

# Artificial Neural Networks

Introduction to Statistical Machine Learning  
ANU COMP 6467/4670, Sem 1, 2008

Scott Sanner  
NICTA / RSISE  
First.Last@nicta.com.au

# General Function Approximation

- You have a data set  $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}$
- You want to learn  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  from  $\mathbf{D}$ 
  - More precisely, you want to minimize  $\sum_i (\mathbf{f}(\mathbf{x}_i) - y_i)^2$
- You believe  $\mathbf{f}$  is a nonlinear function of  $\mathbf{x}$ 
  - but you don't know much more than that
- Your choices:
  - transform  $\mathbf{x}$  to make  $\mathbf{f}$  linear, use linear regr. / class.
  - use non-linear regression / classification



Why?

# Lecture Outline

- **Linear function approximation**
  - What tools do we have?
  - What are benefits of linear approach?
  - How powerful is it?
- **Non-linear function approximation**
  - When do we use a non-linear model?
  - Artificial Neural Net (ANN) model
  - Training via backpropagation

# Linear vs. Non-linear

- $y = f(\mathbf{x}, \mathbf{w})$ 
  - $\mathbf{x}$  is your input vector
  - $\mathbf{w}$  is your parameter vector (weights)

- **Which  $f$  is linear in  $\mathbf{w}$ ?**

i.e.,  $f(\mathbf{x}, \mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle$  (assume  $\mathbf{x}_0 = 1$ )

–  $f_1(\mathbf{x}) = w_1 x_1 + w_2 x_2$  ✓

–  $f_2(\mathbf{x}) = w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2$  ✓

–  $f_3(\mathbf{x}) = w_1 x_1 + w_2 w_3 x_2 + w_3^2 x_3$  ✗

Any transformation of input  $\mathbf{x}$  maintains linear function in  $\mathbf{w}$ !

# How Powerful are Linear Models?

- **Short answer:**

- $\mathbf{y} = \langle \mathbf{w}, \mathbf{x} \rangle$  is surprisingly powerful

- Especially if  $\mathbf{x}$  transformed:  $\mathbf{x} \rightarrow \Phi(\mathbf{x})$

- $\mathbf{y} = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$

*input*  
space

*feature*  
space

- **Let's investigate, first recall**

- Classification:  $\mathbf{y}$  is discrete

- Regression:  $\mathbf{y}$  is continuous

# Your Linear Fun. Approx. Toolbox

- **Classification**

- Naïve Bayes (simple)
- Logistic Regression (better than NB for dependent features)
- Perceptron (didactic, rarely used in practice)
- SVM and Kernel Methods (very powerful)

- **Regression**

- Linear Regression (closed-form solution)
- SVR and Kernel Methods (very powerful)



Why?

- **Key Advantage**

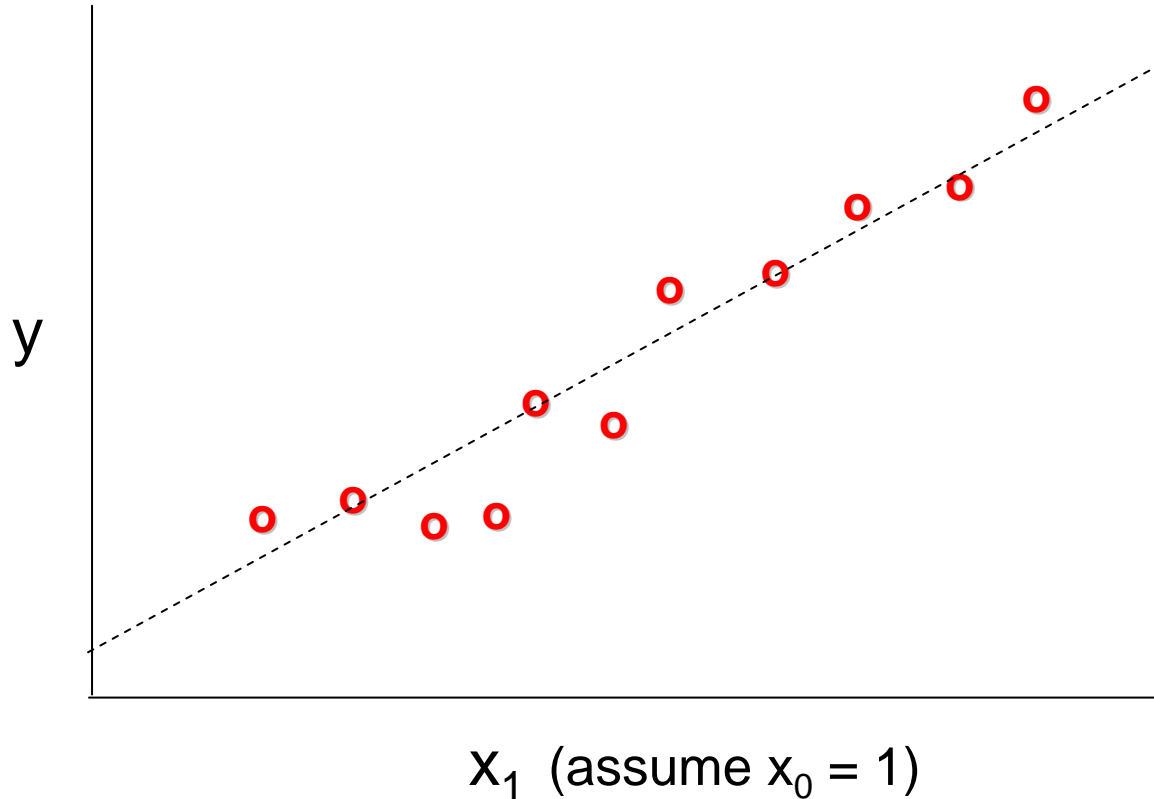
- All of above lead to convex optimization problems  
→ global optima will be found.



Why?

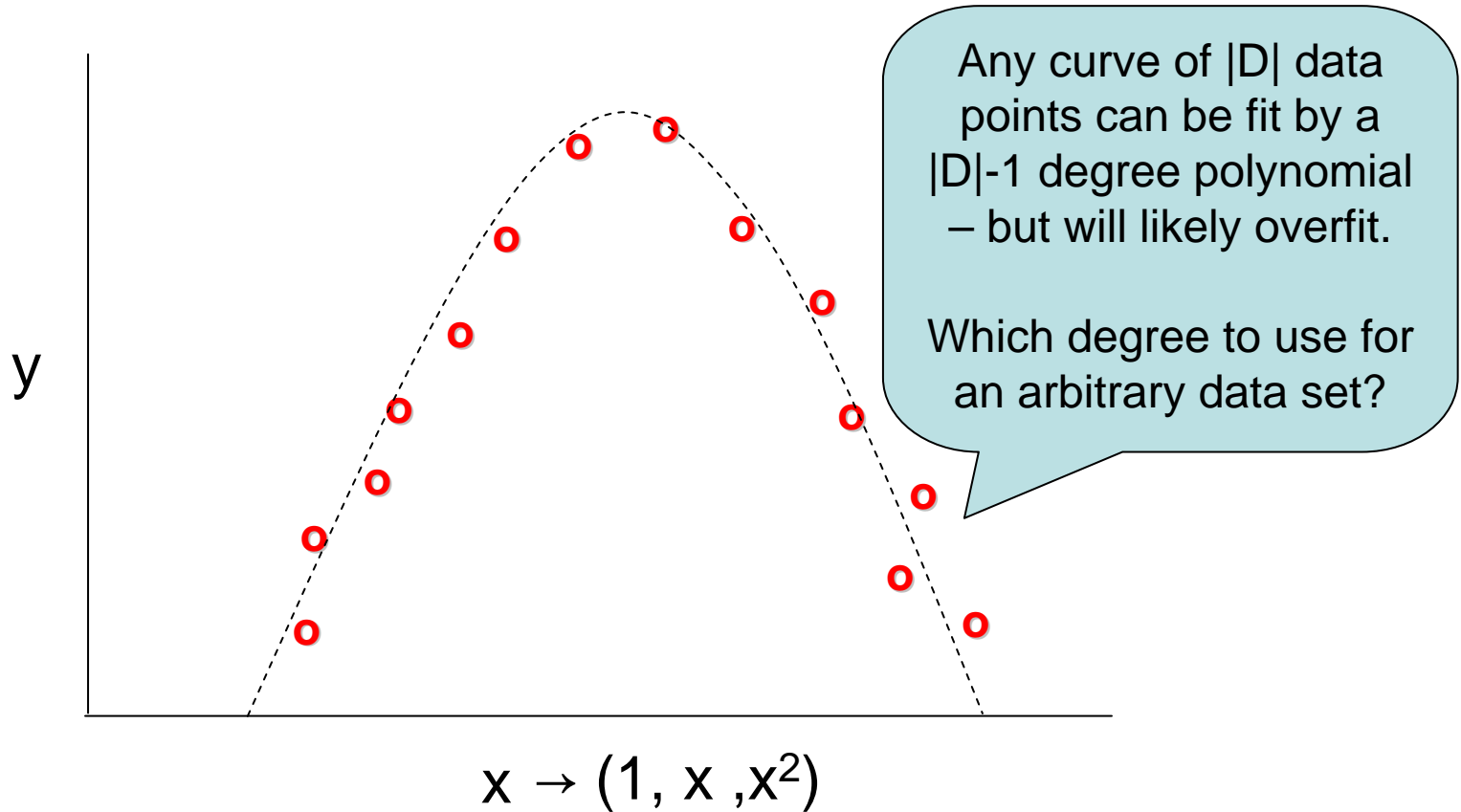
# What can linear functions regress?

- Can a linear model  $\mathbf{y} = \langle \mathbf{w}, \mathbf{x} \rangle$  fit the **o's**?



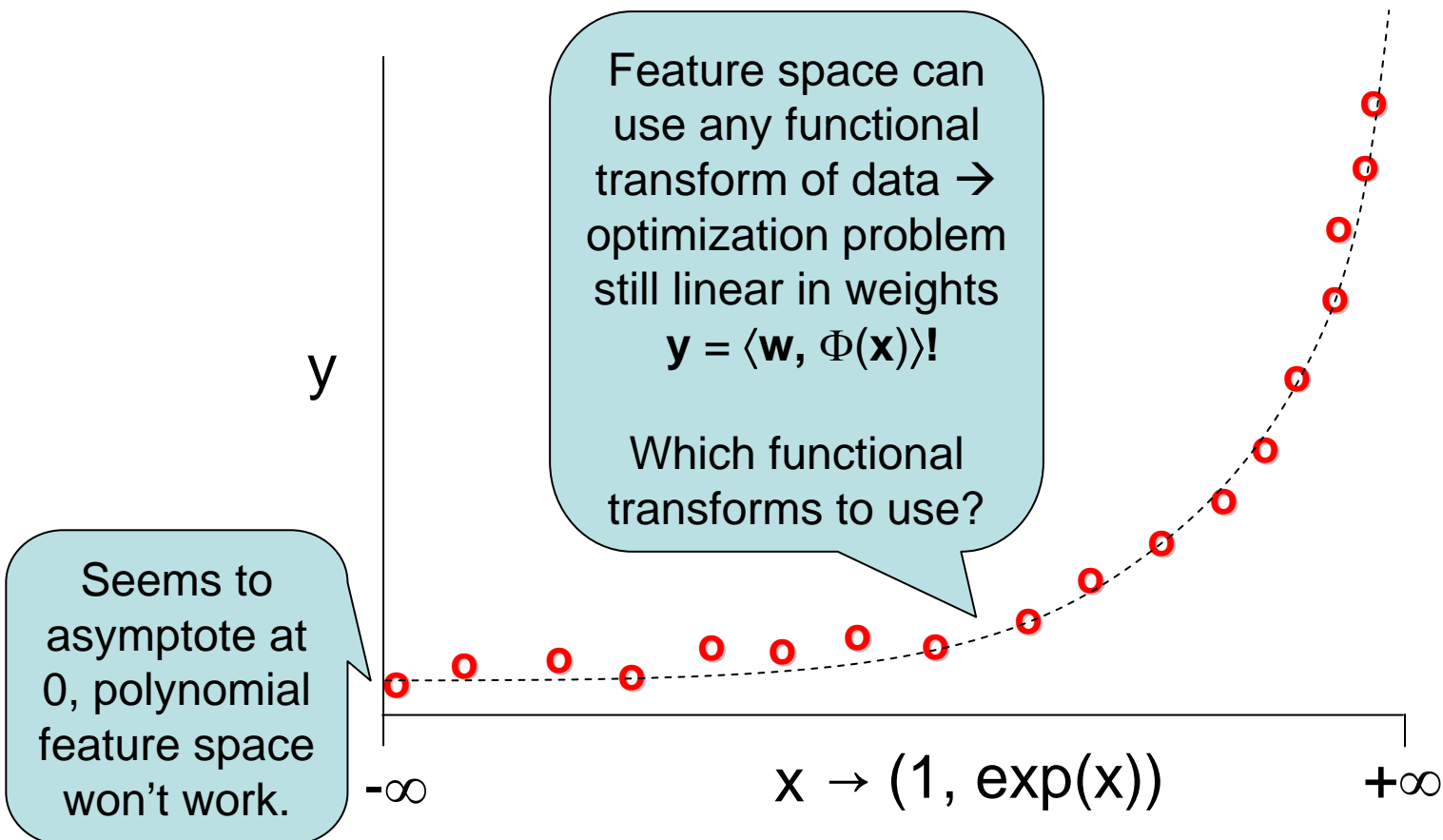
# What can linear functions regress?

- How about this one?



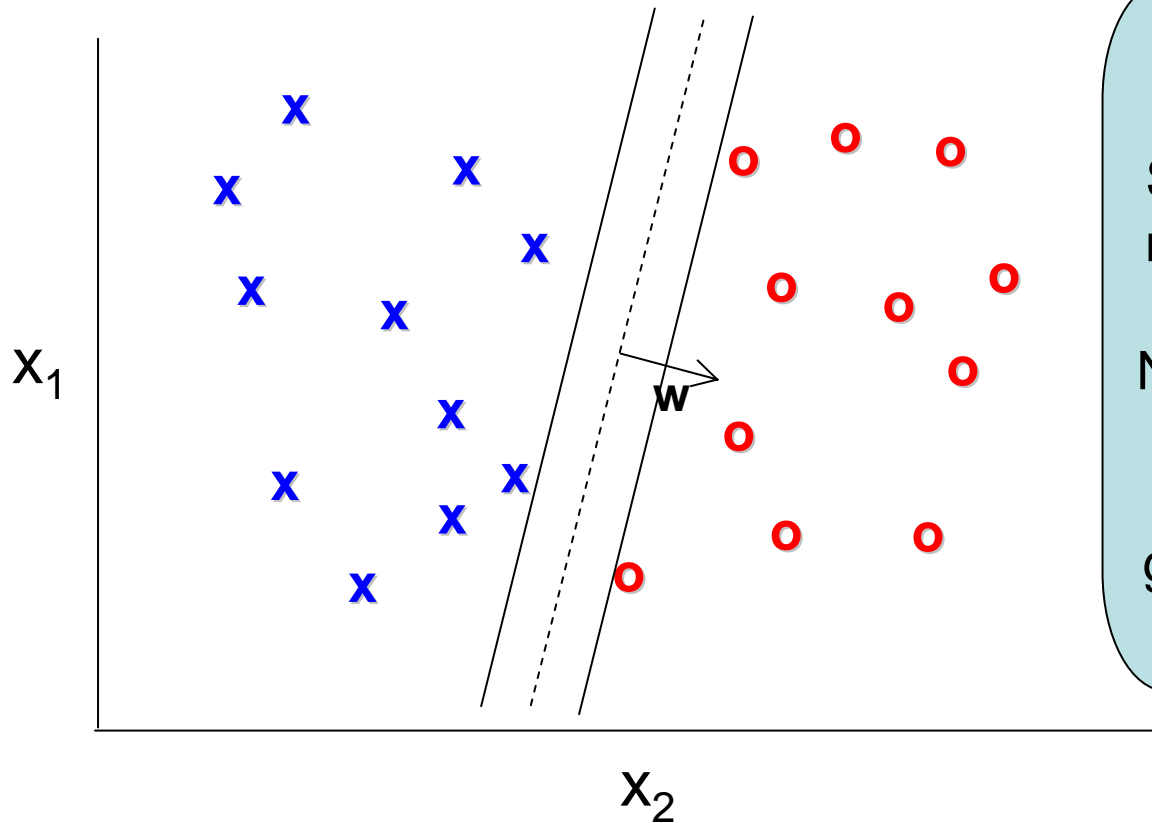
# What can linear functions regress?

- How about this one?



# What can linear functions classify?

- Can a linear model  $\mathbf{y} = \langle \mathbf{w}, \mathbf{x} \rangle$  separate **x** from **o**?
  - Predict **o** if  $\langle \mathbf{w}, \mathbf{x} \rangle > 0$ , otherwise **x**

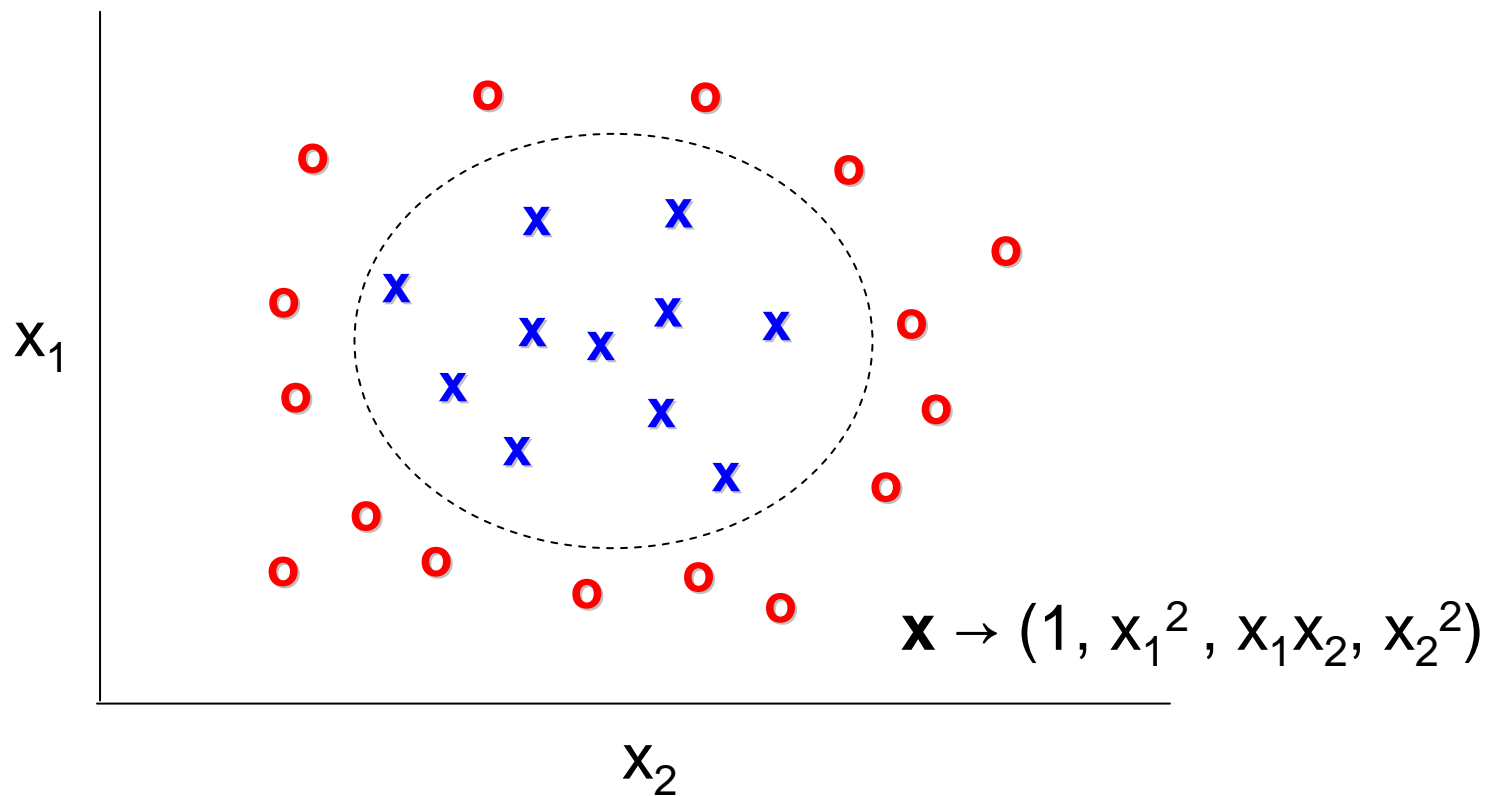


Which separator?  
SVMs give a nice answer.

Naïve Bayes,  
Log Regr.,  
Perceptron  
give different answers.

# What can linear functions learn?

- Can we still separate **x** from **o**?
  - Not linearly separable in input space.

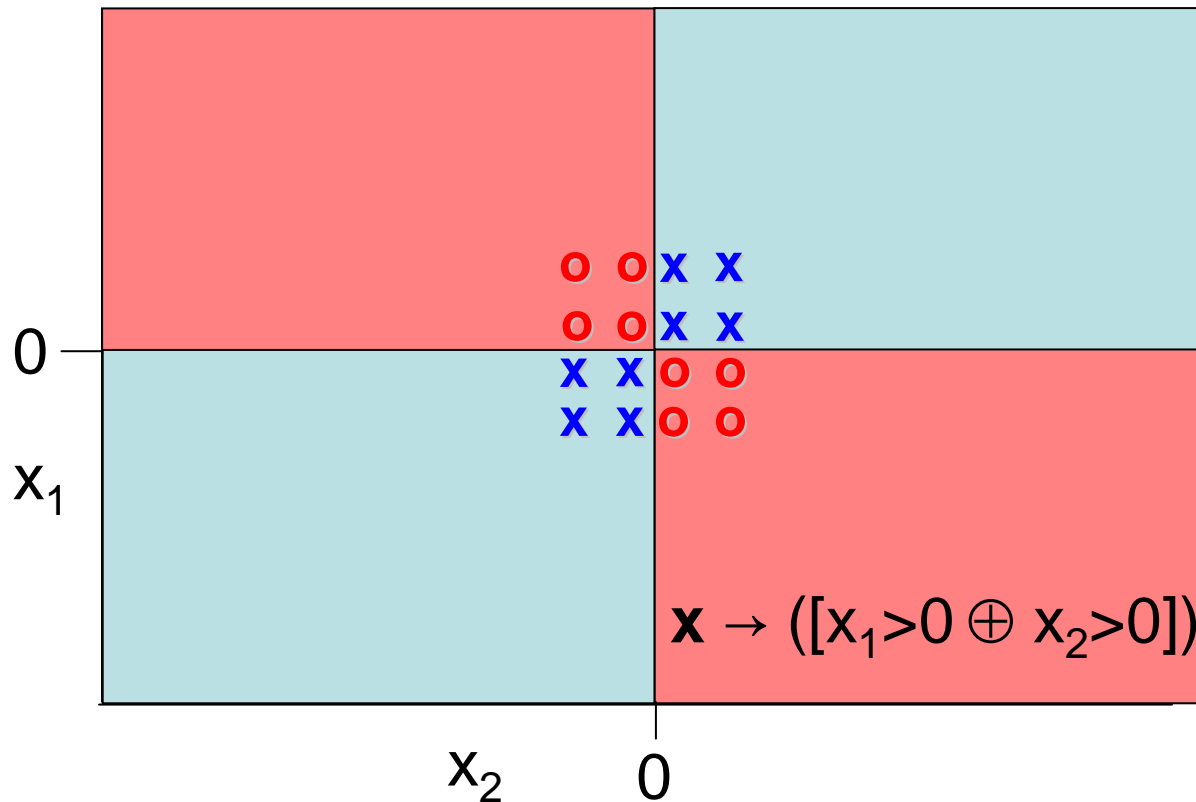


# XOR Problem

- Can we still separate **x** from **o**?
  - Not linearly separable in input space.

Other feature spaces separate too, but with different boundaries.

Can we learn feature space automatically?




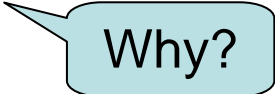
# When to use a Non-linear Model?

- **You have a non-linear model**
  - Model is of scientific / explanatory origin
    - e.g., enzyme kinetics

$$y = \frac{w_1 x}{w_2 + x}$$

- **Could take the log of both sides...**
  - Redefine weights and obtain linear model
  - Why might this be a bad idea?
  - It changes error assumptions in model
    - Minimize  $\sum_i (f(\log x_i) - \log y_i)^2$

# When to use a Non-linear Model?

- **You know model is non-linear in  $x$ ...**
  - But you don't know a good feature space
- ***Could* build complex feature spaces**
  - Add tons of features (polynomials; exp, log, sin, tan; distance between data inputs)  efficient w/ kernel / SVM
- **Back to linear model in  $\Phi(x)$ , but**
  - May be time-consuming due to number of features, CV
  - May lead to overfitting if features inappropriate or too expressive, even with cross validation (CV)!  Why?

# When to use a Non-linear Model?

- **Non-linear model may be more *compact***
  - Use non-linear functions of sub-functions
    - Major feature of ANNs
  - Shared sub-functions → fewer weights
    - May make training faster
    - May mitigate overfitting
- **But optimization may not be convex**
  - No guarantee on optimal solution
  - Must be careful to avoid local optima

Why?

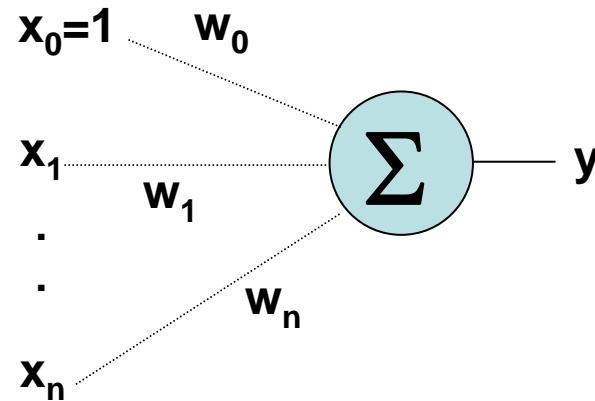
Potential drawbacks

# Remainder of this Lecture

- **We focus on non-linear regression in ANNs**
  - Can be used for classification, use  $y = \pm 1$
- **Why ANNs?**
  - Can approximate range of functions
  - ANNs have yielded impressive practical results
  - More on this at end...
- **But methods apply to many non-linear regression problems...**
  - Just gradient descent optimization using chain rule
  - Common “tricks” to avoid local optima

# Perceptrons

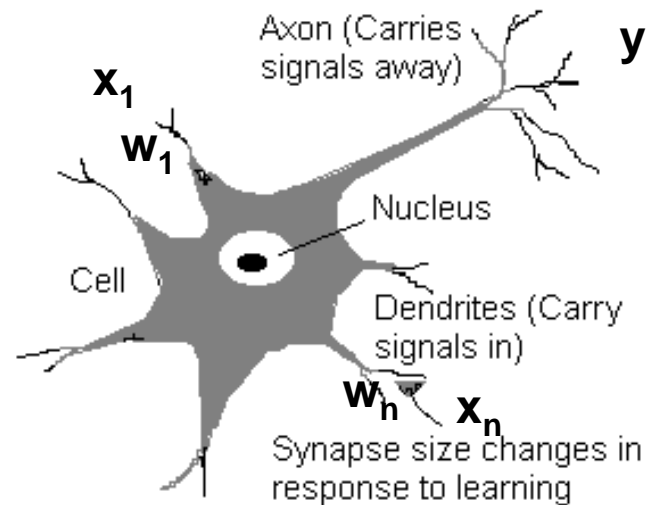
- Graphically represent  $y = \langle \mathbf{w}, \mathbf{x} \rangle$ :



- Threshold output

$$y' = y > 0: y \rightarrow \text{[Step Function]} \rightarrow y'$$

→ Perceptron,  
biologically  
inspired:



# Perceptron Training

- **Training**

- Data  $\mathbf{D} = \{(\mathbf{x}^i, y^i)\}$

- Minimize  $\mathbf{E}(\mathbf{w}) = \sum_i (\langle \mathbf{w}, \mathbf{x}^i \rangle - y^i)^2$

- Follow  $\nabla \mathbf{E}(\mathbf{w})$  to global minima of  $\mathbf{E}(\mathbf{w})$

- **Gradient descent (delta-rule)**

$$\frac{\partial E}{\partial w_j} = \sum_i 2\delta^i x_j^i \quad \text{where} \quad \delta^i = \langle \mathbf{w}, \mathbf{x}^i \rangle - y^i$$

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

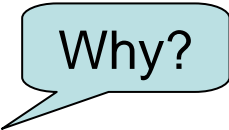
$\eta \sim 0.05$  is learning rate  
**Online:** update for each  $i$   
**Batch:** update for full  $\mathbf{D}$

# Perceptron Training

- **Wait a minute...**

- Minimize  $E(\mathbf{w}) = \sum_i (\langle \mathbf{w}, \mathbf{x}^i \rangle - y^i)^2$
- Global minima at  $\nabla E(\mathbf{w}) = \mathbf{0}$
- Gradient descent (GD) solution = linear regression!

- **Why GD vs. closed-form matrix inversion?**

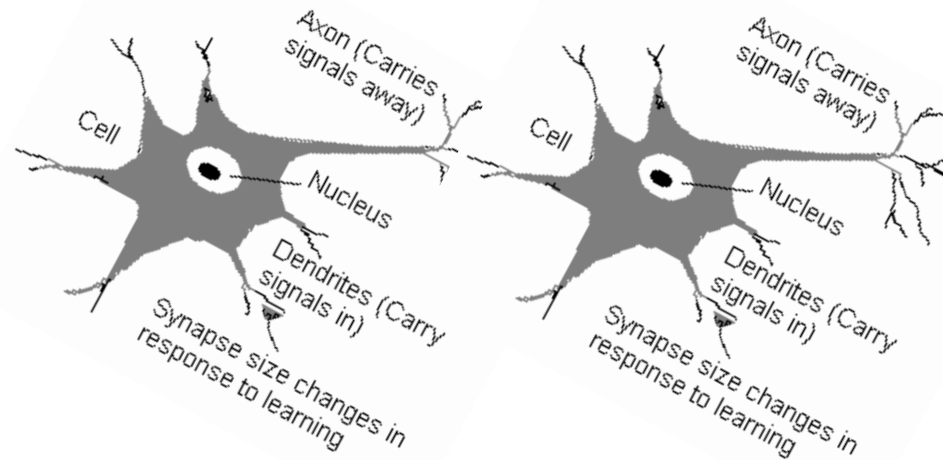
- Biologically plausible?
- Online... data coming fast 
  - Don't need to store data
  - Bounds on classification mistakes until convergence
- If number of weights is large
  - May be faster than matrix inversion
  - May be more numerically stable (but recall pseudoinverse)

- **Works even when there is no closed-form solution**

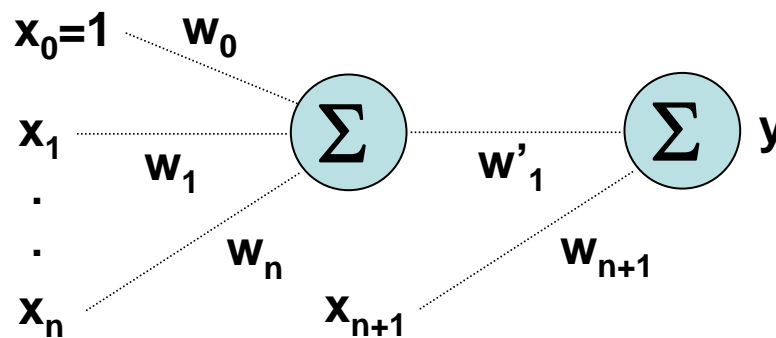
- Useful for generalization to non-linear problems...

# Multilayer Neural Net: An Attempt


- What if we string multiple neurons together?



- Is this interesting? Is it non-linear?

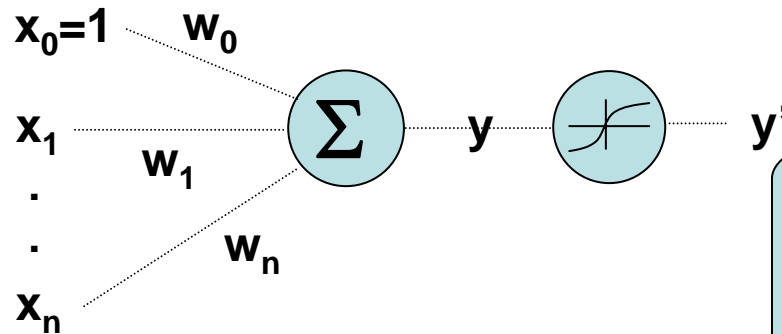


Still linear...  
need non-linear units

like  $y$  —  —  $y'$

# Sigmoid Functions

- Need non-linearity, use  $y' = \sigma(y) = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle)$

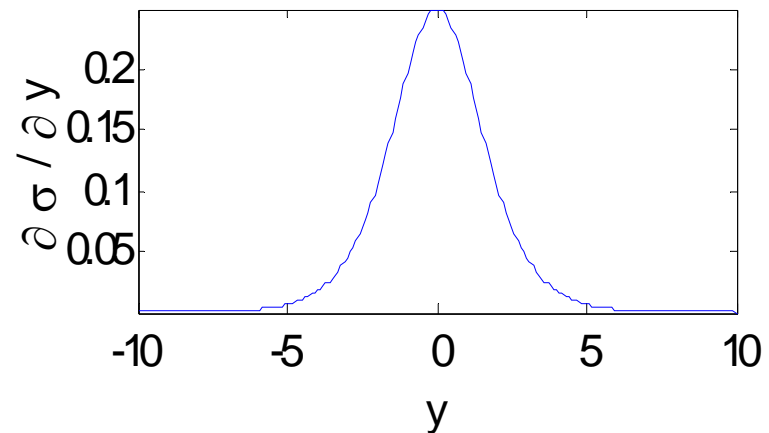
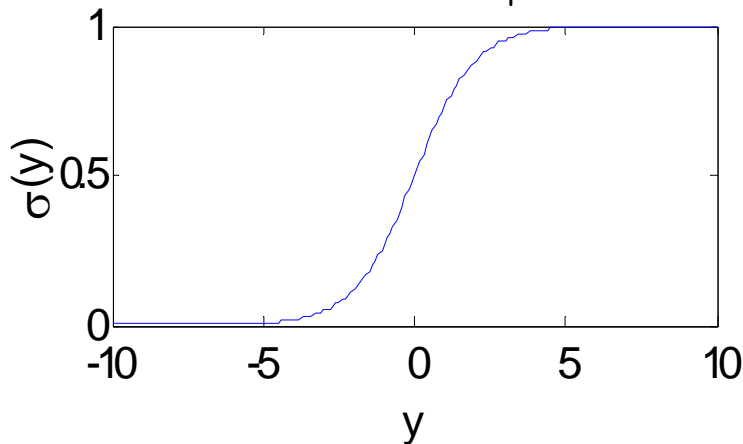


Why sigmoid and not  $y \rightarrow \text{ReLU} \rightarrow y'$ ?

Need continuous differentiability for gradient descent!

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

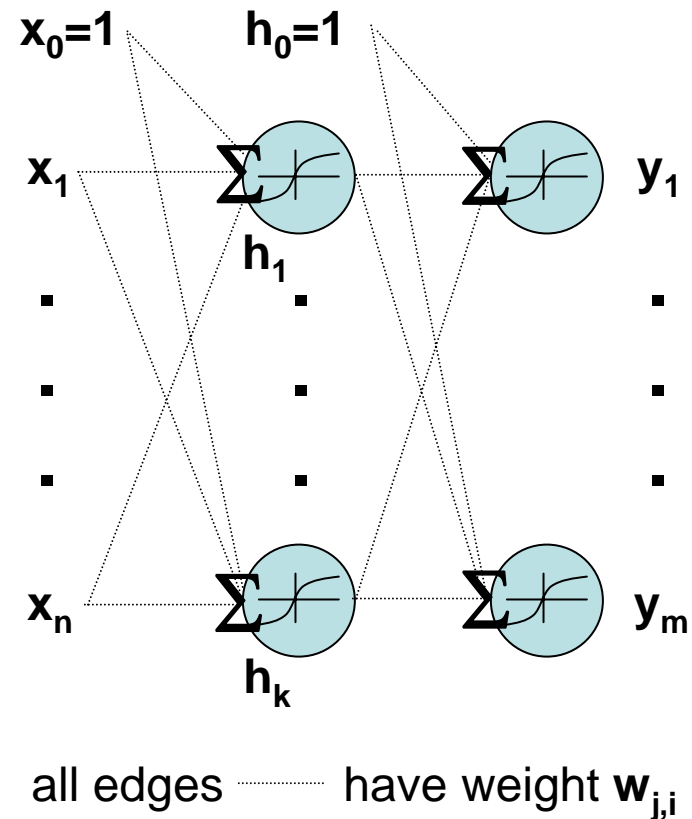
$$\frac{\partial \sigma}{\partial y} = \sigma(y) \cdot (1 - \sigma(y))$$



# ANNs in a Nutshell

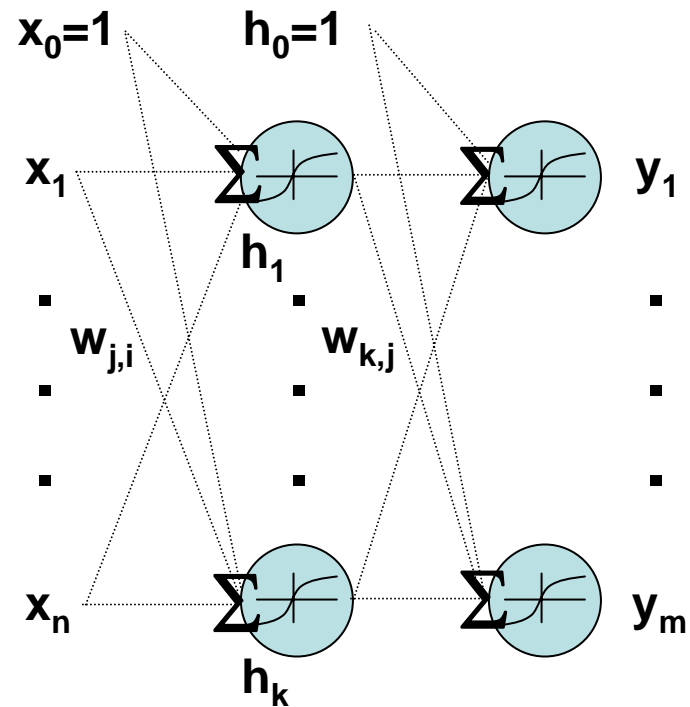
- **Neural Net:**  
non-linear weighted combination of *shared* sub-functions
- **Backpropagation:**  
to minimize SSE,  
train weights using *gradient descent*  
and *chain rule*

Took 20 years to realize b/c focus on biological plausibility!



# Error Backpropagation

- How to compute derivative of error w.r.t. weights?
- Write out error  $E(\mathbf{w})$ , calculate derivative via chain rule
- Helps to cache intermediate network values and errors  $\delta$



$$E(\mathbf{w}) = \sum_k \left\{ \sigma \left[ \sum_j w_{k,j} \sigma \left( \sum_i w_{j,i} x_i \right) \right] - y_k \right\}^2$$

# Backpropagation Derivation

- Gradient for  $w_{kj}$  w.r.t. one data sample:

$$E(\mathbf{w}) = \frac{1}{2} \sum_k \left\{ \sigma \left[ \sum_j w_{k,j} \sigma \left( \underbrace{\sum_i w_{j,i} x_i}_{h_j} \right) \right] - y_k \right\}^2$$

*o<sub>k</sub>* (above the inner sum), *h<sub>j</sub>* (below the inner sum)

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}} = \frac{\partial E(\mathbf{w})}{\partial o_k} \frac{\partial o_k}{\partial w_{kj}}$$

$$= \underbrace{(\sigma(o_k) - y_k) \sigma(o_k) (1 - \sigma(o_k))}_{\delta_k} \frac{\partial}{\partial w_{kj}} \sum_j w_{kj} \sigma(h_j)$$

$$= \delta_k \sigma(h_j)$$

*δ<sub>k</sub>* (under the bracket)

# Backpropagation Derivation

- Gradient for  $w_{ji}$  w.r.t. one data sample:

$$E(\mathbf{w}) = \frac{1}{2} \sum_k \left\{ \sigma \left[ \sum_j w_{k,j} \sigma \left( \underbrace{\sum_i w_{j,i} x_i}_{h_j} \right) \right] - y_k \right\}^2$$

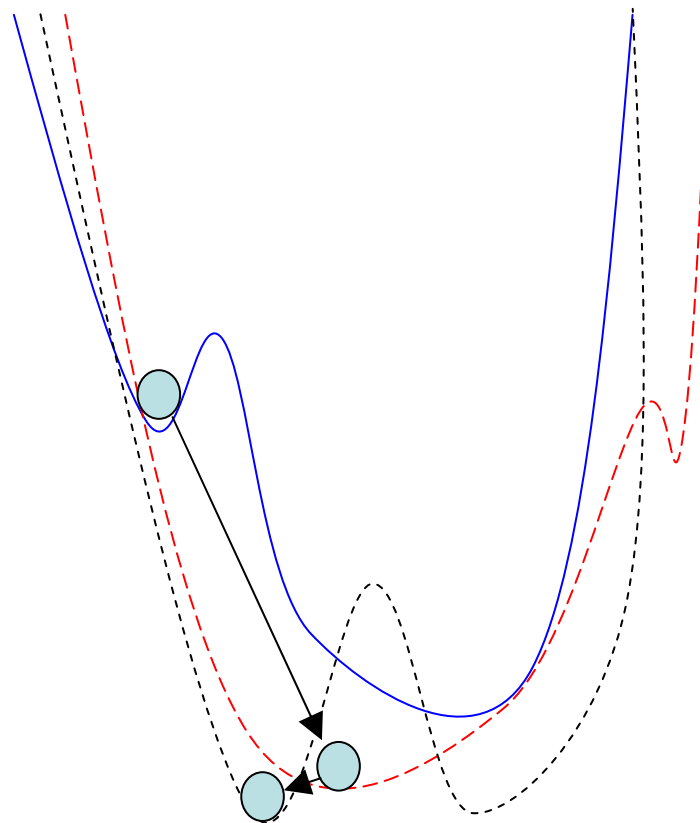
$O_k$

$$\begin{aligned} \frac{\partial E(\mathbf{w})}{\partial w_{ji}} &= \frac{\partial E(\mathbf{w})}{\partial o_k} \frac{\partial o_k}{\partial h_j} \frac{\partial h_j}{\partial w_{ji}} \\ &= \left( \sum_k \delta_k w_{kj} \right) \sigma(h_j) (1 - \sigma(h_j)) \frac{\partial}{\partial w_{ji}} \sum_j w_{ji} \sigma(x_i) \\ &= \underbrace{\delta_j \sigma(x_i)}_{\delta_j} \end{aligned}$$

$h_j$

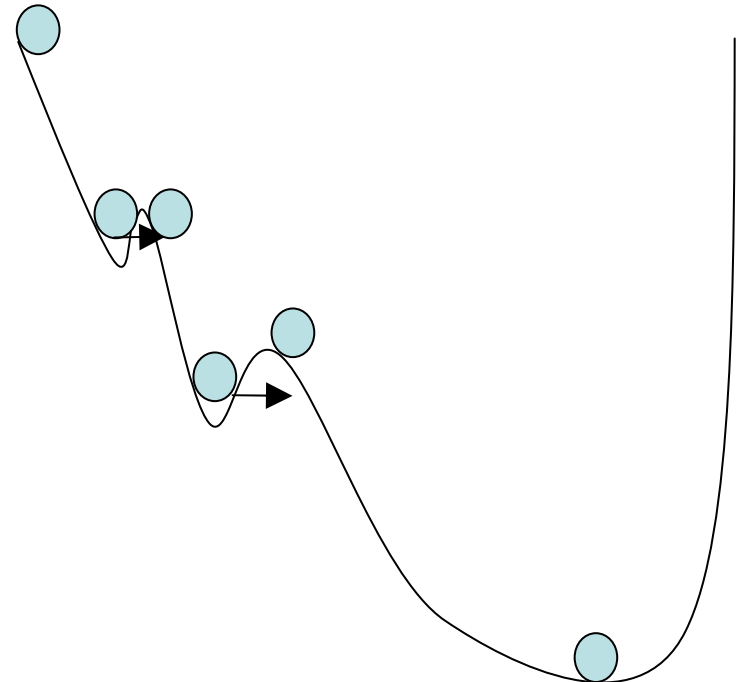
# Stochastic Gradient Descent

- Each subset of data approximates the full error function
- Full error function has local minima
- Each subset usually has different local minima  
→ avoid by only using subsets of data to compute gradient



# Momentum

- Small local minima easy to avoid with a little noise
- But let's be more principled about noise  
→ keep moving in previous direction
- Focus on recent steps:  
$$\Delta \mathbf{w}_t' = \Delta \mathbf{w}_t + \alpha \Delta \mathbf{w}_{t-1}$$
for  $0 < \alpha < 1$



# Backpropagation Algorithm

BACKPROPAGATION(*training\_examples*,  $\eta$ ,  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$ )

Each training example is a pair of the form  $\langle \vec{x}, \vec{t} \rangle$ , where  $\vec{x}$  is the vector of network input values, and  $\vec{t}$  is the vector of target network output values.

$\eta$  is the learning rate (e.g., .05).  $n_{in}$  is the number of network inputs,  $n_{hidden}$  the number of units in the hidden layer, and  $n_{out}$  the number of output units.

The input from unit  $i$  into unit  $j$  is denoted  $x_{ji}$ , and the weight from unit  $i$  to unit  $j$  is denoted  $w_{ji}$ .

- Create a feed-forward network with  $n_{in}$  inputs,  $n_{hidden}$  hidden units, and  $n_{out}$  output units.
- Initialize all network weights to small random numbers (e.g., between  $-.05$  and  $.05$ ).
- 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 and compute the output  $o_u$  of every unit  $u$  in the network.

*Propagate the errors backward through the network:*

2. For each network output unit  $k$ , calculate its error term  $\delta_k$

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \quad (\text{T4.3})$$

3. For each hidden unit  $h$ , calculate its error term  $\delta_h$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{kh} \delta_k \quad (\text{T4.4})$$

4. Update each network weight  $w_{ji}$

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

where

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

$$+ \alpha \Delta \mathbf{w}_{j,i}^{t-1} \text{ (Momentum)}$$

Warning:  
notation slightly  
differs from  
prior derivation,  
but gradients  
are same.

# Representational Capabilities

- **Boolean functions**
  - Every boolean function can be represented exactly network with two layers ( $h, y = \text{sig}$ )
  - Size may be exponential in number of inputs
- **Continuous functions**
  - Can be approximated to arbitrary accuracy with two layers of units ( $h = \text{sigmoid}, y = \text{linear}$ )
- **Arbitrary functions**
  - Any function can be approximated to arbitrary accuracy with three layers ( $h_1 = h_2 = \text{sig}, y = \text{lin}$ )

Linear can't do XOR w/o features.

Why?

# Brief History

- **late-1800's**

- Neural Networks appear as an analogy to biological systems

- **1960's and 70's**

- Perceptron
- XOR problem
- No good algorithms for multilayer nets

- **1986 – Backpropagation**

- Massive explosion of research, applications

# Applications

- **Handwriting recognition**
- **Face recognition**
  - See assignment
- **ALVINN**
  - Learned to steer a car, drove across U.S.
- **TD-Backgammon**
  - Trained a neural net board evaluator via self-play
  - Plays at grandmaster level

# What you Should Know

- **General Machine Learning**
  - Anything marked “Why?”
  - What are benefits of linear & non-linear models?
  - When is it appropriate to use each model?
- **Artificial Neural Networks**
  - What is model?
  - Why are intermediate nodes non-linear?
  - Derivation of backpropagation algorithm
  - Why are stochasticity and momentum useful?
  - How “backpropagation” principles can be used for any non-linear regression problem