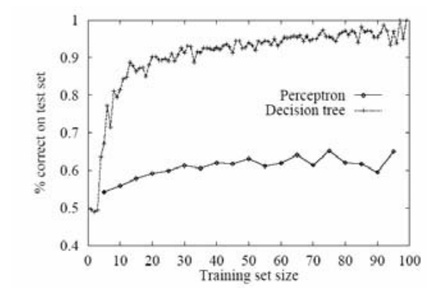
## Foundations of Artificial Intelligence

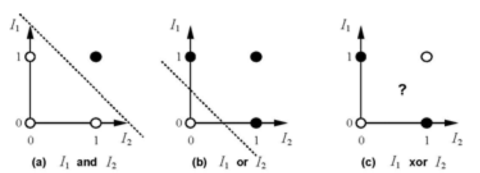
## Neural Networks

**CS472 – Fall 2007**
**Thorsten Joachims**

## Restaurant Data Set




## Limited Expressiveness of Perceptrons

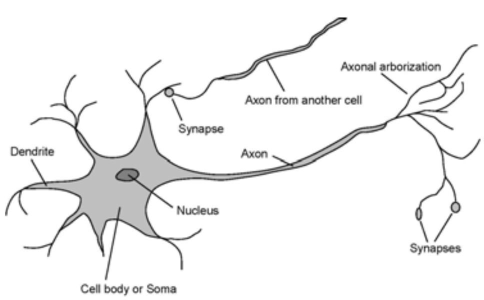



- Minsky and Papert (1969) showed certain simple functions cannot be represented (e.g. Boolean XOR). Killed the field!
- Mid 80th: Non-linear Neural Networks (Rumelhart et al. 1986)

## Neural Networks

- Rich history, starting in the early forties (McCulloch and Pitts 1943).
- Two views:
  - **Modeling the brain**
  - **“Just” representation of complex functions** (Continuous; contrast decision trees)
- Much progress on both fronts.
- Drawn interest from: *Neuroscience, Cognitive science, AI, Physics, Statistics, and CS/EE.*

## Neuron




## Why Neural Nets?

**Motivation:**

Solving problems under the constraints similar to those of the brain may lead to solutions to AI problems that would otherwise be overlooked.

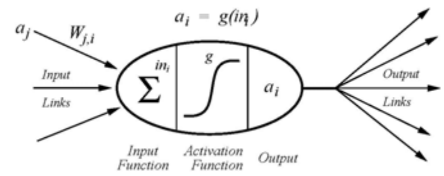
- **Individual neurons operate very slowly**
  *massively parallel algorithms*
- **Neurons are failure-prone devices**
  *distributed representations*
- **Neurons promote approximate matching**
  *less brittle*

## Connectionist Models of Learning

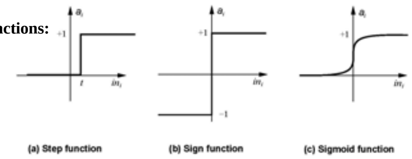
**Characterized by:**

- **A large number of very simple neuron-like processing elements.**
- **A large number of weighted connections between the elements.**
- **Highly parallel, distributed control.**
- **An emphasis on learning internal representations automatically.**

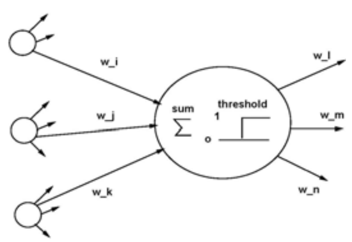
## Artificial Neurons



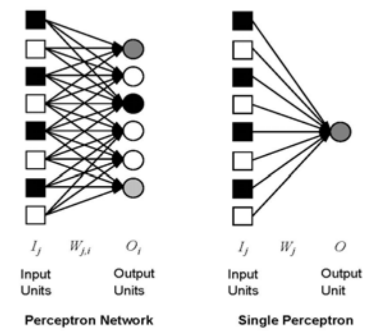

**Activation Functions:**




## Example: Perceptron




## Perceptron Network



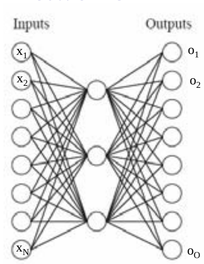

## 2-Layer Feedforward Networks

<u>**Boolean functions**</u>**:**

- **Every boolean function can be represented by network with single hidden layer**
- **But might require exponential (in number of inputs) hidden units**

<u>**Continuous functions**</u>**:**

- **Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]**

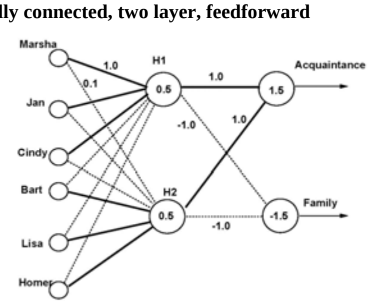



**Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].**

$$o_i = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_j\right)\right)$$

## Multi-Layer Nets

- **Fully connected, two layer, feedforward**



## Backpropagation Training (Overview)

**Training data:**

– $(x_1,y_1),\ldots,(x_n,y_n)$, with target labels $y_z \in \{0,1\}$

**Optimization Problem (single output neuron):**

– Variables: network weights $w_{i\rightarrow j}$

– Obj.: $E=\min_w \sum_{z=1..n}(y_z - o(x_z))^2$, $o_i(x_z) = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_{z,j}\right)\right)$

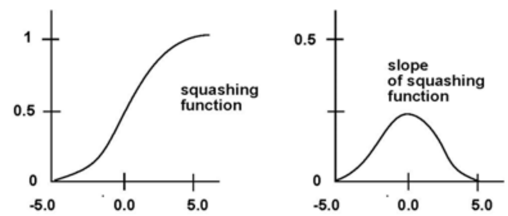
– Constraints: none

**Algorithm: local search via gradient descent.**

- Randomly initialize weights.
- Until performance is satisfactory*,
  – Compute partial derivatives ($\partial E / \partial w_{i\rightarrow j}$) of objective function E for each weight $w_{i\rightarrow j}$
  – Update each weight by $w_{i\rightarrow j} \leftarrow w_{i\rightarrow j} + \alpha\,(\partial E / \partial w_{i\rightarrow j})$

## Smooth and Differentiable Threshold Function

- Replace sign function by a differentiable activation function
  → sigmoid function: $g(x) = \frac{1}{1+e^{-x}}$




## Slope of Sigmoid Function

$$f(x) = \frac{1}{1+e^{-x}}$$

Slope: $\frac{df(x)}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$

$= (1+e^{-x})^{-2}\, e^{-x}$

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

$= f(x)\frac{e^{-x}}{(1+e^{-x})}$

$= f(x)(1-f(x))$

View in terms of output at node:

$= o_j(1-o_j)$

## Backpropagation Training (Detail)

- Input: training data $(\mathbf{x_1},\mathbf{y_1}),\ldots,(\mathbf{x_n},\mathbf{y_n})$, learning rate parameter $\alpha$.
- Initialize weights.
- Until performance is satisfactory
  – **For each training instance,**
    - **Compute the resulting output**
    - **Compute $\beta_z = (y_z - o_z)$ for nodes in the output layer**
    - **Compute $\beta_j = \sum_k w_{j\rightarrow k}\, o_k\, (1-o_k)\, \beta_k$ for all other nodes.**
    - **Compute weight changes for all weights using**

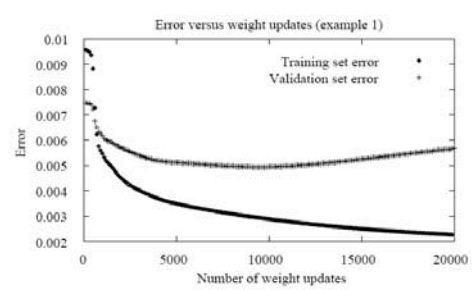
      $\Delta w_{i\rightarrow j}(l) = o_i\, o_j\, (1-o_j)\, \beta_j$
  – **Add up weight changes for all training instances, and update the weights accordingly.**

    $w_{i\rightarrow j} \leftarrow w_{i\rightarrow j} + \alpha \sum_l \Delta w_{i\rightarrow j}(l)$

## Hidden Units

- Hidden units are nodes that are situated between the input nodes and the output nodes.
- Hidden units allow a network to learn non-linear functions.
- Hidden units allow the network to represent combinations of the input features.
- Given too many hidden units, a neural net will simply memorize the input patterns (overfitting).
- Given too few hidden units, the network may not be able to represent all of the necessary generalizations (underfitting).

## How long should you train the net?



## How long should you train the net?

- **The goal is to achieve a balance between correct responses for the training patterns and correct responses for new patterns. (That is, a balance between memorization and generalization).**
- **If you train the net for too long, then you run the risk of overfitting.**
- **Select number of training iterations via cross-validation on a holdout set.**

## Design Decisions

- Choice of learning rate $\alpha$
- Stopping criterion – when should training stop?
- Network architecture
  - How many hidden layers? How many hidden units per layer?
  - How should the units be connected? (Fully? Partial? Use domain knowledge?)
- How many restarts (local optima) of search to find good optimum of objective function?