5.9.5)

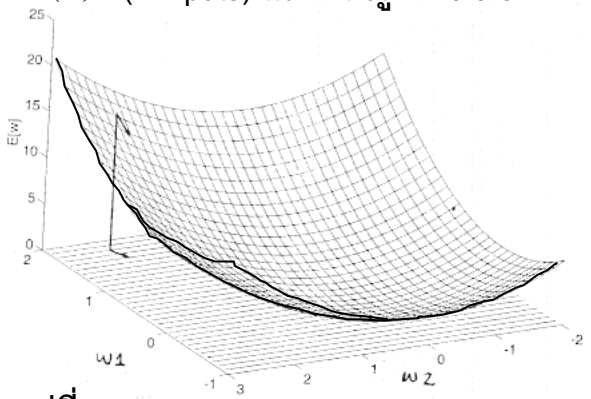


รูปที่ 5.9.5 linear activation function

- เอ๊าท์พุตของ perceptron แสดงโดย $o(\vec{x}) = \vec{w} \cdot \vec{x}$

- นิยาม training error (E(w)) $E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$

  โดยที่ D เป็นเซตของตัวอย่าง, td เป็น target output ของตัวอย่าง d od เป็น output ของ perceptron สำหรับตัวอย่าง d
- $E(\vec{w})$ เป็น parabolic function ของ $\vec{w}$ ซึ่งมีค่าต่ำสุดค่าเดียว
- ตัวอย่างของ $E(\vec{w})$ (2 inputs) แสดงในรูปที่ 5.9.6



รูปที่ 5.9.6 Hypothesis Space of w and E(w)

Derivation of the Delta Rule

- หลักการของ delta rule คือหาค่า $\vec{w}$ ที่ทำให้ $E(\vec{w})$ มีค่าน้อยที่สุด
- เริ่มจากเวกเตอร์น้ำหนักเริ่มต้นแล้วปรับค่าเวกเตอร์น้ำหนักทีละน้อย ในทิศทางลงที่ชันที่สุด(steepest descent)ของerror surface(รูป5.9.6)
- เวกเตอร์ที่สัมผัสกับ error surface คำนวณได้จากอนุพันธ์ของ $E(\vec{w})$ เทียบกับ $\vec{w}$ (ให้เวกเตอร์นี้แทนด้วย $\nabla E(\vec{w})$)

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

  เวกเตอร์นี้แสดงทิศในแนวขึ้น, เวกเตอร์ที่มีทิศทางลงจึงเป็น $-\nabla E(\vec{w})$
- ดังนั้นกฎในการปรับค่าเวกเตอร์น้ำหนักเป็น:

$$\vec{w} = \vec{w} + \Delta\vec{w}$$

โดยที่ $$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

η : learning rate เป็นค่าคงที่เลขบวก

• Delta rule สามารถเขียนให้อยู่ในรูปของสมาชิกแต่ละตัวได้

$$w_i = w_i + \Delta w_i$$

โดยที่ $$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

• $\frac{\partial E}{\partial w_i}$ คำนวณได้โดย:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$$

xid คือสมาชิก xi ของตัวอย่าง d

$$\therefore \Delta w_i = \eta \sum_{d\in D}(t_d - o_d)\, x_{id}$$

Delta Rule Algorithm

Delta-Rule(*training-examples*, η)

Each training example is a pair $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and *t* is the target output value. η is the learing rate.

- Initialize each *wi* to some small random value
- Until the termination condition is met, Do
  - Initialize each Δwi to zero.
  - For each $\langle \vec{x}, t \rangle$ in training-examples, Do
    - -- Input the instance $\vec{x}$ to the unit and compute the output *o*
    - -- For each linear unit weight *wi*, Do

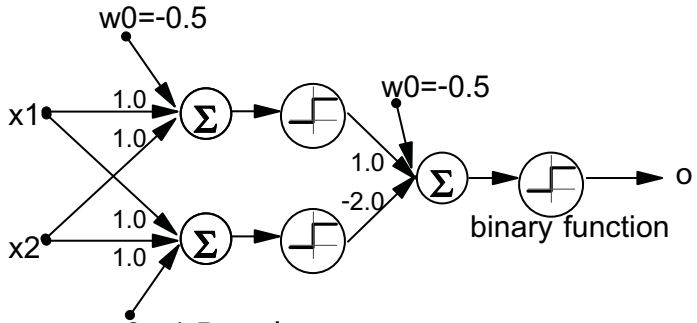
$$\Delta w_i = \Delta w_i + \eta\,(t - o)\,x_i$$

  - For each linear weight wi, Do

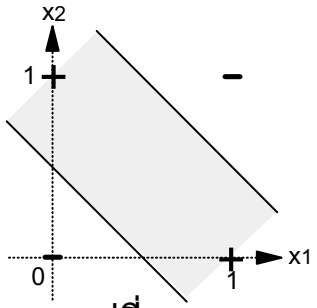
$$w_i = w_i + \Delta w_i$$

Multilayer Network and Backpropagation

- perceptron เดี่ยวสามารถแสดงได้แค่ linear decision surface เท่านั้น
- เน็ตเวิร์กหลายชั้น (multilayer network) สามารถแสดง nonlinear decision surface ได้
- ตัวอย่างของmultilayer networkที่แสดงฟังก์ชัน XORแสดงในรูป 5.9.7 และ decision surface แสดงในรูป 5.9.8



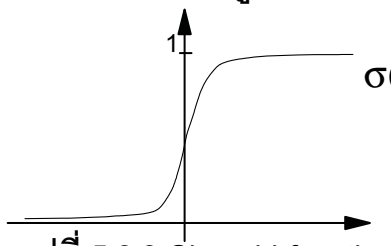
รูปที่ 5.9.7 multilayer network solving XOR



x1	x2	x1 XOR x2
0	0	0
0	1	1
1	0	1
1	1	0

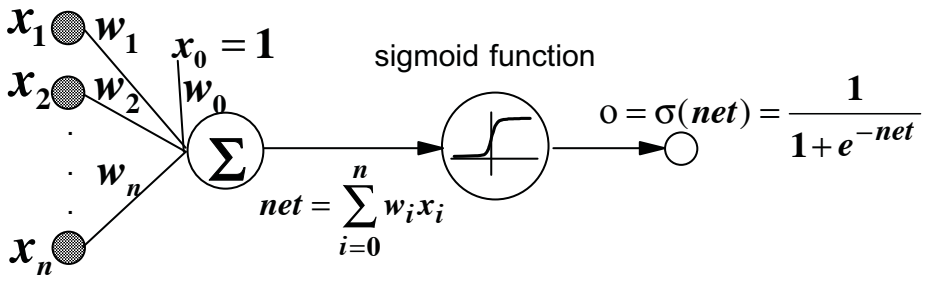
รูปที่ 5.9.8 decision surface ของ network ในรูปที่ 5.9.7

• Multilayer network ใช้ activation function ที่สามารถหาค่าอนุพันธ์ได้ เช่น sigmoid function (แสดงในรูปที่ 5.9.9)



$$\sigma(y) = \frac{1}{1+e^{-y}}$$

รูปที่ 5.9.9 Sigmoid function



รูปที่ 5.9.10 Perceptron ที่ใช้ Sigmoid function

- คุณสมบัติหนึ่งของ sigmoid function คือ ค่าอนุพันธ์แสดงอยู่ในรูปของเอ๊าท์พุตได้อย่างง่าย

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

Backpropagation Algorithm

- Backpropagation (BP) algorithm เรียนรู้ค่าเวกเตอร์น้ำหนักสำหรับ multilayer feedforward network (MLFF) โดยการใช้ gradient descent เพื่อหาค่าต่ำสุดของerrorระหว่างเอ๊าท์พุตของเน็ตเวอร์กับtarget value
- ค่า error (E) นิยามเป็น $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$

  โดยที่ outputs คือเซตของ output units ในเน็ตเวิร์ก,

  t_{kd} และ o_{kd} เป็น target value และ output value ตามลำดับของ output unit ที่ k และตัวอย่างที่ d
- BP ค้นหาเวกเตอร์น้ำหนักที่ให้ค่า error ต่ำสุด แต่ในกรณีของ MLFF ค่าต่ำสุดมีมากกว่าหนึ่ง ดังนั้นคำตอบของ BP จึงเป็น local minimum

Backpropagation(training-examples,η,n_{in},n_{out},n_{hidden})

Each training example is a pair $\langle \vec{x}, \vec{t} \rangle$,where $\vec{x}$ is the input vector, $\vec{t}$ is the target output vector, η is the learning rate. n_{in},n_{out}, n_{hidden} are number of network inputs, units in the hidden layer, output units, respectively. The input from unit i into unit j, and the weight from unit i to unit j are denoted x_{ji} and w_{ji}

- Initialize all network weights to small random numbers (e.g., [-0.05..0.05])
- Until the termination condition is met, Do
- For each $\langle \vec{x}, \vec{t} \rangle$ in training-examples, Do

  {Propagate the input forward through the network}

  1. Input the instance $\vec{x}$ to the network, compute the output o_u of every unit u.

  {Propagate the errors backward through the network}

  2. For each network output unit k, calculate its error term δ_k

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h, calculate its error term δ_h

$$\delta_h = o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

  4. Update each network weight w_{ji}: $w_{ji} = w_{ji} + \Delta w_{ji}$

  where $\Delta w_{ji} = \eta \delta_j x_{ji}$

Derivation of Backpropagation Rule

- ให้errorของแต่ละตัวอย่าง d เป็น E_d แล้วน้ำหนัก w_{ji} ถูกปรับโดย Δw_{ji} ตาม gradient descent

$$\Delta w_{ji} = -h \frac{\partial E_d}{\partial w_{ji}}$$

  โดยที่ E_d เป็น error ของตัวอย่าง d คำนวณรวมทั้งหมดสำหรับทุก output unit

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

  outputs เป็นเซตของ output units, t_k และ o_k เป็น target value และ output value ของ unit k สำหรับตัวอย่าง d ตามลำดับ
- ให้ $net_j = \sum_i w_{ji} x_{ji}$ เราจะได้ว่า $\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$

• พิจารณา $\frac{\partial E_d}{\partial net_j}$ จากสมการที่แล้วเป็น 2 กรณี:
1 กรณีที่ unit j เป็น output unit และ 2 กรณีที่ unit j เป็น hidden unit

• กรณีที่ 1: training rule สำหรับน้ำหนักของ output unit
เนื่องจาก wji มีผลต่อเน็ตเวิร์กโดยผ่านทาง *netj*
และ *netj* มีผลต่อเน็ตเวิร์กโดยผ่านทาง *oj* เท่านั้น

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

– คำนวณ $\frac{\partial E_d}{\partial o_j}$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

เพราะว่า อนุพันธ์ของพจน์อื่น ๆที่ k ที่ไม่เท่ากับ j เป็น 0

$$\frac{\partial E_d}{\partial o_j} = -(t_j - o_j)$$

– คำนวณ $\frac{\partial o_j}{\partial net_j}$,
จากคุณสมบัติของ sigmoid function

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

$$= o_j(1-o_j)$$

∴ จะได้ว่า $\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1-o_j)$

• ดังนั้น gradient descent rule สำหรับ output unit จึงเป็น

$$\Delta w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}}$$

$$= \eta(t_j - o_j)o_j(1-o_j)x_{ji}$$

• กรณีที่ 2: training rule สำหรับน้ำหนักของ hidden unit
ในการคำนวณกฎสำหรับ wji ซึ่ง j เป็น hidden unit นั้น เราต้องพิจารณาผลทางอ้อมของ wji ที่มีต่อเอ๊าท์พุตของเน็ตเวิร์กและ Ed
– ให้ Downstream(j) เป็น เซตของ units ทั้งหมดที่ได้รับ input จาก output ของ unit j
– netj สามารถส่งผลต่อเอ๊าท์พุตของเน็ตเวิร์ก (และ Ed) ได้โดยผ่านทาง unit ใน Downstream(j) เท่านั้น

$$\frac{\partial E_d}{\partial net_j} = \sum_{k\in Downstream(j)}\frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k\in Downstream(j)} -\delta_k\frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k\in Downstream(j)} -\delta_k\frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k\in Downstream(j)} -\delta_k w_{kj}\frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k\in Downstream(j)} -\delta_k w_{kj}o_j(1-o_j)$$

– ให้ δ_j แสดง $-\frac{\partial E_d}{\partial net_j}$

$$\delta_j = o_j(1-o_j)\sum_{k\in Downstream(j)}\delta_k w_{kj}$$

และ $\Delta w_{ji} = \eta\delta_j x_{ji}$

• ในอัลกอริทึมเป็นกรณีที่ Downstream(j) = outputs

An Example of Multilayer Networks

- ตัวอย่างของ multilayer networks ที่ใช้รู้จำเสียงพูด