Neural Networks
[RN2] Sec 20.5
[RN3] Sec 18.7

CS 486/686

University of Waterloo

Lecture 17: June 26, 2017

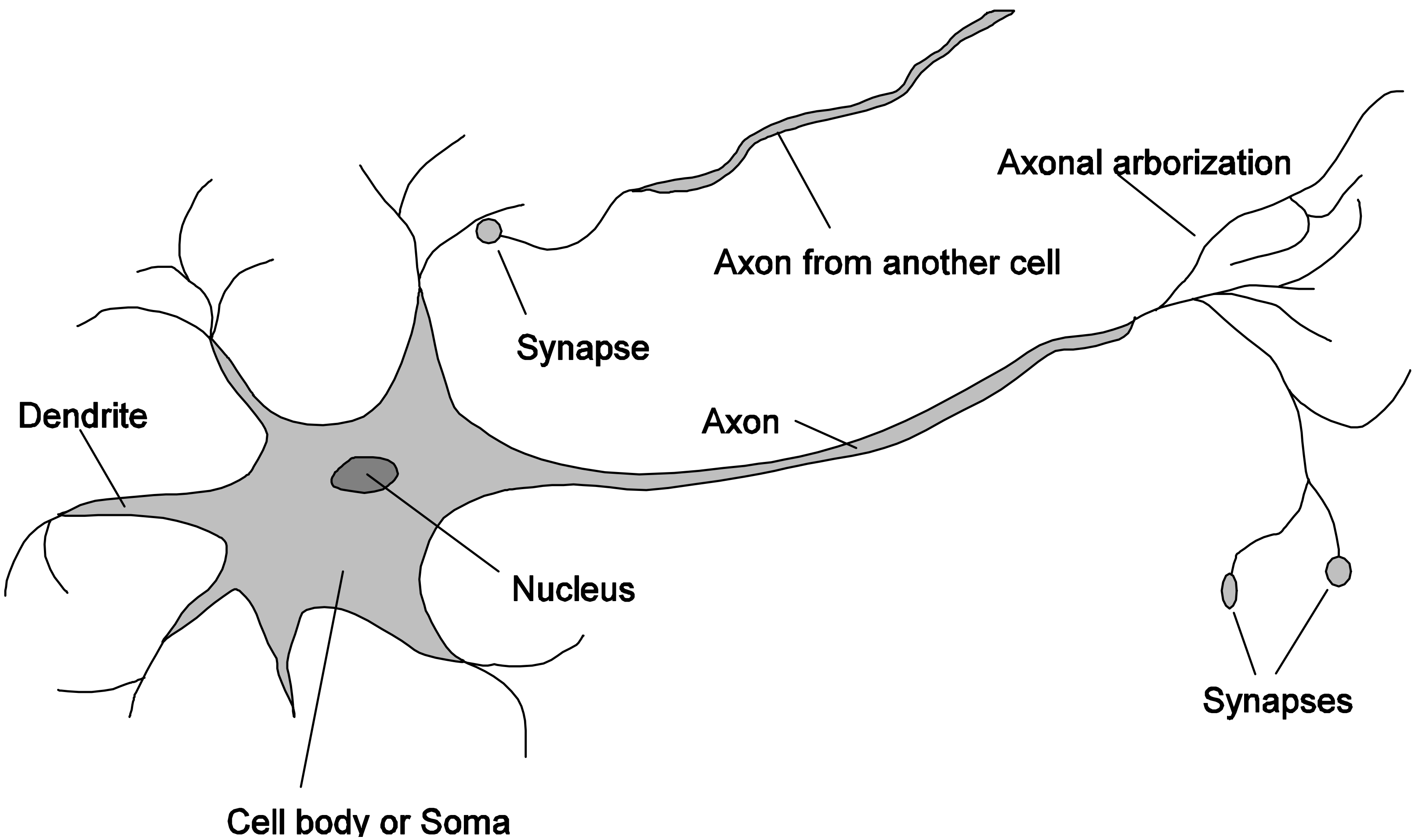
Outline

- Neural networks
  - Perceptron
  - Supervised learning algorithms for neural networks

Brain

- Seat of human intelligence
- Where memory/knowledge resides
- Responsible for thoughts and decisions
- Can learn
- Consists of nerve cells called neurons

Neuron


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

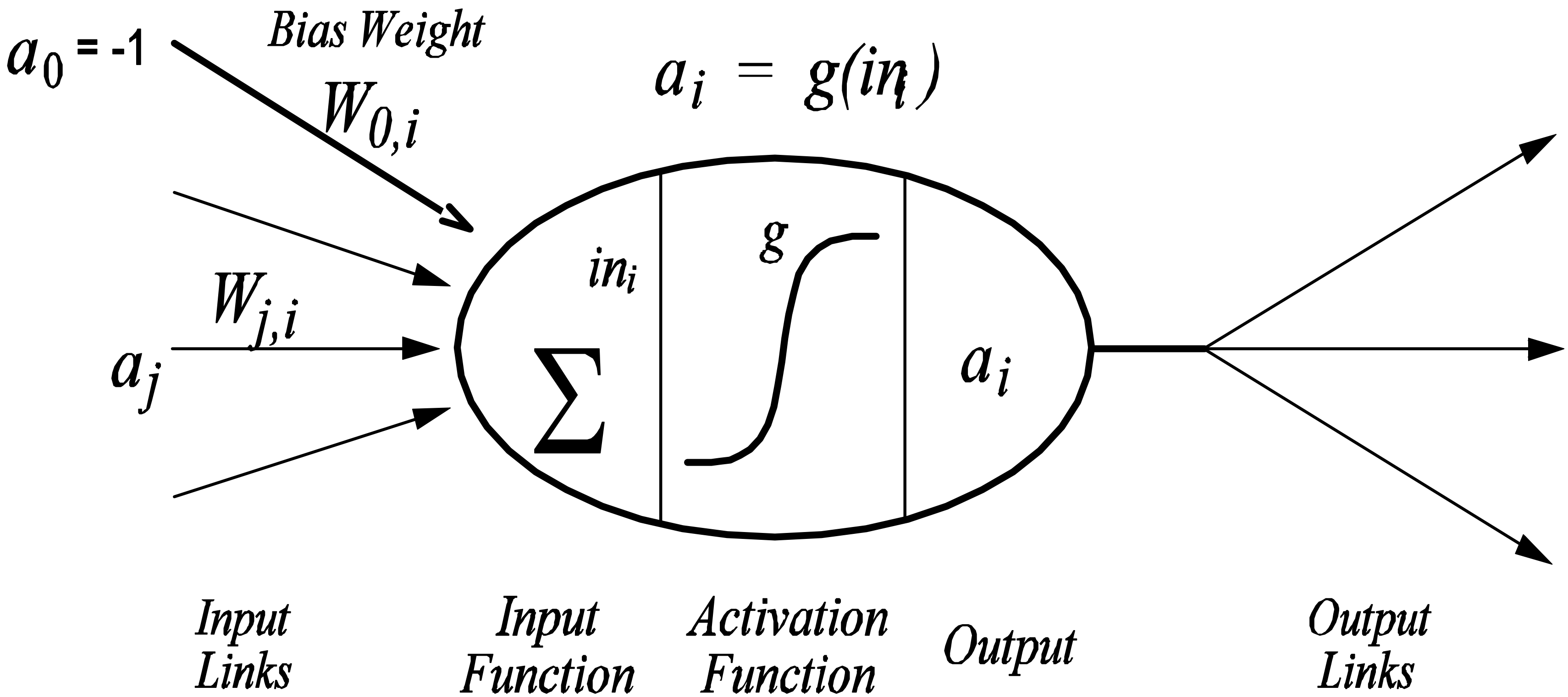
Artificial Neural Networks

- Idea: mimic the brain to do computation
- Artificial neural network:
  - Nodes (a.k.a. units) correspond to neurons
  - Links correspond to synapses
- Computation:
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal correspond to neurons firing rate

ANN Unit

- For each unit i:
- Weights: W_{ji}
  - Strength of the link from unit j to unit i
  - Input signals a_j weighted by W_{ji} and linearly combined: $in_i = \Sigma_j W_{ji} a_j$
- Activation function: g
  - Numerical signal produced: $a_i = g(in_i)$

ANN Unit



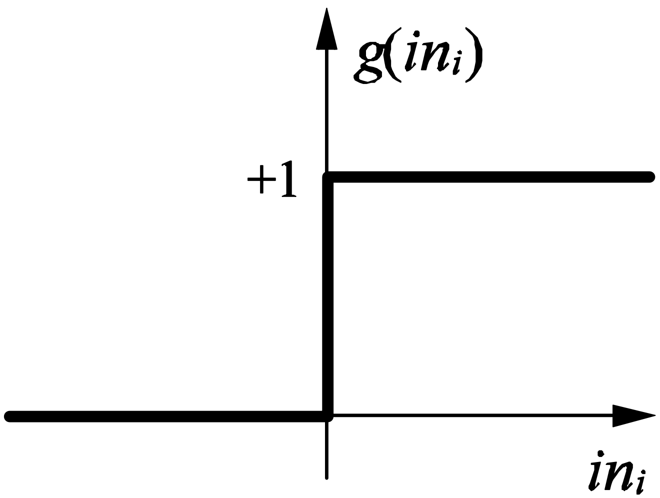
a_0 = -1
Bias Weight
$W_{0,i}$
$a_i = g(in_i)$
$W_{j,i}$
a_j
in_i
Σ
g
a_i
Input Links
Input Function
Activation Function
Output
Output Links

Activation Function

- Should be nonlinear
  - Otherwise network is just a linear function
- Often chosen to mimic firing in neurons
  - Unit should be "active" (output near 1) when fed with the "right" inputs
  - Unit should be "inactive" (output near 0) when fed with the "wrong" inputs

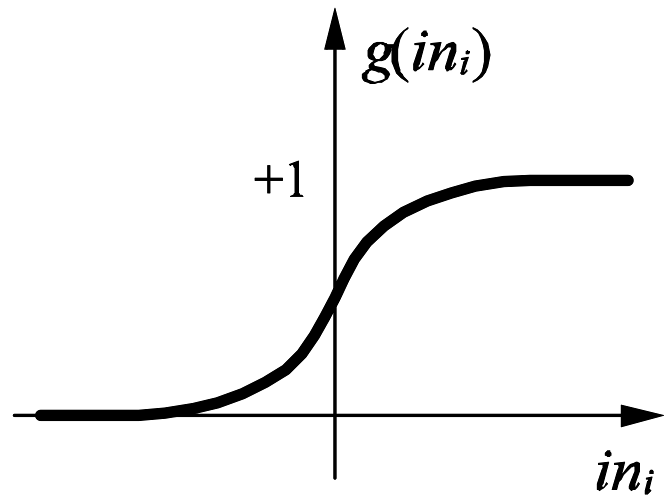
Common Activation Functions

Threshold



(a)

Sigmoid

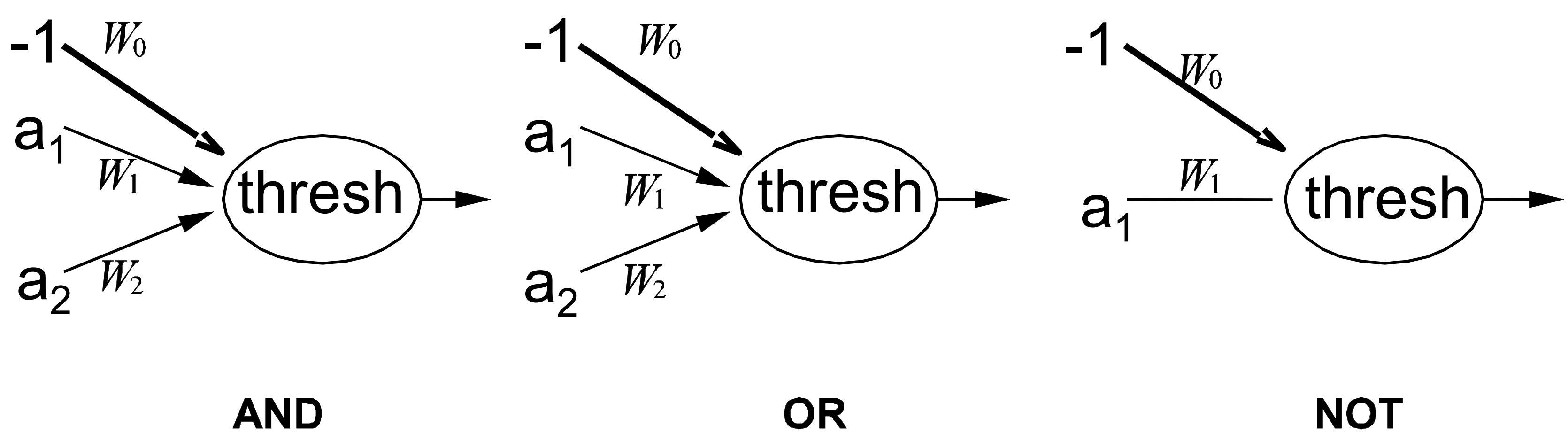


(b)

$$g(x) = 1/(1+e^{-x})$$

Logic Gates

- McCulloch and Pitts (1943)
  - Design ANNs to represent Boolean fns
- What should be the weights of the following units to code AND, OR, NOT ?

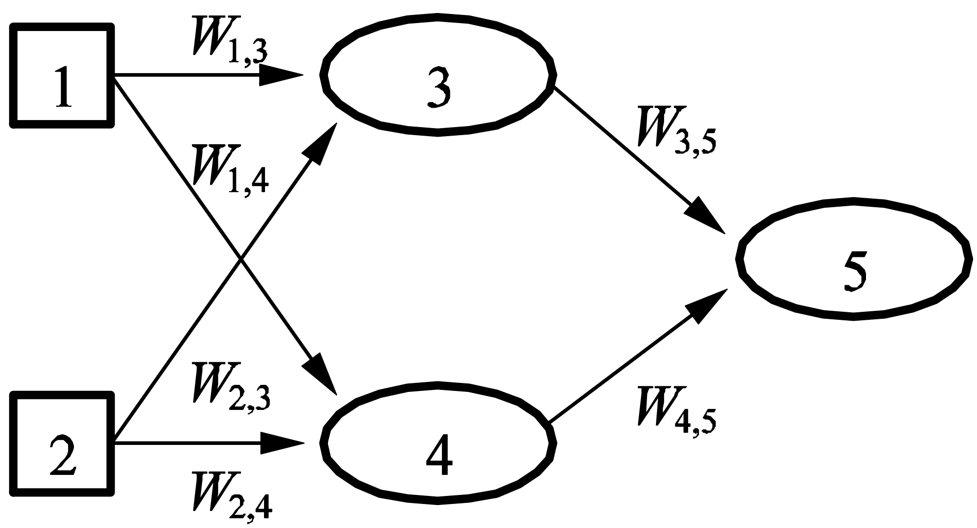


Network Structures

- Feed-forward network
  - Directed acyclic graph
  - No internal state
  - Simply computes outputs from inputs
- Recurrent network
  - Directed cyclic graph
  - Dynamical system with internal states
  - Can memorize information

Feed-forward network

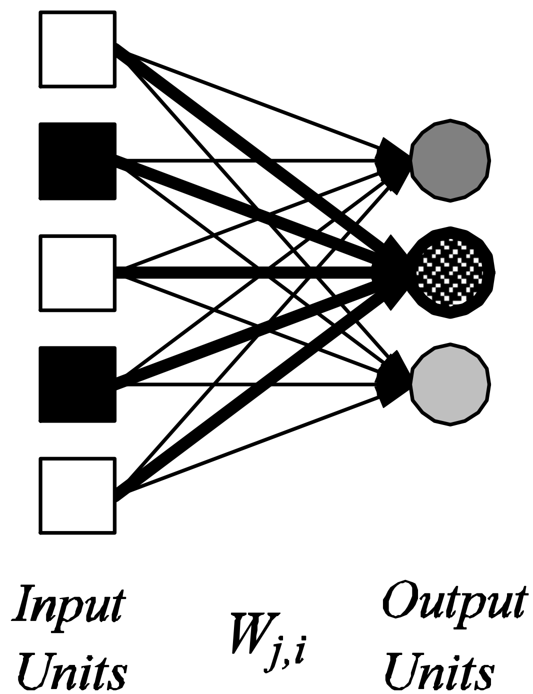
- Simple network with two inputs, one hidden layer of two units, one output unit



$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4)$$
$$= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))$$

Perceptron

- Single layer feed-forward network



Threshold Perceptron Hypothesis Space

- Hypothesis space h_W:
  - All binary classifications with parameters W s.t.

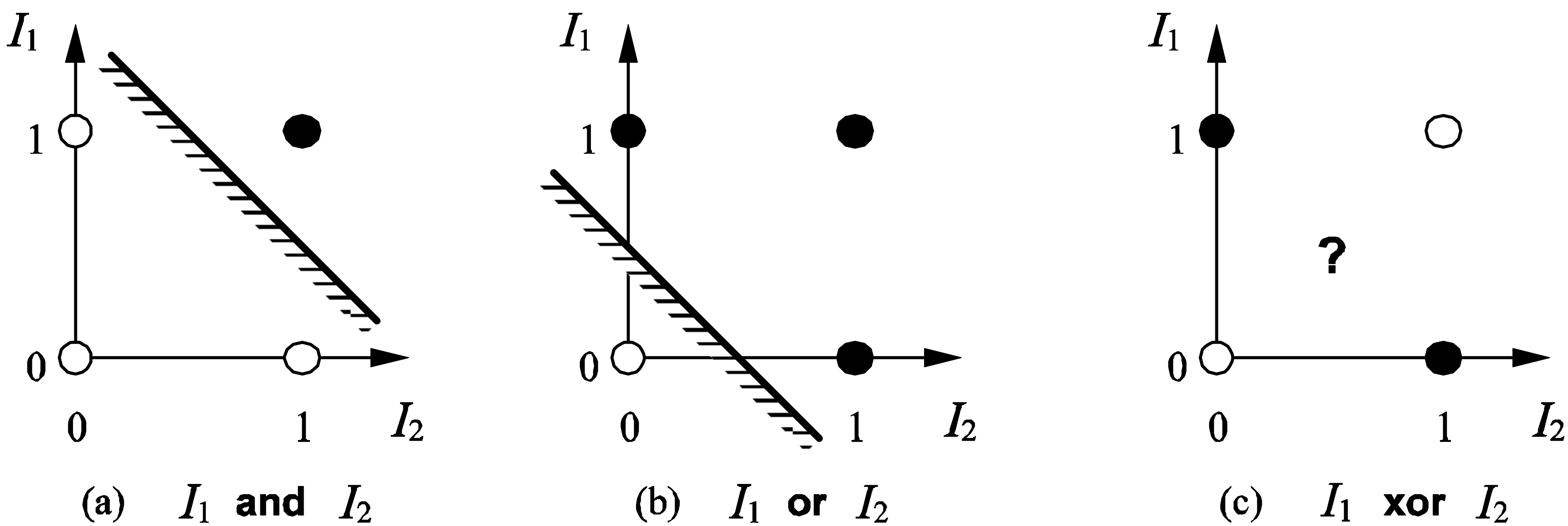
    $a \bullet W \geq 0 \rightarrow 1$

    $a \bullet W < 0 \rightarrow 0$

- Since $a \bullet W$ is linear in W, perceptron is called a linear separator

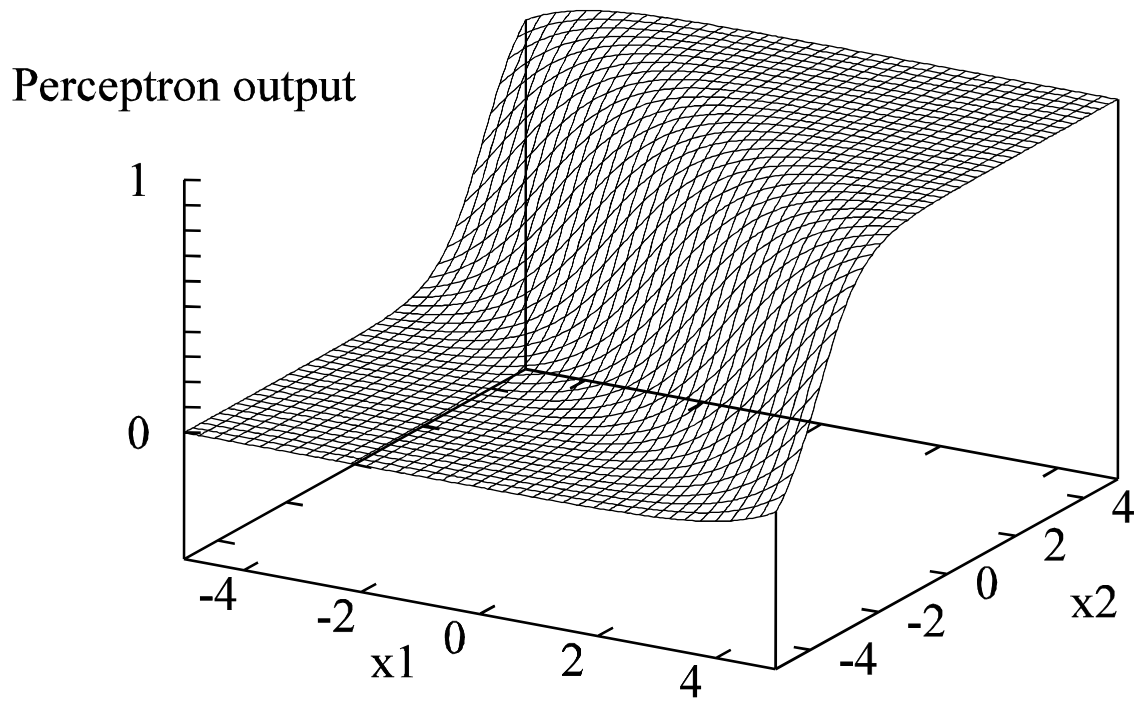
Threshold Perceptron Hypothesis Space

- Are all Boolean gates linearly separable?



Sigmoid Perceptron

- Represent "soft" linear separators



Sigmoid Perceptron Learning

- Formulate learning as an optimization search in weight space
  - Since g differentiable, use gradient descent

- Minimize squared error:

  $E = 0.5\ Err^2 = 0.5\ (y - h_W(\mathbf{x}))^2$

  - $\mathbf{x}$: input
  - y: target output
  - $h_W(\mathbf{x})$: computed output

Perceptron Error Gradient

- $E = 0.5\ Err^2 = 0.5\ (y - h_{\mathbf{W}}(\mathbf{x}))^2$

- $\partial E/\partial W_j = Err\ \partial Err/\partial W_j$
  $= Err\ \partial(y - g(\Sigma_j\ W_j x_j))/\partial W_j$
  $= -Err\ g'(\Sigma_j\ W_j x_j)\ x_j$

- When g is sigmoid fn, then $g' = g(1-g)$

Perceptron Learning Algorithm

- Perceptron-Learning(examples,network)
  - Repeat
    - For each e in examples do

      in $\leftarrow \Sigma_j W_j x_j[e]$

      Err $\leftarrow y[e] - g(in)$

      $W_j \leftarrow W_j + \alpha \text{ Err } g'(in)\, x_j[e]$
  - Until some stopping criteria satisfied
  - Return learnt network

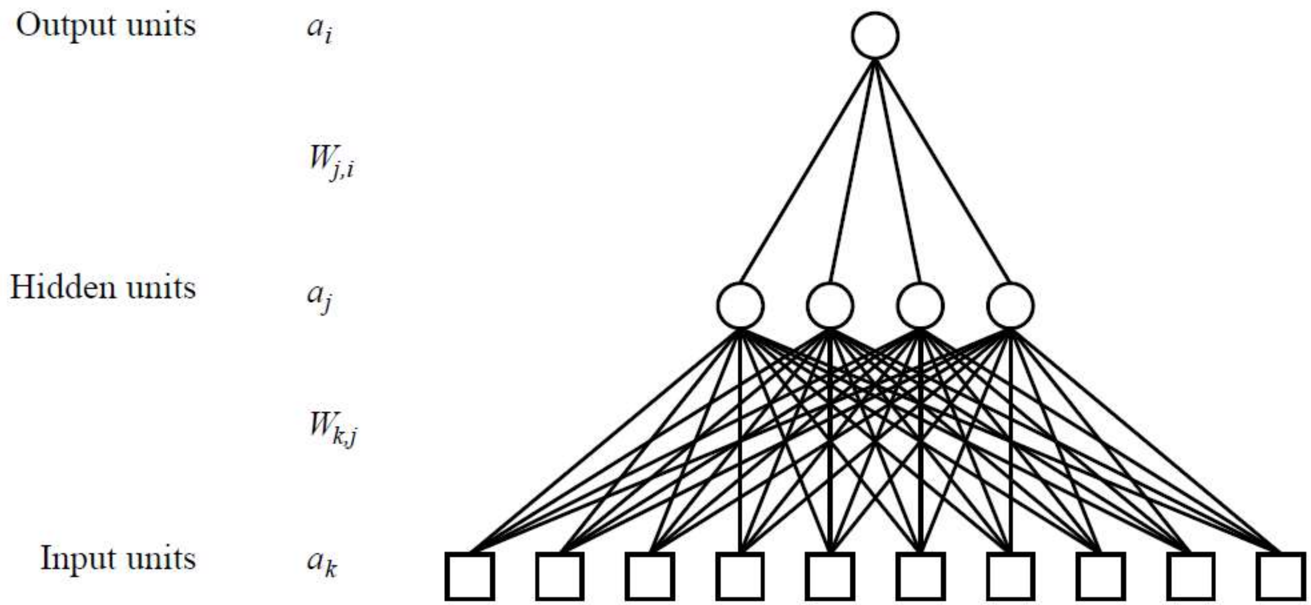
- N.B. α is a learning rate corresponding to the step size in gradient descent

Multilayer Feed-forward Neural Networks

- Perceptron can only represent (soft) linear separators
  - Because single layer

- With multiple layers, what fns can be represented?
  - Virtually any function!

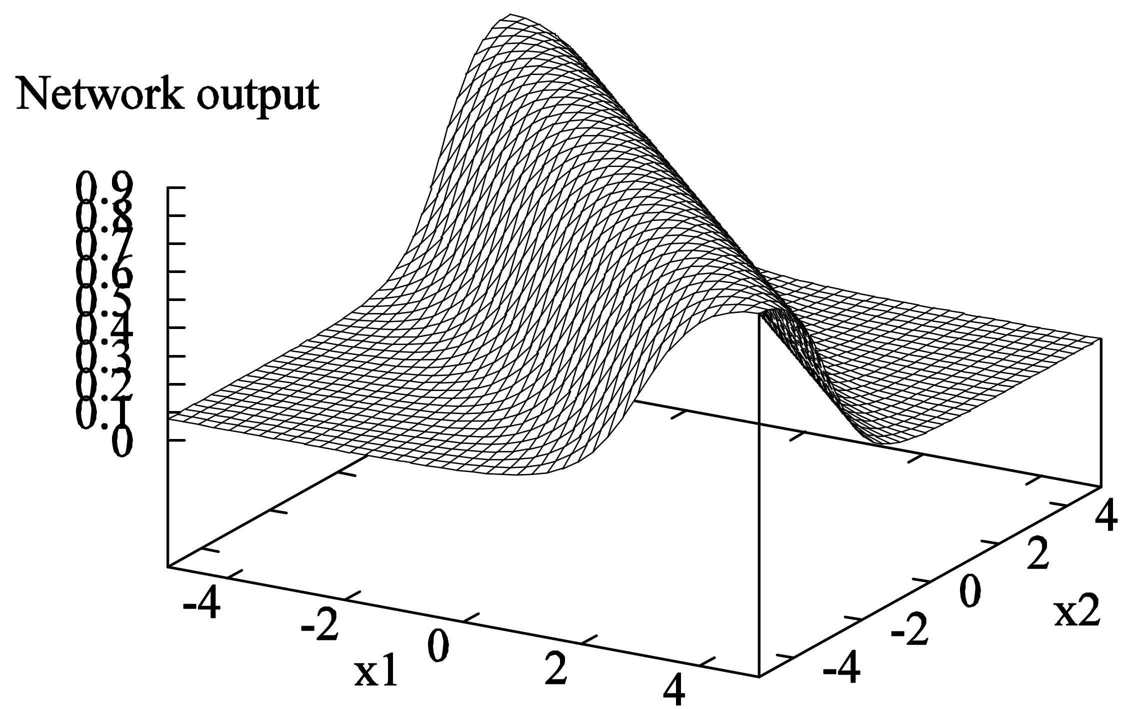
Multilayer Networks



$$a_i = g\left(\sum_j W_{ji}\, g\left(\sum_k W_{kj} a_k\right)\right)$$

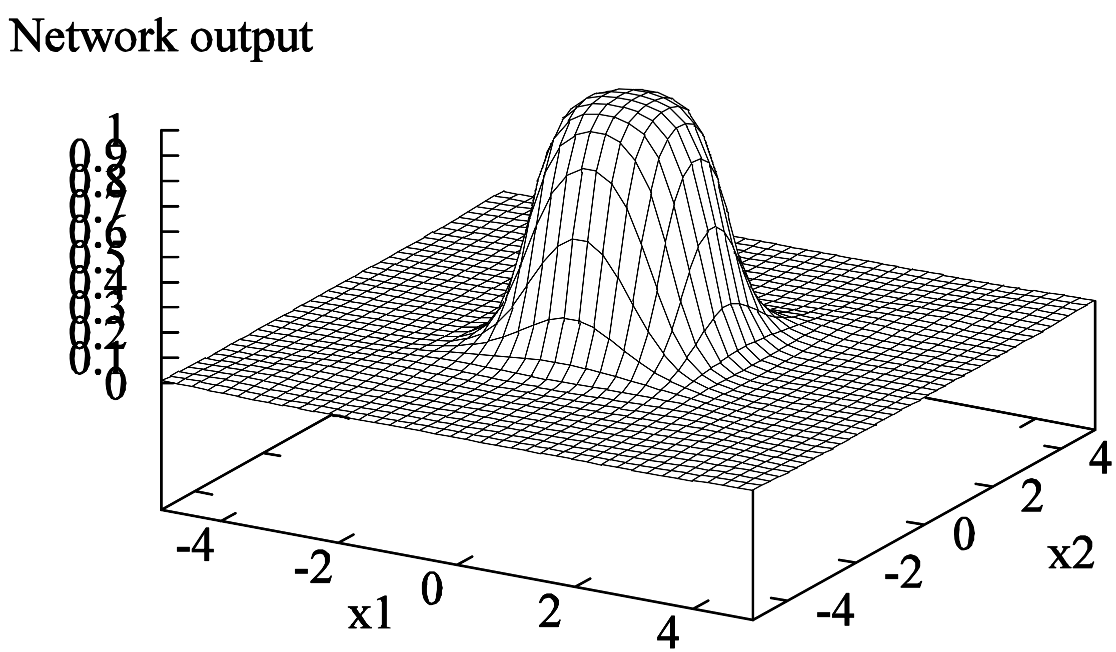
Multilayer Networks

- Adding two sigmoid units with parallel but opposite "cliffs" produces a ridge



Multilayer Networks

- Adding two intersecting ridges (and thresholding) produces a bump



Multilayer Networks

- By tiling bumps of various heights together, we can approximate any function

- **Theorem:** Neural networks with at least one hidden layer of sufficiently many sigmoid units can approximate any function arbitrarily closely.

Common Activation Functions

- Threshold: $h(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

- Sigmoid: $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

- Gaussian: $h(x) = e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

- Hyperbolic tangent: $h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

- Identity: $h(x) = x$

Weight Training

- Parameters: $< \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots >$
- Objectives:
  - **Error minimization**
    - **Backpropagation (aka "backprop")**
  - Maximum likelihood
  - Maximum a posteriori
  - Bayesian learning

Least squared error

- Error function

$$E(\boldsymbol{W}) = \frac{1}{2}\sum_{n} E_n(\boldsymbol{W})^2 = \frac{1}{2}\sum_{n} ||f(\boldsymbol{x_n}, \boldsymbol{W}) - y_n||_2^2$$

where $\boldsymbol{x_n}$ is the input of the n^{th} example

y_n is the label of the n^{th} example

$f(\boldsymbol{x_n}, \boldsymbol{W})$ is the output of the neural net

Sequential Gradient Descent

- For each example $(\boldsymbol{x}_n, y_n)$ adjust the weights as follows:

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E_n}{\partial W_{ji}}$$

- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: **backpropagation algorithm**

Backpropagation

- Back-Prop-Learning(examples,network)
  - Repeat
    - For each example e do
      - Compute output a of each node in **forward** pass
        - » Input nodes: $a_j \leftarrow x_j[e]$
        - » Other nodes: $in_i \leftarrow \sum_j W_{ji} a_j$ and $a_i \leftarrow g(in_i)$
      - Compute modified error Δ of each node in **backward** pass ($l = L$ to 1)
        - » Output nodes: $\Delta_i \leftarrow g'(in_i)\ (y_i[e] - a_i)$
        - » For each node j in layer l: $\Delta_j \leftarrow g'(in_j) \sum_i W_{ji} \Delta_i$
          - » For each node i in layer $l+1$: $W_{ji} \leftarrow W_{ji} + \alpha\ a_j\ \Delta_i$
  - Until some stopping criteria satisfied
  - Return learnt network

Forward phase

- Propagate inputs forward to compute the output of each unit
- Output a_i at unit i:

$$a_i = g(in_i) \quad \text{where} \quad in_i = \sum_j W_{ji} a_j$$

Backward phase

- Use chain rule to recursively compute gradient
  - For each weight W_{ji}: $\frac{\partial E_n}{\partial W_{ji}} = \frac{\partial E_n}{\partial in_i}\frac{\partial in_i}{\partial W_{ji}} = \Delta_i a_j$
  - Let $\Delta_i \equiv \frac{\partial E_n}{\partial in_i}$ then

$$\Delta_i = \begin{cases} g'(in_i)(y_i - a_i) & \text{base case: } i \text{ is an output unit} \\ g'(in_i)\sum_j W_{ji}\Delta_j & \text{recursion: } i \text{ is a hidden unit} \end{cases}$$

  - Since $in_i = \sum_j W_{ji} a_j$ then $\frac{\partial in_i}{\partial W_{ji}} = a_j$

Simple Example

- Consider a network with two layers:
  - Hidden nodes: $g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
    - Tip: $tanh'(x) = 1 - tanh^2(x)$
  - Output node: $g(x) = x$

- Objective: squared error

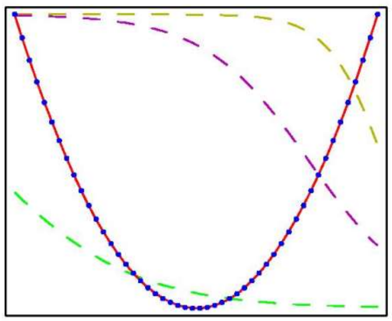
Simple Example

- Forward propagation:
  - Hidden units: $in_j = \sum_k W_{kj} a_k$ $\quad a_j = \tanh(in_j)$
  - Output units: $in_i = \sum_j W_{ji} a_j$ $\quad a_i = \tanh(in_i)$
- Backward propagation:
  - Output units: $\Delta_i = a_i - y_i$
  - Hidden units: $\Delta_j = \left(1 - \tanh^2(in_j)\right) \sum_i W_{ji} \Delta_i$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial W_{kj}} = a_k \Delta_j = a_k \left(1 - \tanh^2(in_j)\right) \sum_i W_{ji} \Delta_i$
  - Output layer: $\frac{\partial E_n}{\partial W_{ji}} = a_j \Delta_i = a_j (a_i - y_i)$

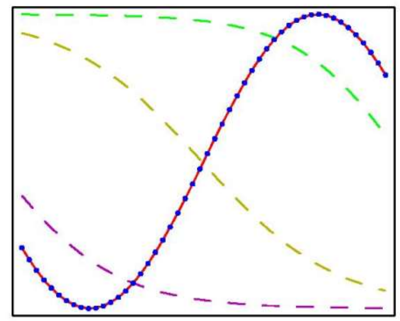
Non-linear regression examples

- Two layer network:
  - 3 tanh hidden units and 1 identity output unit

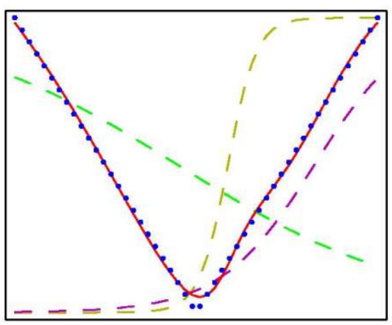
$y = x^2$



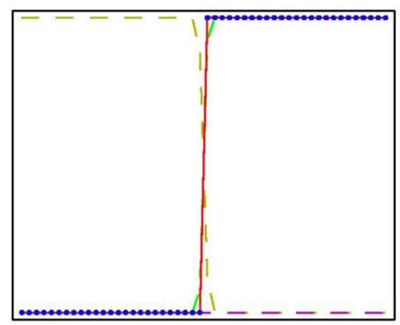
$y = \sin x$



$y = |x|$



$y = \int_{-\infty}^{x} \delta(t)dt$



Analysis

- Efficiency:
  - Fast gradient computation: linear in number of weights
- Convergence:
  - Slow convergence (linear rate)
  - May get trapped in local optima
- Prone to overfitting
  - Solutions: early stopping, regularization (add $||w||_2^2$ penalty term to objective)

Neural Net Applications

- Neural nets can approximate any function, hence 1000's of applications
  - Speech recognition
  - Character recognition
  - Paint-quality inspection
  - Vision-based autonomous driving
  - Etc.