CS-688 Spring 2016
Neural Networks

Outline

- Perceptron: limitations;
- Feedforward networks and Backpropagation;

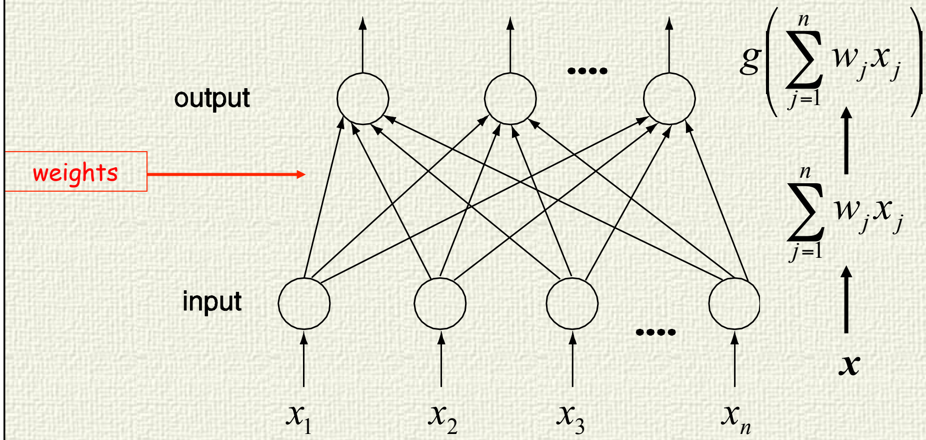
What is a Neural Network, anyway?

- Often associated with biological devices (brains), electronic devices, or network diagrams;
- But the best conceptualization for this presentation is **none** of these: ***think of a neural network as a mathematical function***

The pros of Neural Networks

- Successfully used on a variety of domains:
  PC games, Business strategy, Buyer prospect selection, Stock market analysis, Consumer price forecasts, Cost analysis, Employee selection, Intelligent software applications, Legal strategies, Managerial decision making, Personnel profiling, Process control, Quality control, Real estate market forecasting, Sales forecasts, Security analysis, Spectral analysis, Stock market analysis, Temperature and weather prediction, Troubleshooting and much more.
- Can provide solutions to very complex and nonlinear problems;
- If provided with sufficient amount of data, can solve classification and forecasting problems accurately and easily
- Once trained, prediction is fast;

Introduction: A Simple Architecture



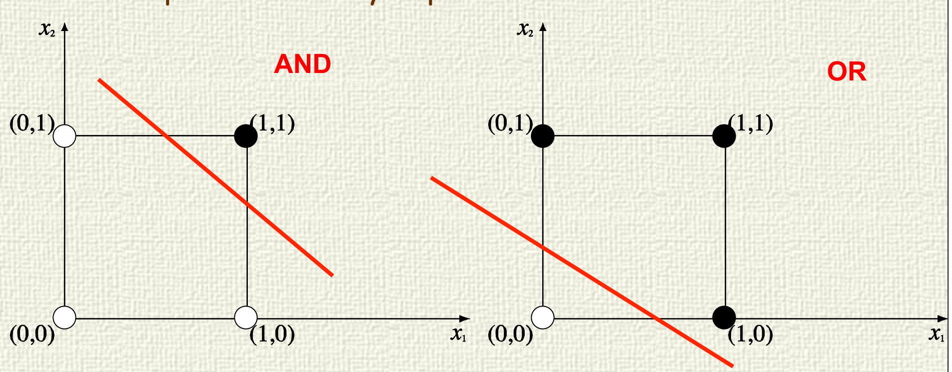
5

Representational Power of Perceptrons

- Marvin Minsky and Seymour Papert, "Perceptron" 1969:

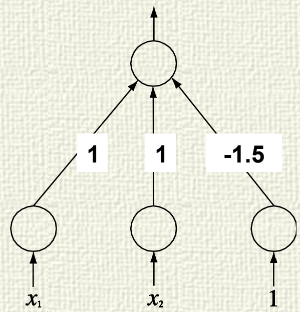
"The perceptron can solve only problems with linearly separable classes."

- Examples of linearly separable Boolean functions:

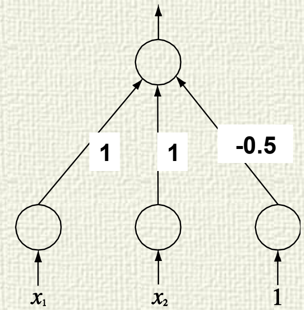


6

Representational Power of Perceptrons



Perceptron that computes the AND function

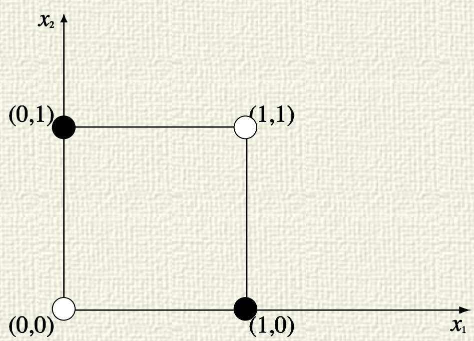


Perceptron that computes the OR function

7

Representational Power of Perceptrons

- Example of a non linearly separable Boolean function:

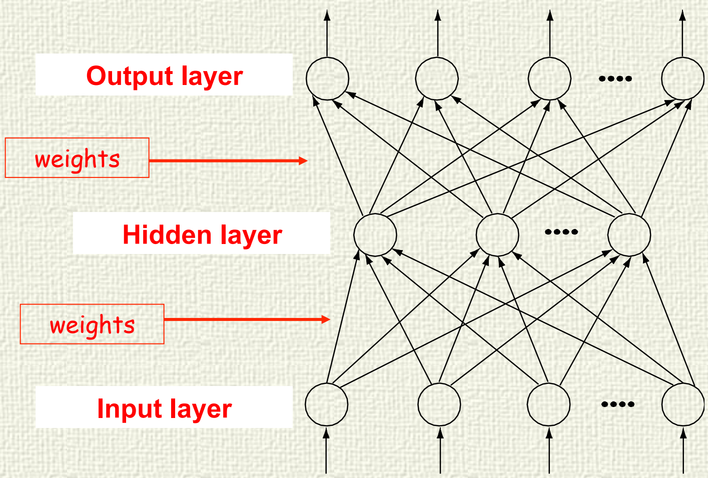


EX-OR

The EX-OR function **cannot** be computed by a perceptron

8

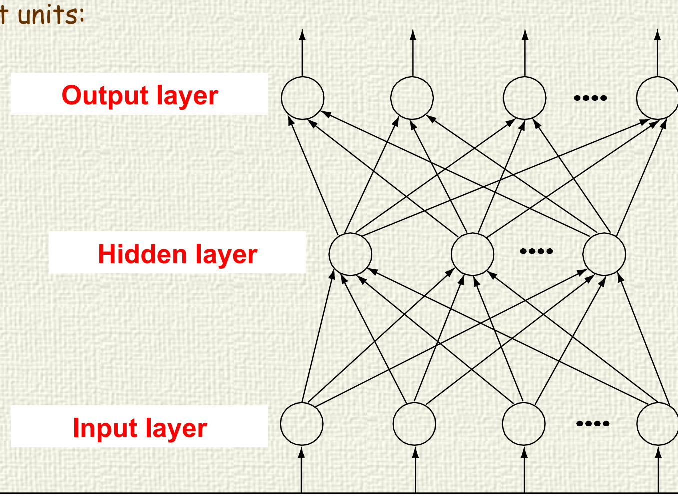
Adding a Hidden Layer



9

Multilayer Neural Networks

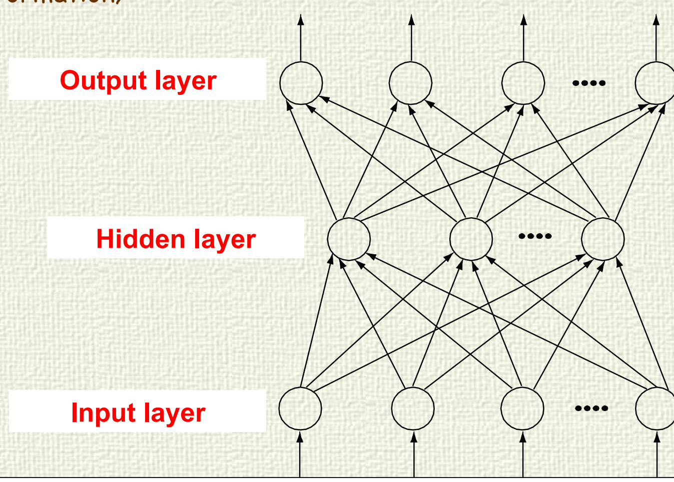
- Generalization of a perceptron;
- Achieve increased computational power;
- Idea: Introduce layers of units between the input and the output units:



10

Multilayer Neural Networks

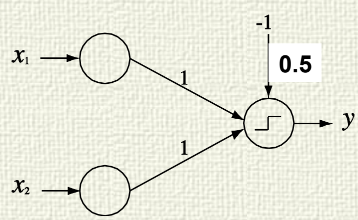
- Allow to learn non linearly separable transformations from input to output;
- A single hidden layer allows to compute any input/output transformation;



11

Example: EX-OR

- Consider first a perceptron:



- Correct answer in three cases:

$$\boldsymbol{x}_1 = 0, \boldsymbol{x}_2 = 0 \qquad H(-0.5) = 0$$
$$\boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = 0 \qquad H(1-0.5) = 1$$
$$\boldsymbol{x}_1 = 0, \boldsymbol{x}_2 = 1 \qquad H(1-0.5) = 1$$

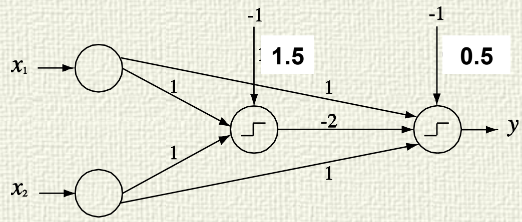
- 4th case:

$$\boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = 1 \qquad H(1+1-0.5) = 1$$ **Wrong!**

12

Example: EX-OR (contd.)

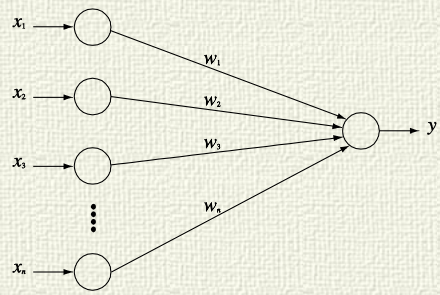
- **Idea**: Introduce one hidden unit with a large enough threshold, so that it is activated only in the 4th case. The hidden unit provides a negative input to the output unit to correct its response in the 4th case



- First three cases: as before. **OK**
- 4th case: $x_1 = 1, x_2 = 1 \qquad H(1+1-2-0.5)=0$ **OK!**

13

Multilayer Neural Networks



- **Activation function**:

Differentiable function g:

$$y = g\left(\sum_k w_k x_k\right) \in (0,1)$$

- **Network's dynamic**:

f: target transformation (unknown) from input to output;

For each configuration x of the input layer, the network computes a configuration y of the output layer;

The network adjusts the weights so that, after a finite number of steps, the network's output $y \sim f(x)$

- **Criterion:**

Minimize the difference between the network's response and the desired output.

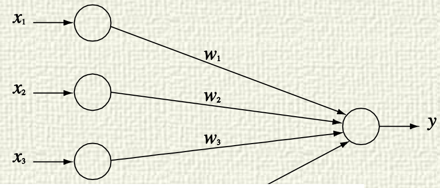
14

Learning Algorithm: No Hidden Units first

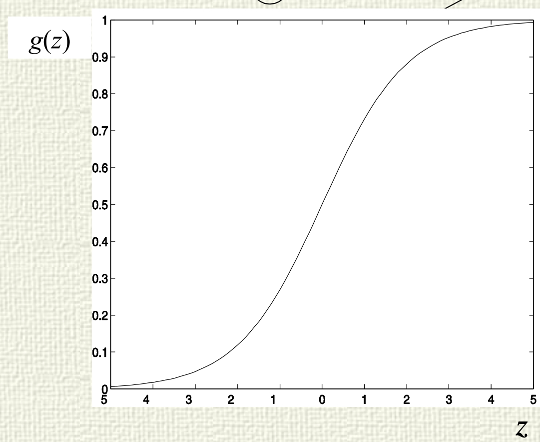
$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$y = g(z)$$

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



Sigmoid function



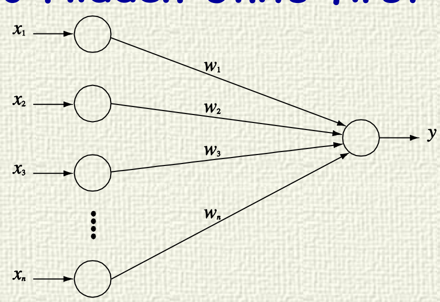
Learning Algorithm: No Hidden Units first

$$J(\boldsymbol{w}) = \frac{1}{2}\left(y^* - y\right)^2$$

$$y = g(\boldsymbol{w}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

$$\frac{\partial y}{\partial w_i} = y(1-y)x_i$$

$$\frac{\partial J}{\partial w_i} = -(y^* - y)y(1-y)x_i$$



By applying gradient descent:

$$\Delta w_i = \mu y(1-y)(y^* - y)x_i$$

DELTA RULE for the Sigmoid function (no hidden units)

$$\Delta w_i = w_i^{t+1} - w_i^t$$

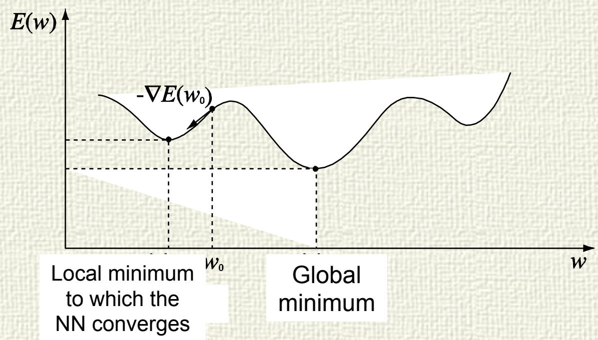
μ is the learning rate

Generalization of Delta Rule for Feedforward Networks

Fixed target function we want to learn: $\boldsymbol{t}_k = f(\boldsymbol{x}_k)$

Error over input $\boldsymbol{x}_k$ $\quad E_k = \frac{1}{2}\sum_j \left(t_{kj} - y_{kj}\right)^2$

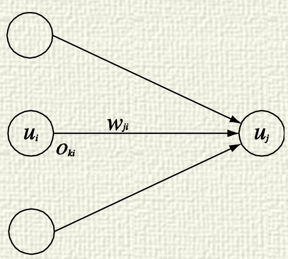
Total error $\quad E = \sum_k E_k$



Backpropagation algorithm: provides an efficient procedure to compute derivatives

17

Backpropagation algorithm



$$z_{kj} = \sum_i w_{ji} o_{ki}$$

$$o_{kj} = g_j\left(z_{kj}\right),\ g_j \ \textbf{\textit{differentiable in}}\ z_{kj}$$

<u>Goal</u>: learn the weights so that the mean squared error is minimized

18

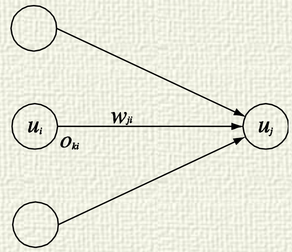
Backpropagation

Fixed target function we want to learn: $\boldsymbol{t}_k = f(\boldsymbol{x}_k)$

Error over input $\boldsymbol{x}_k$ $\quad E_k = \frac{1}{2}\sum_j \left(t_{kj} - y_{kj}\right)^2$

Total error $\quad E = \sum_k E_k$

We want $\quad \Delta_k w_{ji} \propto -\frac{\partial E_k}{\partial w_{ji}}$



$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial (z_{kj})}\frac{\partial (z_{kj})}{\partial w_{ji}}$$

$$z_{kj} = \sum_i w_{ji} o_{ki} \qquad \frac{\partial (z_{kj})}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\sum_t w_{jt} o_{kt} = o_{ki}$$

Lets define $\quad \frac{\partial E_k}{\partial (z_{kj})} = -\delta_{kj} \Rightarrow \frac{\partial E_k}{\partial w_{ji}} = -\delta_{kj} o_{ki}$

Thus: to perform a gradient descent on the surface of E we need to modify the weights as:

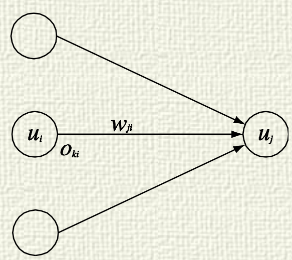
$$\boxed{\Delta_k w_{ji} = \mu \delta_{kj} o_{ki}}$$

19

Backpropagation

We need to compute the values δ_{kj}: $\quad \frac{\partial E_k}{\partial (z_{kj})} = -\delta_{kj}$

$$\delta_{kj} = -\frac{\partial E_k}{\partial (z_{kj})} = -\frac{\partial E_k}{\partial o_{kj}}\frac{\partial o_{kj}}{\partial (z_{kj})}$$



From $\quad o_{kj} = g_j\left(z_{kj}\right) \qquad \frac{\partial o_{kj}}{\partial (z_{kj})} = g_j'\left(z_{kj}\right)$

To compute $\frac{\partial E_k}{\partial o_{kj}}$ we distinguish two cases:

- 1st case: u_j is an output unit
- 2nd case: u_j is a hidden unit

20

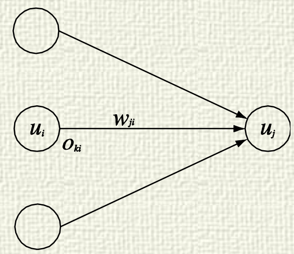
Backpropagation

- 1^{st} case: u_j is a output unit

$$because\ E_k = \frac{1}{2}\sum_j \left(t_{kj} - y_{kj}\right)^2$$

$$\frac{\partial E_k}{\partial o_{kj}} = -\left(t_{kj} - y_{kj}\right) = -\left(t_{kj} - o_{kj}\right)$$

$$\Rightarrow \delta_{kj} = \left(t_{kj} - o_{kj}\right) g_j^{'}\left(z_{kj}\right)$$



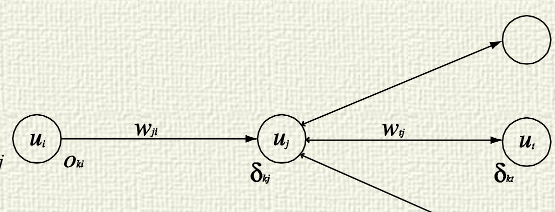
Backpropagation

2^{nd} case: u_j is a hidden unit

$$\frac{\partial E_k}{\partial o_{kj}} = \sum_t \frac{\partial E_k}{\partial z_{kt}} \frac{\partial z_{kt}}{\partial o_{kj}} = \sum_t \frac{\partial E_k}{\partial z_{kt}} \frac{\partial}{\partial o_{kj}} \left(\sum_l w_{tl} o_{kl} \right) =$$

$$\sum_t \frac{\partial E_k}{\partial z_{kt}} w_{tj} = -\sum_t \delta_{kt} w_{tj}$$

$$\Rightarrow \delta_{kj} = g_j^{'}\left(z_{kj}\right) \sum_t \delta_{kt} w_{tj}$$



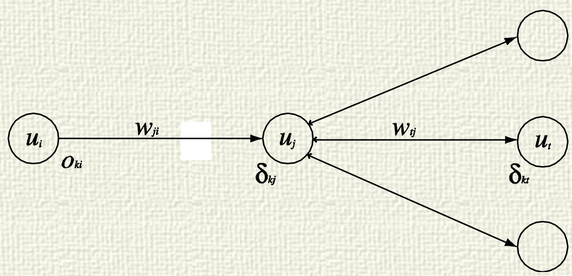
Recursive procedure to compute δ for all the units of the network!

Such δ are used in: $\boxed{\Delta_k w_{ji} = \mu \delta_{kj} o_{ki}}$

Wrapping up the Backpropagation Algorithm

Three key equations:

- **Generalized Delta rule:** $\boxed{\Delta_k w_{ji} = \mu \delta_{kj} o_{ki}}$
- **For output units the error signal is:** $\boxed{\delta_{kj} = (t_{kj} - o_{kj}) g_j^{'}(z_{kj})}$
- **For hidden units, the error signal is:** $\boxed{\delta_{kj} = g_j^{'}(z_{kj}) \sum_t \delta_{kt} w_{tj}}$



Backpropagation: Summary

- Activation: each input unit u_j is given the state x_{kj}
- Signal propagation: For each hidden and output unit, compute

$$o_{kj} = g_j\left(\sum_i w_{ji} o_{ki}\right)$$

- Comparison: For each output unit u_j compute:

$$\delta_{kj} = (t_{kj} - o_{kj}) g_j^{'}\left(\sum_i w_{ji} o_{ki}\right)$$

- Backpropagation: (the computed δ become the input of the reversed network) For each hidden unit u_j compute:

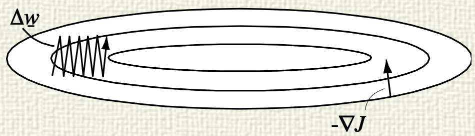
$$\delta_{kj} = f_j^{'}\left(\sum_i w_{ji} o_{ki}\right) \sum_t \delta_{kt} w_{tj}^{k-1}$$

- Weight Update:

$$w_{ji}^{k} = w_{ji}^{k-1} + \mu \delta_{kj} o_{ki}$$

Learning Rate and Momentum

- μ too small $\Longrightarrow$ very slow learning rate
- μ too large $\Longrightarrow$ oscillating behavior



- We want to set μ as large as possible avoiding oscillations
- Solution: introduce **momentum** in the learning rule. The momentum includes the direction of the previous update:

$$\Delta w_{ji}(n+1) = \mu \delta_{kj} o_{ki} + \alpha \Delta w_{ji}(n)$$

$$\alpha = 0.9$$

Backpropagation: applications

- Perhaps the most successful and widely used learning algorithm for NNs;
- Used in a variety of domains:
  - clinical diagnosis,
  - predicting protein structure,
  - character recognition,
  - fingerprint recognition,
  - modeling residual chlorine decay in water,
  - weather forecast,
  - waveform recognition,
  - backgammon, etc.

References

- Original paper on backpropagation:
  - Rumelhart, Hinton, Williams, *Learning internal representations by error propagation*, 1986. In Parallel Distributed Processing, Vol1.