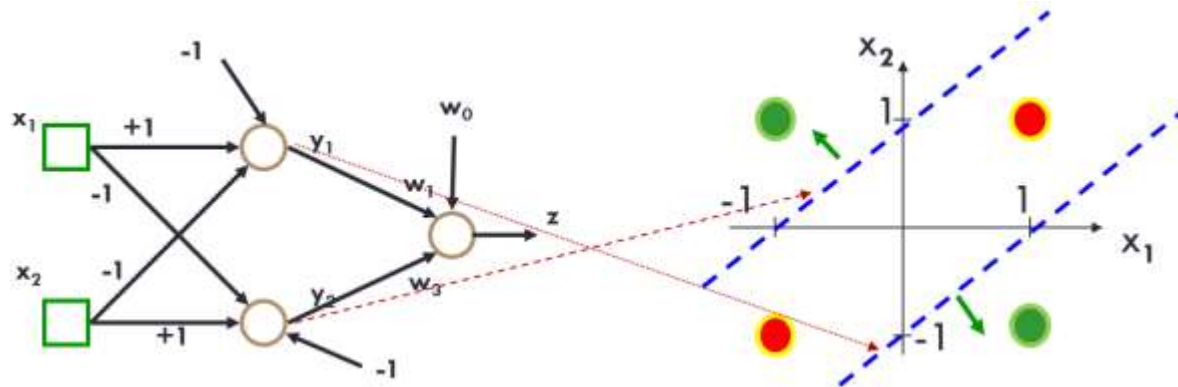


Neural Networks

Multi Layer Perceptron



Outline

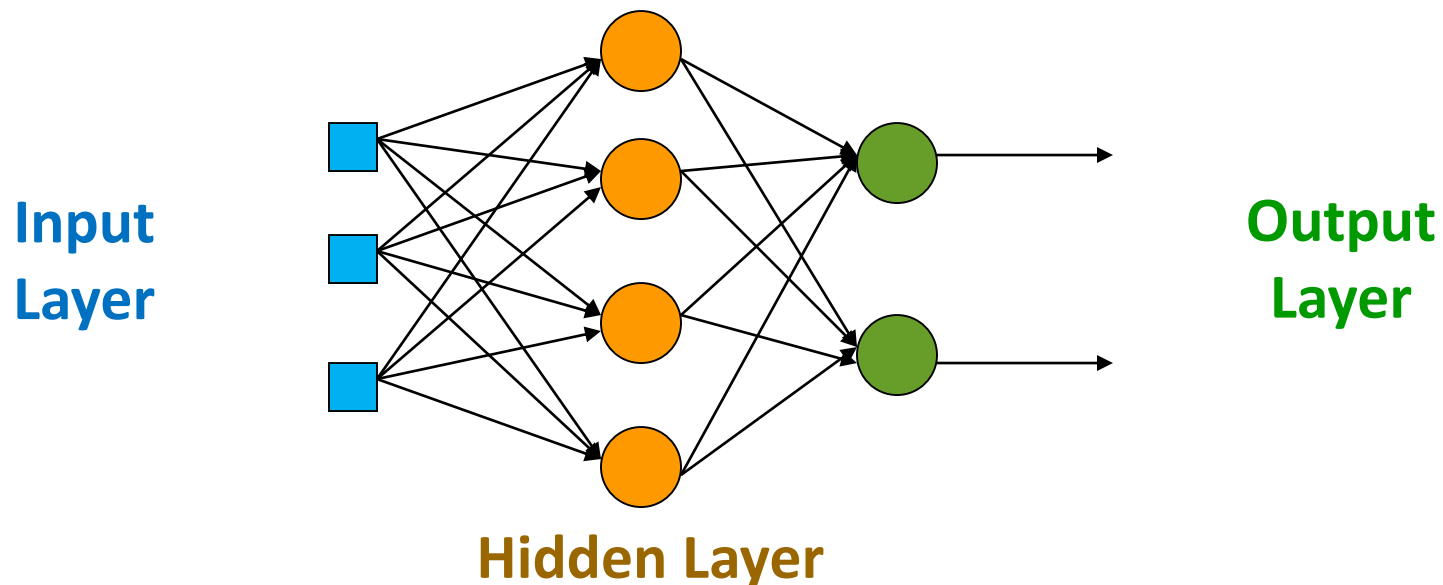
2

- Multi-layer Neural Networks
- Feed forward Neural Networks
 - ▣ FF NN model
 - ▣ Back Propagation (BP) Algorithm
 - ▣ Practical Issues of FFNN
- FFNN Examples

Multi-layer NN

3

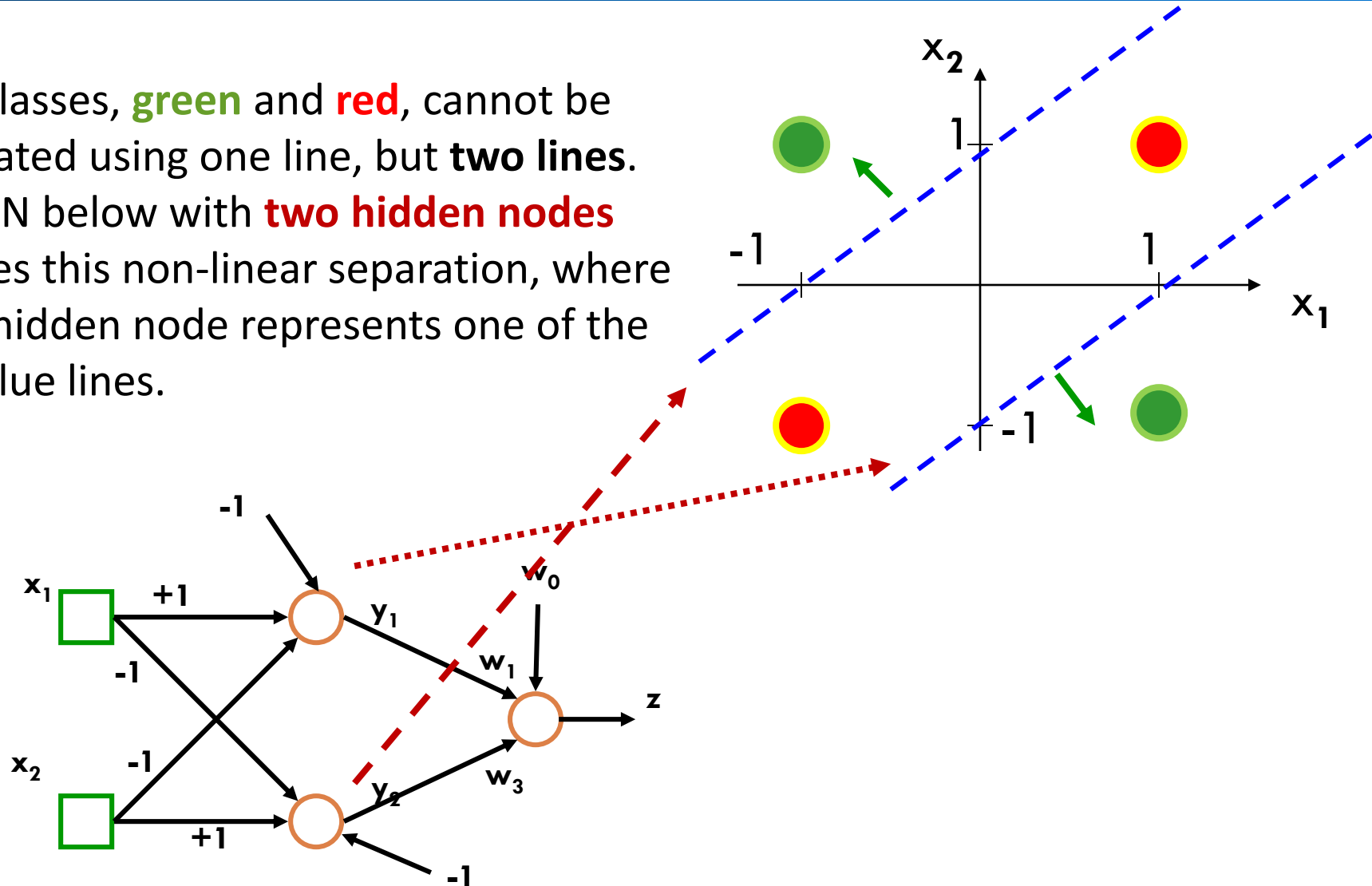
- Between the input and output layers there are hidden layers, as illustrated below.
 - ▣ Hidden nodes do not directly send outputs to the external environment.
- Multi-layer NN overcome the limitation of a single-layer NN
 - ▣ They can handle non-linearly separable learning tasks.



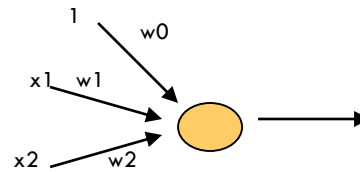
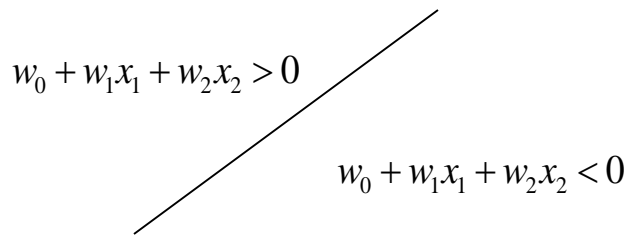
XOR problem

4

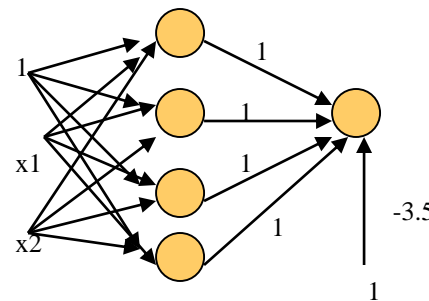
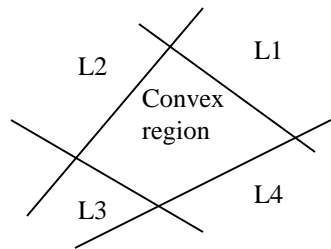
Two classes, **green** and **red**, cannot be separated using one line, but **two lines**. The NN below with **two hidden nodes** realizes this non-linear separation, where each hidden node represents one of the two blue lines.



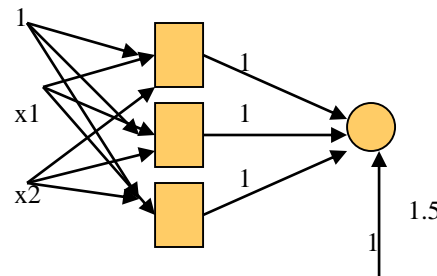
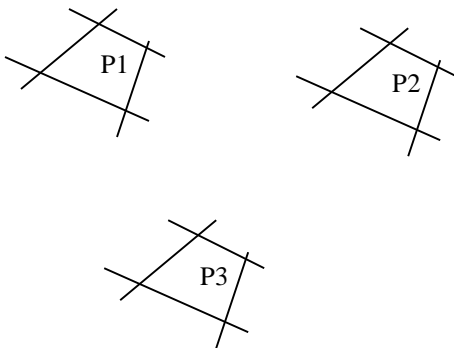
Types of decision regions



Network with a single node

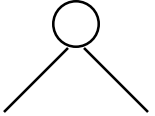
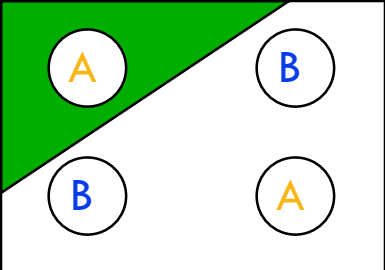
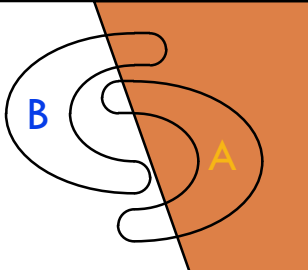
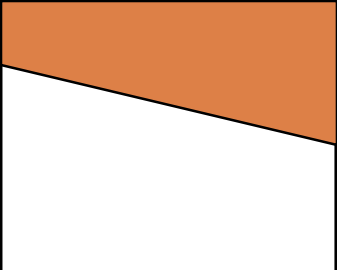
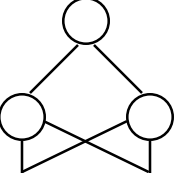
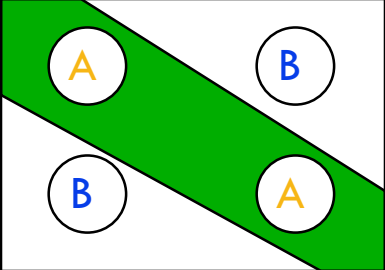
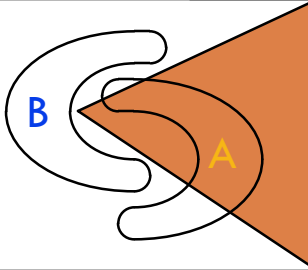
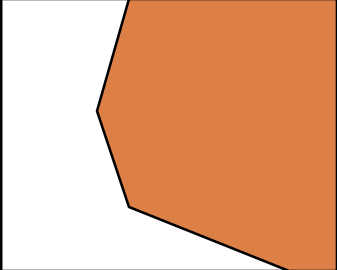
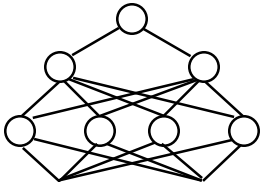
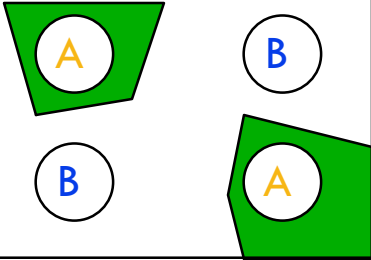
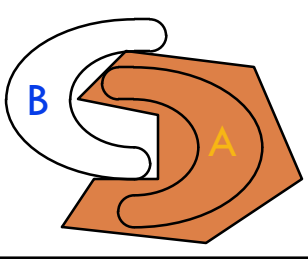
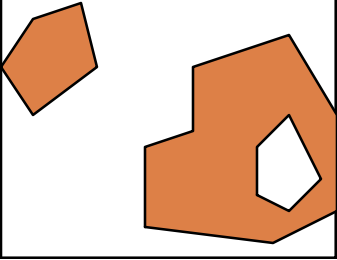


One-hidden layer network that realizes the convex region: Each hidden node realizes one of the lines bounding the convex region



Two-hidden layer network that realizes the union of three convex regions: **each box represents a one hidden layer network realizing one convex region**

Different Non-Linearly Separable Problems

Structure	Types of Decision Regions	Exclusive-OR Problem	Class Separation	Most General Region Shapes
<p>Single-Layer</p> 	<p>Half Plane Bounded By Hyperplane</p>			
<p>Two-Layer</p> 	<p>Convex Open Or Closed Regions</p>			
<p>Three-Layer</p> 	<p>Arbitrary (Complexity Limited by No. of Nodes)</p>			

Outline

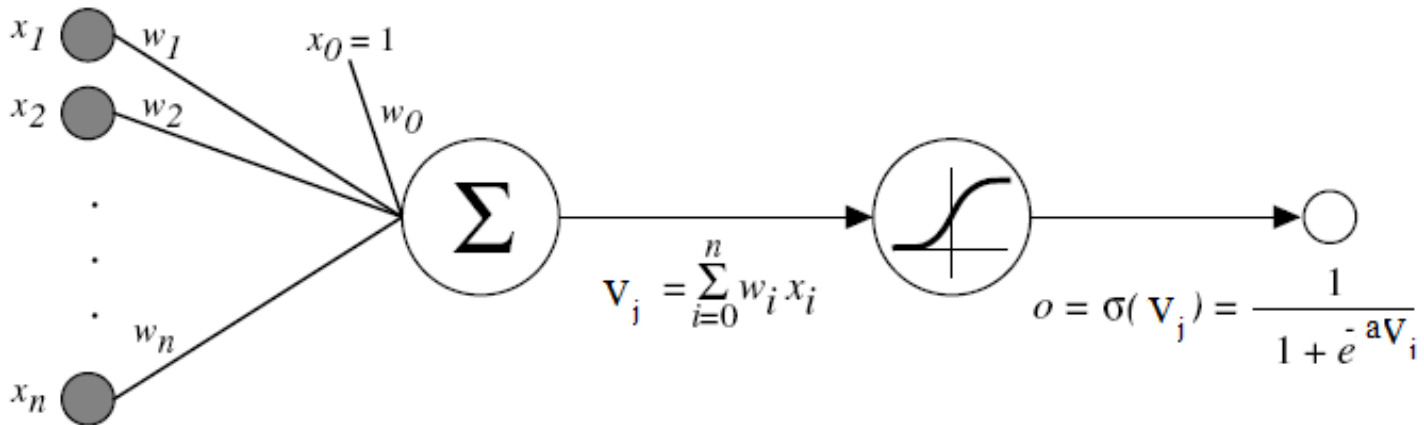
7

- Multi-layer Neural Networks
- **Feedforward Neural Networks**
 - FF NN model
 - Backpropagation (BP) Algorithm
 - BP rules derivation
 - Practical Issues of FFNN
- FFNN Examples

FFNN NEURON MODEL

8

- The classical learning algorithm of FFNN is based on the **gradient descent** method.
- The activation function used in FFNN are continuous functions of the weights, **differentiable** everywhere.
 - ▣ A typical activation function is the **Sigmoid** Function



FFNN NEURON MODEL

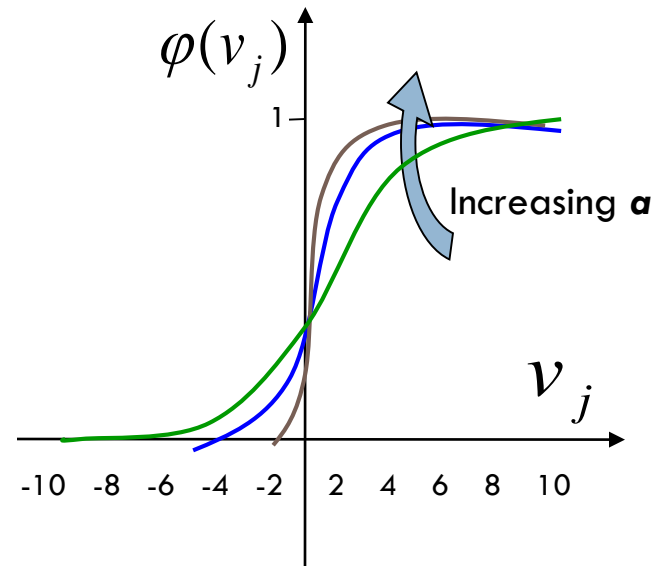
9

- Sigmoid Function:

$$\varphi(v_j) = \frac{1}{1+e^{-av_j}} \quad \text{with } a > 0$$

where $v_j = \sum_i w_{ji} y_i$

with w_{ji} weight of link from node i
to node j and y_i output of node i



- When α approaches to 0, φ tends to a linear function
- When α tends to infinity then φ tends to the step function

The objective of multi-layer NN

- The **error of output neuron j** after the activation of the network on the n -th training example $(x(n), d(n))$ is:

$$e_j(\mathbf{n}) = d_j(\mathbf{n}) - o_j(\mathbf{n})$$

- The **network error** is the sum of the squared errors of the output neurons:

$$E(\mathbf{n}) = \frac{1}{2} \sum_j e_j^2(\mathbf{n})$$

- *The total mean squared error is the average of the network errors over the training examples.*

$$E(W) = \frac{1}{N} \sum_{n=1}^N E(\mathbf{n}) = \frac{1}{2N} \sum_n \sum_j (d_j(n) - o_j(n))^2$$

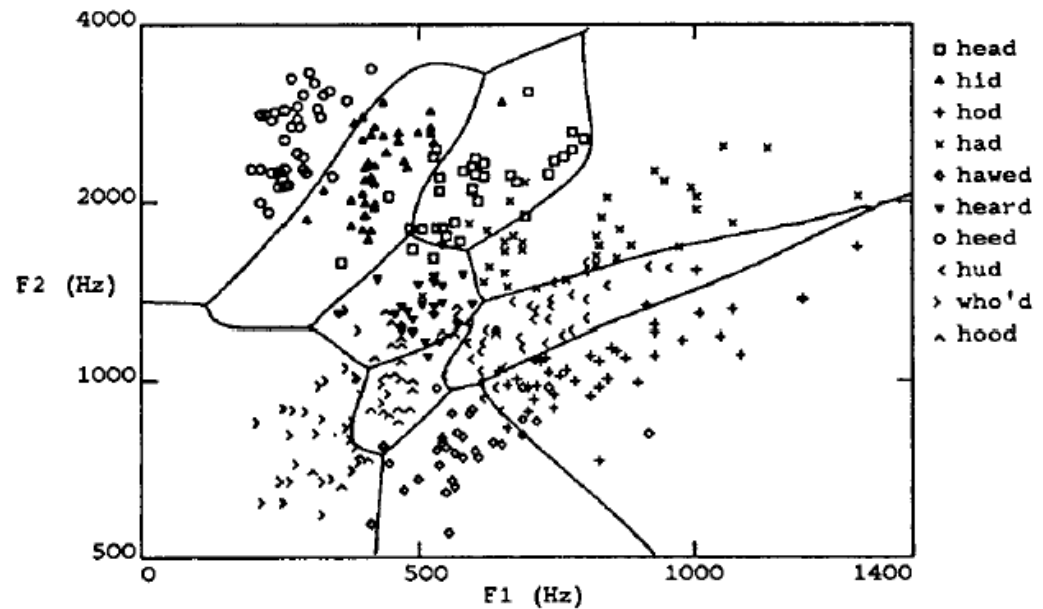
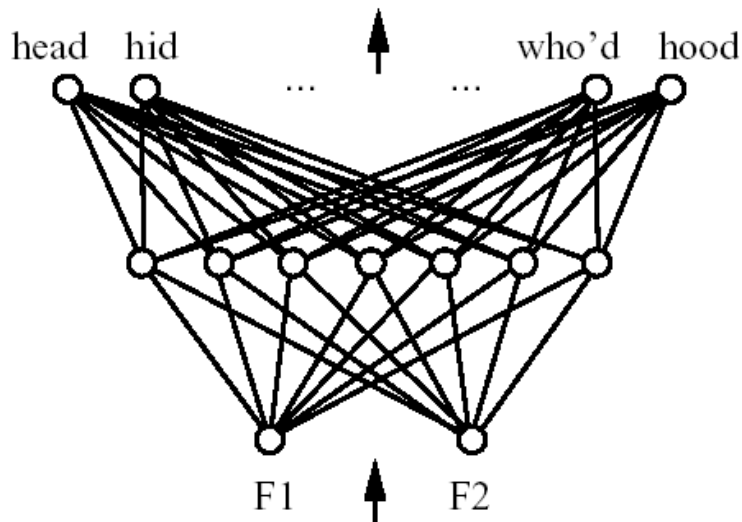
Feed forward NN

11

- **Idea: Credit assignment problem**
- Problem of assigning **credit** or **blame** to individual elements involving in forming overall response of a learning system (hidden units)
- In neural networks, problem relates to distributing the network error to the weights.

Multilayer Networks of Sigmoid Units

12



Sigmoid Unit

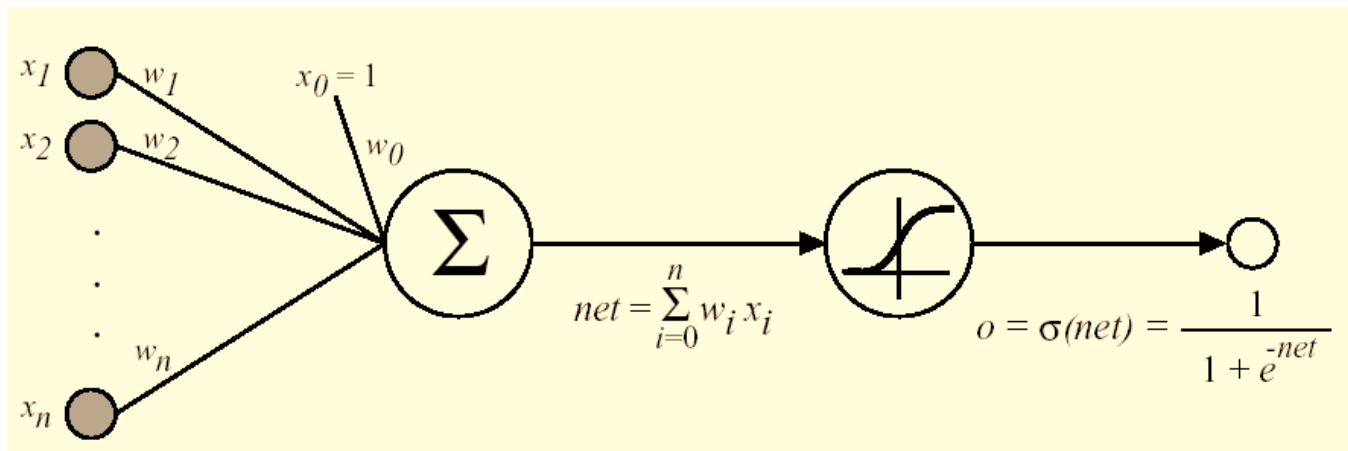
13

$\sigma(x)$ is the sigmoid function $\frac{1}{1 + e^{-x}}$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train:

- One sigmoid unit
- **Multilayer networks** of sigmoid units → Backpropagation



Error Gradient for a Sigmoid Unit

14

We know:

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\ &= -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}\end{aligned}$$

So:

$$\begin{aligned}\frac{\partial o_d}{\partial net_d} &= \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d) \\ \frac{\partial net_d}{\partial w_i} &= \frac{\partial(\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}\end{aligned}$$

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

Keep in mind
for later use!

Outline

15

- Multi-layer Neural Networks
- Feed forward Neural Networks
 - FF NN model
 - **Backpropagation (BP) Algorithm**
 - Practical Issues of FFNN
- FFNN Examples

Training: Backprop algorithm

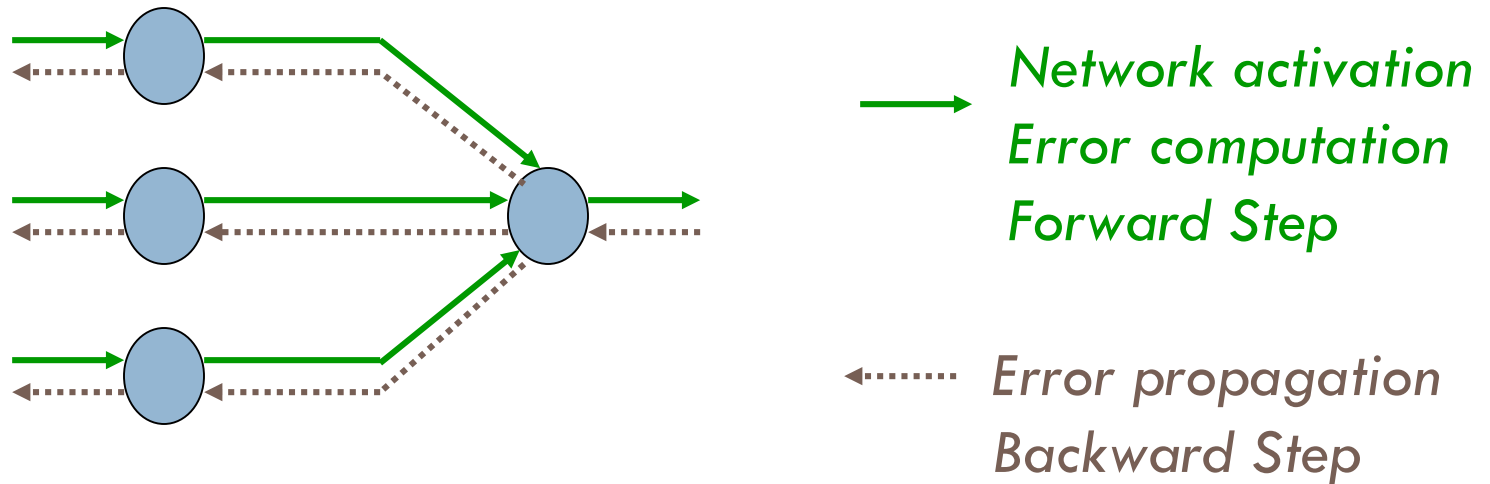
16

- Searches for weight values that **minimize the total error of the network** over the set of training examples.
- **Repeated** procedures of the following two passes:
 - **Forward pass**: Compute the **outputs** of all units in the network, and the **error** of the output layers.
 - **Backward pass**: The network error is used for updating the weights (**credit assignment problem**).
 - Starting at the output layer, **the error is propagated backwards through the network, layer by layer**. This is done by recursively computing the **local gradient** of each neuron.

Backprop

17

- Back-propagation training algorithm illustrated:



- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

BP for the case of sigmoid

18

Initialize all weights to small random numbers.

- While Unsatisfied, Do
- For each training example, Do
 - **Feed Forward:** Input the training example to the network and compute the network outputs
 - **Gradient Descent:** For each output unit k : $\delta_k \leftarrow o_k(1 - o_k) (t_k - o_k)$
 - **Backprop:** For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$

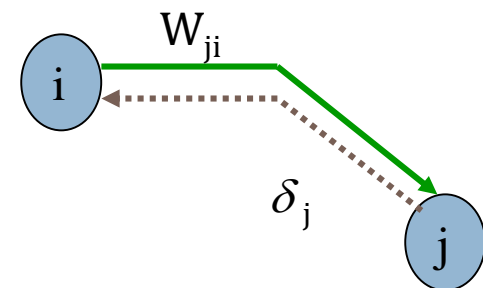
- **Adjust Weights:** Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\text{where } \Delta w_{ji} = \eta \delta_j y_i$$

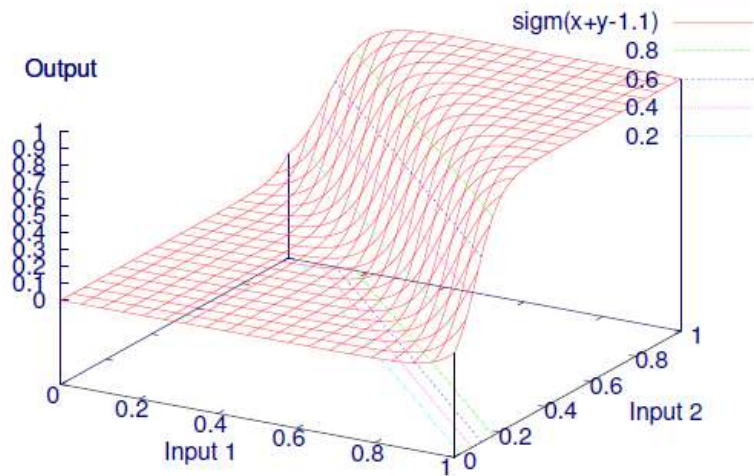
y_i is the output of neuron i in the previous layer:

- End For
- End While

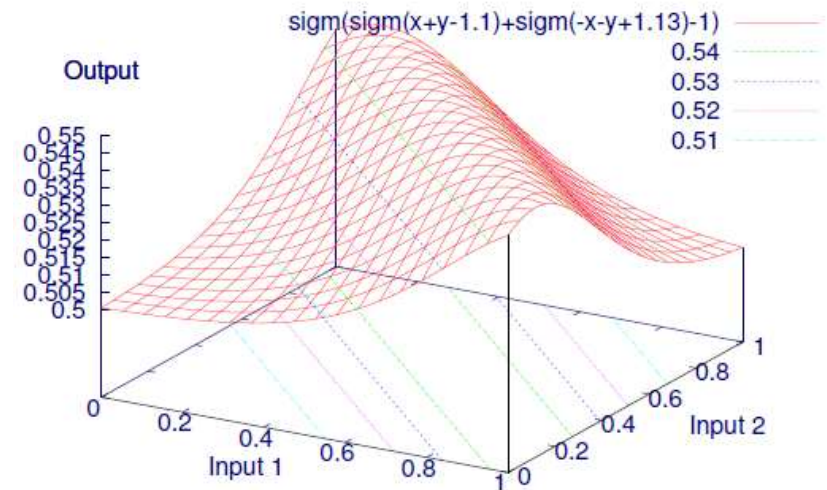


Nonlinear decision surfaces

19



One output
No hidden



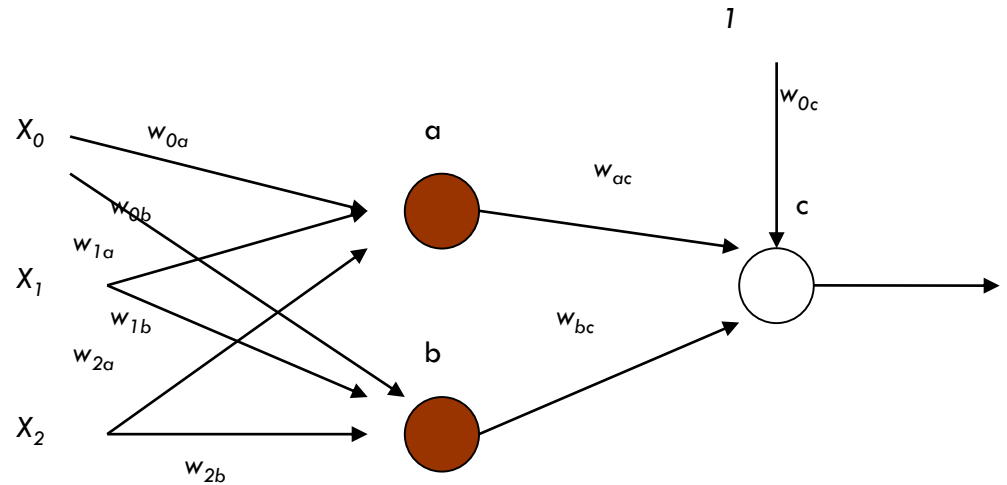
One output
Two hidden

BP Example

20

□ XOR

□ X0	X1	X2	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Sigmoid A.F. ; $\eta=0.5$; Sample{(1, 0, 0), 0}

Neuron a		Neuron b		Neuron C	
$w_{oa}=0.34$	$v_a=0.34$	$w_{ob}=-0.12$	$v_b=-0.12$	$w_{oc}=-0.99$	$v_c=-0.54$
$w_{1a}=0.13$	$o_a=0.58$	$w_{1b}=0.57$	$o_b=0.47$	$w_{ac}=0.16$	$o_c=0.37$
$w_{2a}=-0.92$		$w_{2b}=-0.33$		$w_{bc}=0.75$	
$\delta_a = o_a(1-o_a)\sum_k w_{ak} \cdot \delta_k$ $= 0.58*(1-0.58)*0.16*(-0.085)$ $= -0.003$		$\delta_b = o_b(1-o_b)\sum_k w_{bk} \cdot \delta_k$ $= 0.47*(1-0.47)*0.75*(-0.085)$ $= -0.016$		$\delta_c = o_c(1-o_c)(t_c - o_c)$ $= 0.37*(1-0.37)*(0-0.37)$ $= -0.085$	
$\Delta w_{oa} = \eta \delta_a x_0 = 0.5*(-0.003)*1$ $= -0.015$		$\Delta w_{ob} = \eta \delta_b x_0 = 0.5*(-0.016)*1$ $= -0.008$		$\Delta w_{oc} = \eta \delta_c 1 = 0.5*(-0.085)*1$ $= -0.043$	
$\Delta w_{1a} = \eta \delta_a x_1 = 0.5*(-0.003)*0 = 0$		$\Delta w_{1b} = \eta \delta_b x_1 = 0.5*(-0.01)*0 = 0$		$\Delta w_{ac} = \eta \delta_c O_a = 0.5*(-0.085)*0.58 = -0.025$	
$\Delta w_{2a} = \eta \delta_a x_2 = 0.5*(-0.003)*0 = 0$		$\Delta w_{2b} = \eta \delta_b x_2 = 0.5*(-0.01)*0 = 0$		$\Delta w_{bc} = \eta \delta_c O_b = 0.5*(-0.085)*0.47 = -0.020$	

Weight updating

Neuron a		Neuron b		Neuron C	
$w_{oa} = w_{oa} + \Delta w_{oa} = 0.34 - 0.015 = 0.325$		$w_{ob} = w_{ob} + \Delta w_{ob} = -0.12 - 0.008$		$w_{oc} = w_{oc} + \Delta w_{oc} = -0.99 - 0.043$	
$w_{1a} = w_{1a} + \Delta w_{1a} = 0.13 + 0$		$w_{1b} = w_{1b} + \Delta w_{1b} = 0.57 + 0$		$w_{ac} = w_{ac} + \Delta w_{ac} = 0.16 - 0.025$	
$w_{2a} = w_{2a} + \Delta w_{2a} = -0.92 + 0$		$w_{2b} = w_{2b} + \Delta w_{2b} = -0.33 + 0$		$w_{bc} = w_{bc} + \Delta w_{bc} = 0.75 - 0.02$	
$\Delta w_{oa} = \eta \delta_a x_0 = 0.5 * (-0.003) * 1 = -0.015$		$\Delta w_{ob} = \eta \delta_b x_0 = 0.5 * (-0.016) * 1 = -0.008$		$\Delta w_{oc} = \eta \delta_c 1 = 0.5 * (-0.085) * 1 = -0.043$	
$\Delta w_{1a} = \eta \delta_a w_{1a} = 0.5 * (-0.003) * 0 = 0$		$\Delta w_{1b} = \eta \delta_b w_{1b} = 0.5 * (-0.01) * 0 = 0$		$\Delta w_{ac} = \eta \delta_c w_{ac} = 0.5 * (-0.085) * 0.58 = -0.025$	
$\Delta w_{2a} = \eta \delta_a w_{2a} = 0.5 * (-0.003) * 0 = 0$		$\Delta w_{2b} = \eta \delta_b w_{2b} = 0.5 * (-0.01) * 0 = 0$		$\Delta w_{bc} = \eta \delta_c w_{bc} = 0.5 * (-0.085) * 0.47 = -0.020$	

Backpropagation: Properties

- Gradient descent over entire network weight vector.
- Easily generalized to arbitrary directed graphs.
- Will find a local, not necessarily global error minimum:
 - ▣ In practice, often works well (can run multiple times with different initial weights).
- Minimizes error over training examples:
 - ▣ Will it generalize well to subsequent examples?
- Training can take hundreds of iterations → slow
- Using the network after training is very fast.

تنفس

پسر کوچکی وارد مغازه ای شد، جعبه نوشابه را به سمت تلفن هل داد و بر روی جعبه رفت تا دستش به دکمه های تلفن برسد و شروع کرد به گرفتن شماره.

مغازه دار متوجه پسر بود و به مکالماتش گوش می داد.

پسرک پرسید: خانم، می توانم خواهش کنم کوتاه کردن چمن های حیاط خانه تان را به من بسپارید؟

زن پاسخ داد: کسی هست که این کار را برایم انجام می دهد!

پسرک گفت: خانم، من این کار را با نصف قیمتی که او می دهد انجام خواهم داد!

زن در جوابش گفت: که از کار این فرد کاملاً راضی است.

تنفس

پسرک بیشتر اصرار کرد و پیشنهاد داد: خانم، من پیاده رو و جدول جلوی خانه را هم برایتان جارو می کنم. در این صورت شما در یکشنبه زیباترین چمن را در کل شهر خواهید داشت.
مجددا زن پاسخش منفی بود.

حاسبوا قبل ان تحاسبوا

کاری به تو بدهم.
پسر جواب داد: نه ممنون، من فقط داشتم عملکردم را می سنجیدم.
من همان کسی هستم که برای این خانم کار می کند!

آیا ما هم می توانیم چنین ارزیابی از کار خود داشته باشیم؟