

Neural Network Introduction

CPS 271
Ron Parr

Why Neural Networks?

- Maybe we should make our computers more brain-like:

	Computers	Brains
Computational Units	10^8 gates/CPU	10^{11} neurons
Storage Units	10^{10} bits RAM 10^{12} bits HD	10^{11} neurons 10^{14} synapses
Cycle Time	10^{-9} S	10^{-3} S
Bandwidth	10^{10} bits/s*	10^{14} bits/s
Compute Power	10^9 Ops/s	10^{14} Ops/s

Neural Network Motivation

- Individual neurons are slow, boring
- Brains succeed by using massive parallelism
- Copy what works
- Raises many issues:
 - Is the computational metaphor suited to the computational hardware?
 - How do we know if we are copying the important part?
 - Are we aiming too low?



Artificial Neural Networks

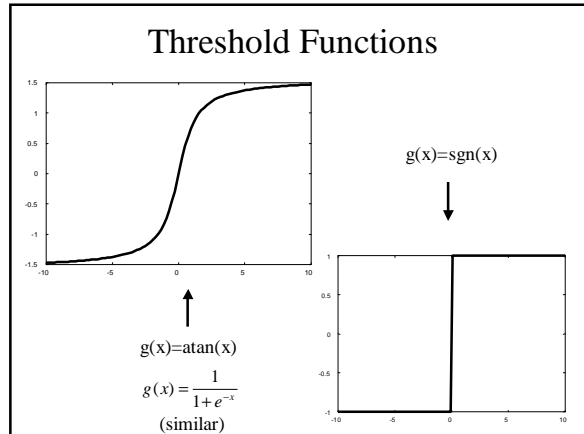
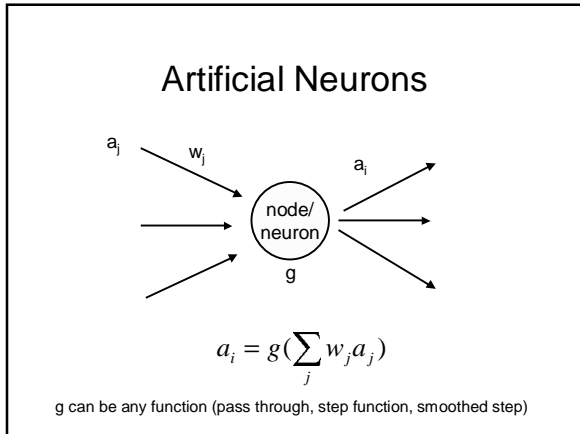
- Develop *abstraction* of function of actual neurons
- Simulate large, massively parallel artificial neural networks on conventional computers
- Some have tried to build the hardware too
- Try to approximate human learning, robustness to noise, robustness to damage, etc.

Use of neural networks

- Trained to pronounce English
 - Training set: Sliding window over text, sounds
 - 95% accuracy on training set
 - 78% accuracy on test set
- Trained to recognize handwritten digits
 - >99% accuracy
- Trained to drive
(Pomerleau's no-hands across America)

Neural Network Lore

- Neural nets have been adopted with an almost religious fervor within the AI community several times
- They are often ascribed near magical powers by people, usually those who know the least about computation or brains
- For most AI people, the magic is gone, but neural nets remain extremely interesting and useful mathematical objects



- ### Network Architectures
- Cyclic vs. Acyclic
 - Cyclic is tricky, but more biologically plausible
 - Hard to analyze in general
 - May not be stable
 - Need to assume latches to avoid race conditions
 - Hopfield nets: special type of cyclic net useful for associative memory
 - Single layer (perceptron)
 - Multiple layer

- ### Feedforward Networks
- We consider acyclic networks
 - One or more computational layers
 - Entire network viewed as computing a complex non-linear function
 - Typical uses in learning:
 - Classification
 - Function approximation

- ### Single Layer Function Approximation
- g function is a pass through
 - Output is weighted combination of inputs
 - Goal: Minimize squared error
 - But this is just regression!
 - Find least squares fit
 - Orthogonal projection

- ### Why is single layer case interesting?
- Relates biologically plausible structure to mathematical procedure
 - Is the computation to determine the weights plausible?
 - Idea: Break down procedure into an iteration that can be done by a "simple" neuron

Solving by Gradient Descent

$$E = 0.5 \sum_i \text{error}(x_i, w)^2$$

$$= 0.5 \sum_i \left(\sum_k w_k x_{ik} - t_k \right)^2$$

$$\frac{\partial E}{\partial w_j} = \sum_i \left(\sum_k w_k x_{ik} - t_k \right) x_{ij}$$

"Learning Rate" \rightarrow $\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$

$$w_j \leftarrow w_j + \Delta w_j$$

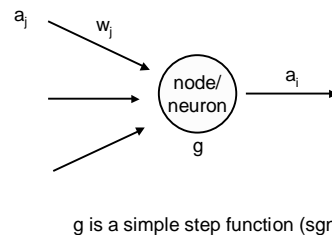
Summary

- Each "neuron" must know:
 - Difference between output and target
 - Weights
- Updates done in batch
- Gradient descent to global optimum
- Robbins-Monro approach
 - Decreasing step sizes prevent oscillation
 - On-line updates possible

Classification

- Classification and regression very similar
- Linear discriminants exist for special cases of normal distributions (approximation for others)
- Issues
 - Activation function
 - Training Algorithm

Perceptron



Perceptron Learning

- We are given a set of inputs $X_1 \dots X_n$
- $T_1 \dots T_n$ is a set of target outputs (boolean)
- w is our set of weights
- $\text{net}(X_i, w)$ = output of perceptron
 - input X_i
 - weights w
- $\text{error}(X_i, w) = T_i - \text{net}(X_i, w)$
- Goal: Pick w to optimize:

$$\min_w \sum_i \text{error}(X_i, w)$$

Update Rule

Repeat until convergence:

$$\forall i \forall j : w_j \leftarrow w_j + \eta x_{i,j} \text{error}(x_i, w)$$

↑
"Learning Rate"

- i iterates over samples
- j iterates over weights

<http://neuron.eng.wayne.edu/java/Perceptron/New38.html>

Compare with Least Squares

Perceptron:

$$w_j \leftarrow w_j + \eta x_{i,j} \text{error}(x_i, w)$$

Least Squares

$$w_j \leftarrow w_j - \eta x_{i,j} \sum_k w_k x_{ik} - t_k$$

$$w_j \leftarrow w_j - \eta x_{i,j} \text{error}(x_i, w)$$

Perceptron Learning Properties

- Good news:
 - If there exists a set of weights that will correctly classify every example, the perceptron learning rule will find it
 - Does not depend on step size
- Bad news:
 - Perceptrons can represent only a small class of functions, "linearly separable," functions
 - May oscillate if not separable

Linearly Separable Functions

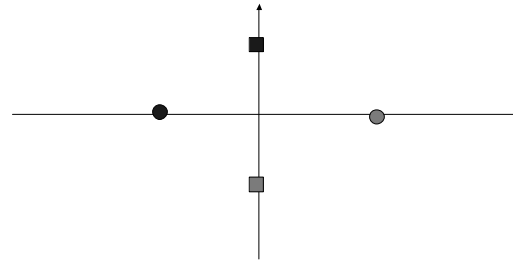
What is a perceptron really doing?

It checks if a linear combination of the inputs is greater than a threshold.

$$w_1 x_1 + w_2 x_2 \dots w_n x_n > 0?$$

Perceptron asks: On what side of a hyperplane does x lie?

Visualizing Linearly Separable Functions



Is red linearly separable from green?
Are the circles linearly separable from the squares?

Observations

- Linear separability is fairly weak
- We have other tricks:
 - Functions that are not linearly separable in one space, may be linearly separable in another space
 - If we twiddle our inputs to our neural network, then we change the space in which we are constructing linear separators
 - Every function has a linear separator (in some space)
- Perhaps other network architectures will help