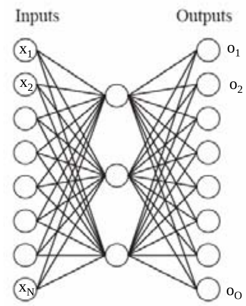
2-Layer Feedforward Networks



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

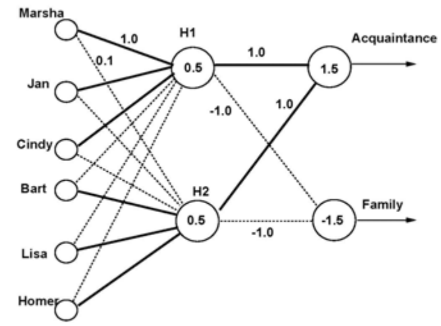
Continuous functions:
- **Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]**

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)

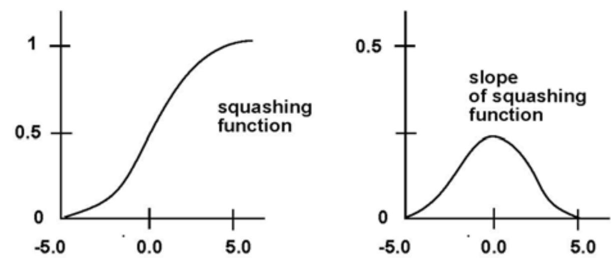
Training data:
- $(x_1,y_1),\ldots,(x_n,y_n)$, with target labels $y_z \in \{0,1\}$

Optimization Problem (single output neuron):
- Variables: network weights $w_{i\to j}$
- Objective fct: $\min_w \sum_{z=1..n} (y_z - o_z)^2$, $\quad o_i = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_j\right)\right)$
- Constraints: none

Algorithm: local search via gradient descent.
- Randomly initialize weights.
- Until performance is satisfactory*,
  - **Present all training instances. For each one,**
    - Calculate actual output. (forward pass)
    - Compute the weight changes that move the output o closer to the desired label y. (backward pass)
  - **Add up weight changes and change the weights.**

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 α.
- 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 $\beta_j = \sum_k w_{j\to k}\, o_k\, (1-o_k)\, \beta_k$ for all other nodes.**
    - **Compute weight changes for all weights using**

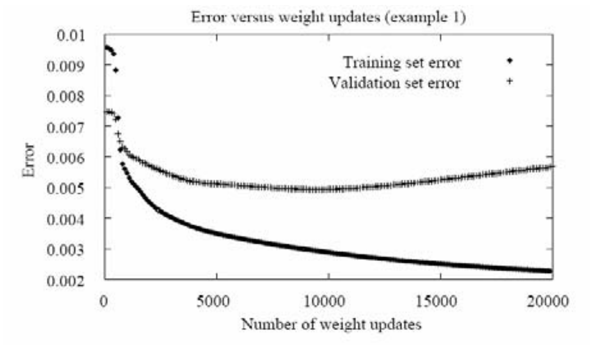
      $$\Delta w_{i\to j}(l) = o_i\, o_j\, (1-o_j)\, \beta_j$$
  - **Add up weight changes for all training instances, and update the weights accordingly.**

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

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.**
- **In general, the network is trained until it reaches an acceptable error rate (e.g. 95%).**

Design Decisions

- Choice of learning rate *r*
- 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?