Limited Expressiveness of Perceptrons

- Minsky and Papert (1969) showed certain simple functions cannot be represented (e.g. Boolean XOR). Killed the field!
- Mid 80s: 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.

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
- Neurons are failure-prone devices
- 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

\[ a_i = g(\sum W_{ij} x_j + \theta) \]

Activation Functions:

- Step function
- Linear function
- Sigmoid function

Example: Perceptron

Perceptron Network

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].

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 \(y_i \in \{0, 1\}\)

Optimization Problem (single output neuron):
- Variables: network weights \(w_{ij}\)
- Obj.: \(E = \sum_{i=1}^{n} (y_i - \sigma(x_i))^2\), \(\sigma(x) = \sum_{j} w_{ij} \sigma(x_j)\)
- Constraints: none

Algorithm: local search via gradient descent.
- Randomly initialize weights.
- Until performance is satisfactory*,
  - Compute partial derivatives \(\frac{\partial E}{\partial w_{ij}}\) of objective function \(E\) for each weight \(w_{ij}\)
  - Update each weight by \(w_{ij} \leftarrow w_{ij} + \alpha \frac{\partial E}{\partial w_{ij}}\)

Smooth and Differentiable Threshold Function
- Replace sign function by a differentiable activation function
  - sigmoid function: \(g(x) = \frac{1}{1+e^{-x}}\)

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?
- Error versus weight updates (example 1)
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?