


Neural Networks

CPS 170

Ron Parr

Neural Network Motivation

- Human brains are only known example of actual intelligence
 - Individual neurons are slow, boring
 - Brains succeed by using massive parallelism
 - Idea: 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?

Why Neural Networks?

Maybe computers should be more brain-like:

	Computers	Brains
Computational Units	10^8 gates/CPU	10^{11} neurons
Storage Units	10^{10} bits RAM 10^{13} 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^{10} Ops/s	10^{14} Ops/s

Comments on Jaguar

(world's fastest supercomputer as of 4/10)

- 2,332 Teraflops
- 10^{15} Ops/s (Jaguar) vs. 10^{14} Ops/s (brain)
- 224,256 processor cores
- 300 TB RAM (10^{15} bits)
- 10 PB Disk storage
- 7 Megawatts power
(~\$500K/year in electricity [my estimate])
- ~\$100M cost
- 4400 sq ft size (very large house)
- Pictures and other details: <http://www.nccs.gov/jaguar/>

More Comments on Modern Supercomputers vs. Brains

- What is wrong with this picture?
 - Weight
 - Size
 - Power Consumption
- What is missing?
 - Still can't replicate human abilities
(though vastly exceeds human abilities in many areas)
 - Are we running the wrong programs?
 - Is the architecture well suited to the programs we might need to run?

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.

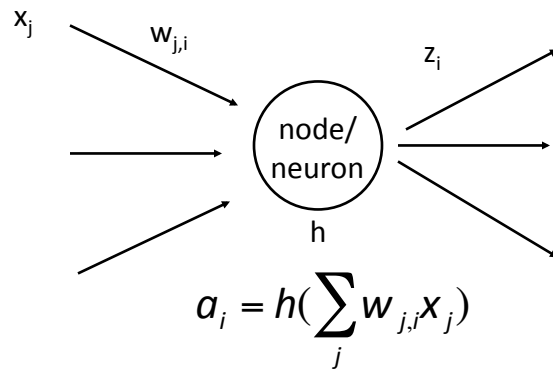
Use of neural networks

- Classic examples
 - 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 w/>99% accuracy
 - Trained to drive (Pomerleau's no-hands across America)
- Current examples
 - Credit risk evaluation, OCR systems, voice recognition, etc. (though not necessarily the best method for any of these tasks)
 - Built in to many software packages, e.g., matlab

Neural Network Lore

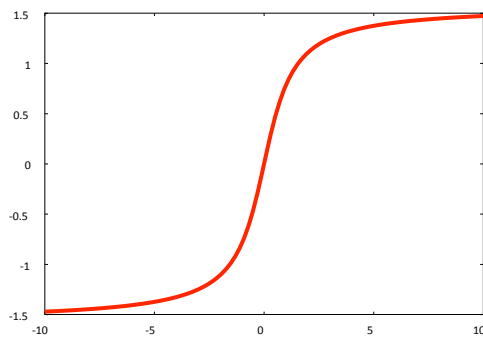
- Neural nets have been adopted with an almost religious fervor within the AI community - several times
- Often ascribed near magical powers by people, usually those who know the least about computation or brains
- For most AI people, magic is gone, but neural nets remain extremely interesting and useful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

Artificial Neurons



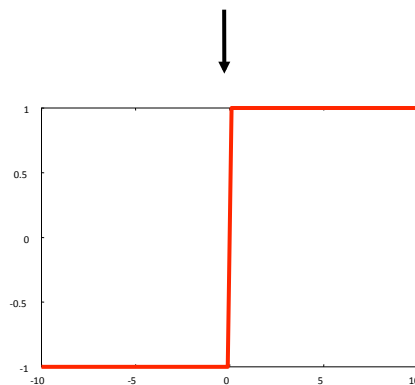
a_i is the activation level of neuron i
 h can be any function, but usually a smoothed step function

Threshold Functions



$h(x) = \tanh(x)$ or $1/(1+\exp(-x))$
(logistic sigmoid)

$h(x) = \text{sgn}(x)$
(perceptron)



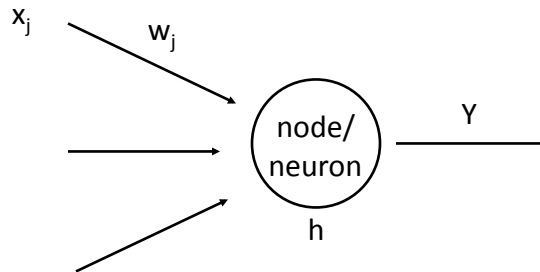
Network Architectures

- 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 can be viewed as computing a complex non-linear function
- Typical uses in learning:
 - Classification (usually involving complex patterns)
 - General continuous function approximation

Special Case: Perceptron



h is a simple step function (sgn)

Perceptron Learning

- We are given a set of inputs $\mathbf{x}^{(1)} \dots \mathbf{x}^{(n)}$
- $t^{(1)} \dots t^{(n)}$ is a set of target outputs (boolean) $\{-1, 1\}$
- \mathbf{w} is our set of weights
- output of perceptron = $\mathbf{w}^T \mathbf{x}$
- $\text{Perceptron_error}(\mathbf{x}^{(i)}, \mathbf{w}) = -\mathbf{w}^T \mathbf{x}^{(i)} * t^{(i)}$
- Goal: Pick \mathbf{w} to optimize:

$$\min_{\mathbf{w}} \sum_{i \in \text{misclassified}} \text{perceptron_error}(\mathbf{x}^{(i)}, \mathbf{w})$$

Update Rule

Repeat until convergence:

$$\forall_{i \in \text{misclassified}} \forall_j : w_j \leftarrow w_j + \alpha x_j^{(i)} t^{(i)}$$

↑
“Learning Rate”
(can be any constant)

- i iterates over samples
- j iterates over weights

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

Perceptron Learning

The Good News First

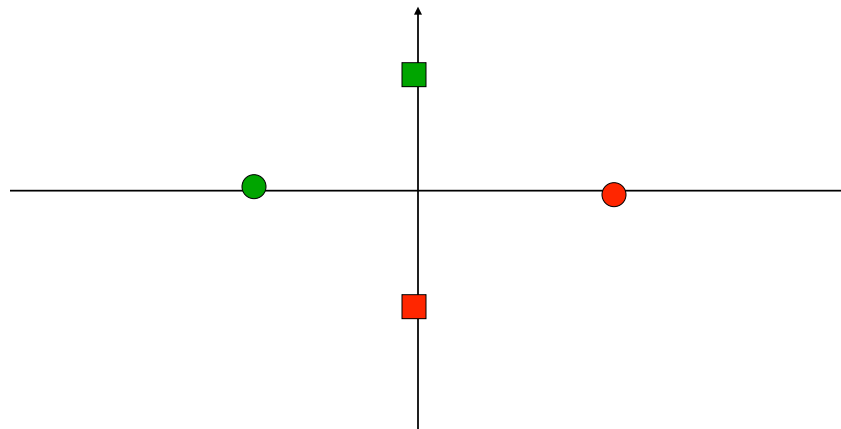
- For functions that are representable using the perceptron architecture (more on this later):
 - Perceptron learning rule converges to correct classifier for any choice of α
 - Online classification possible for streaming data (very efficient implementation)
- Positive perceptron results set off an explosion of research on neural networks

Perceptron Learning

Now the Bad News

- Perceptron computes a linear function of its inputs,
- Asks if the input lies above a line (hyperplane, in general)
- Representable functions are functions that are “linearly separable”; i.e., there exists a line (hyperplane) that separates the positive and negative examples
- If the training data are not linearly separable:
 - No guarantees
 - Perceptron learning rule may produce oscillations

Visualizing Linearly Separable Functions



Is red linearly separable from green?

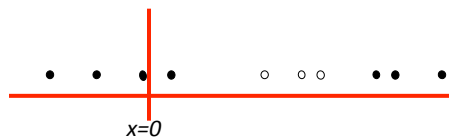
Are the circles linearly separable from the squares?

Observations

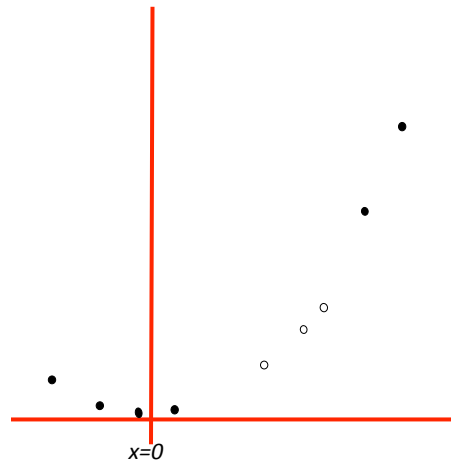
- Linear separability is fairly weak
- We have other tricks:
 - Functions that are not linearly separable in one space, may be linearly separable in another space
 - If we engineer our inputs to our neural network, then we change the space in which we are constructing linear separators
 - Every function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

Separability in One Dimension

If we have just a single input x , there is no way a perceptron can correctly classify these data



Harder 1-dimensional dataset



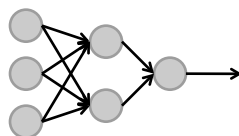
- Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too, using $1, x, x^2$ as inputs to the perceptron

Copyright © 2001, 2003, Andrew W. Moore

Multilayer Networks

- Once people realized how simple perceptrons were, they lost interest in neural networks for a while (feature engineering turned out to be impractical in many cases)
- Multilayer networks turn out to be much more expressive (with a smoothed step function)
 - Use sigmoid, e.g., $\tanh(w^T x)$ or logistic sigmoid: $1/(1+\exp(-x))$
 - With 2 layers, can represent any continuous function
 - With 3 layers, can represent many discontinuous functions
- Tricky part: How to adjust the weights



Smoothing Things Out

- Idea: Do gradient descent on a smooth error function
- Error function is sum of squared errors
- Consider a single training example first

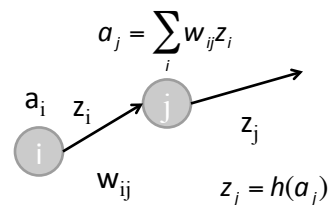
$$E = 0.5 \text{error}(X^{(i)}, w)^2$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial a_j} = \delta_j$$

$$\frac{\partial a_j}{\partial w_{ij}} = z_i$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j z_i$$

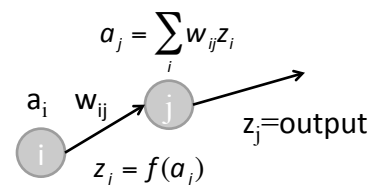


Propagating Errors

- For output units (assuming no weights on outputs)

$$\frac{\partial E}{\partial a_j} = \delta_j = y - t$$

- For hidden units



$$\frac{\partial E}{\partial a_i} = \delta_i = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_i} = \sum_k \frac{\partial E}{\partial a_k} w_{ki} \frac{\partial h_i}{\partial a_i} = h'(a_i) \sum_k w_{ki} \delta_k$$

All upstream nodes from i

Differentiating h

- Recall the logistic sigmoid:

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

- Differentiating:

$$h'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{(1 + e^{-x})} \frac{e^{-x}}{(1 + e^{-x})} = h(x)(1 - h(x))$$

Putting it together

- Apply input \mathbf{x} to network (sum for multiple inputs)
 - Compute all activation levels
 - Compute final output (forward pass)

- Compute δ for output units

$$\delta = y - t$$

- Backpropagate δ 's to hidden units

$$\delta_j = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} = h'(a_j) \sum_k w_{kj} \delta_k$$

- Compute gradient update: $\frac{\partial E}{\partial w_{ij}} = \delta_j a_i$

Summary of Gradient Update

- Gradient calculation, parameter update have recursive formulation
- Decomposes into:
 - Local message passing
 - No transcendentals:
 - $h'(x)=1-h(x)^2$ for $\tanh(x)$
 - $h'(x)=h(x)(1-h(x))$ for logistic sigmoid
- Highly parallelizable
- Biologically plausible(?)
- Celebrated *backpropagation* algorithm

Good News

- Can represent any continuous function with two layers (1 hidden)
- Can represent essentially any function with 3 layers
- (But how many hidden nodes?)
- Multilayer nets are a universal approximation architecture with a highly parallelizable training algorithm

Back-prop Issues

- Backprop = gradient descent on an error function
- Function is nonlinear (= powerful)
- Function is nonlinear (= local minima)
- Big nets:
 - Many parameters
 - Many optima
 - Slow gradient descent
 - Risk of overfitting
 - Biological plausibility \neq Electronic plausibility
- Many NN experts became experts in numerical analysis (by necessity)

Neural Network Tricks

- Many gradient descent acceleration tricks
- Early stopping (prevents overfitting)
- Methods of enforcing transformation invariance (e.g. if you have symmetric inputs)
 - Modify error function
 - Transform/augment training data
 - Weight sharing
- Handcrafted network architectures

Neural Nets in Practice

- Many applications for pattern recognition tasks
- Very powerful representation
 - Can overfit
 - Can fail to fit with too many parameters, poor features
- Very widely deployed AI technology, but
 - Few open research questions (Best way to get a machine learning paper rejected: “Neural Network” in title.)
 - Connection to biology still uncertain
 - Results are hard to interpret
- “Second best way to solve any problem”
 - Can do just about anything w/enough twiddling
 - Now third or fourth to SVMs, boosting, and ???