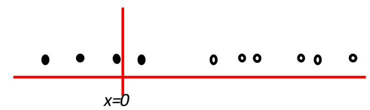
Neural Networks

CPS 570

Ron Parr

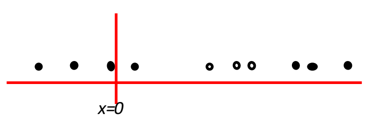
Suppose we're in 1-dimension

Easy to find a linear separator

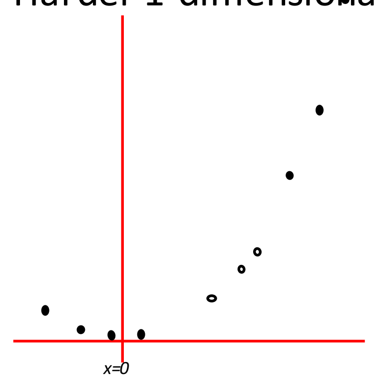


Harder 1-dimensional dataset

What can be done about this?



Harder 1-dimensional dataset

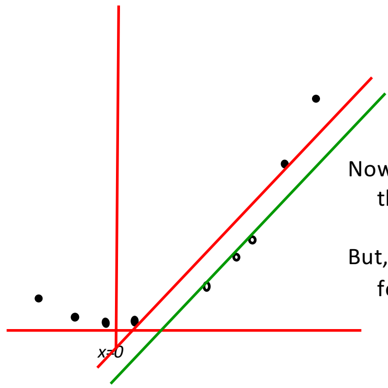


Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\Phi = (x, x^2)$$

Harder 1-dimensional dataset



Now linearly separable in the new feature space

But, what if the right feature set isn't obvious

$$\Phi = (x, x^2)$$

Motivation for non-linear Classifiers

- Linear methods are "weak"
  - Make strong assumptions
  - Can only express relatively simple functions of inputs
- Coming up with good features can be hard
- Why not make the classifier do more work for us?
  - What does the space of hypotheses look like?
  - How do we navigate in this space?

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 Sunway TaihuLight
(world's fastest supercomputer as of 4/12)

- 93 Petaflops
- ~10^{18} Ops/s (TaihuLight) vs. 10^{14} Ops/s (brain)
- 10M processor cores + GPU cores
- 1.3 PB RAM (10^{17} bits)
- 15 Megawatts power(>$1M/year in electricity [my estimate])
- ~$273M cost

More Comments on Titan

- 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.

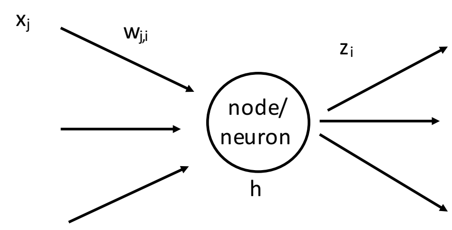
Early Use of neural networks

- 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
  - First coming: Perceptron
  - Second coming: Multilayer networks
  - Third coming (present): Deep networks
- Sound science behind neural networks: gradient descent
- Unsound social phenomenon behind neural networks: **HYPE!**

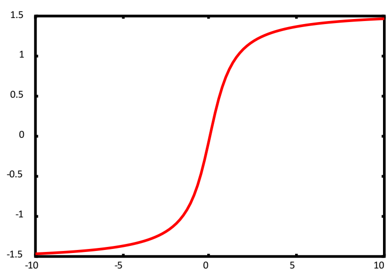
Artificial Neurons



$$a_i = h(\sum_j w_{j,i} x_j)$$

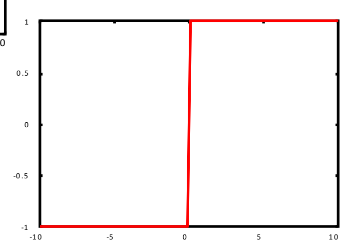
h can be any function, but usually a smoothed step function

Threshold Functions



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

h(x)=sgn(x)
(perceptron)



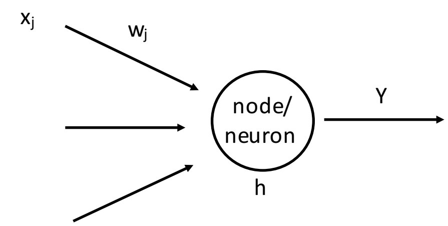
Network Architectures

- 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

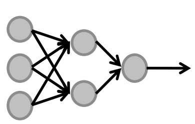
Special Case: Perceptron



h is a simple step function (sgn)

Multilayer Networks

- Once people realized how simple perceptrons were, they lost interest in neural networks for a while
- 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
  - 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

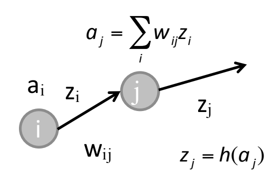
$$E = 0.5 error(X^{(i)}, w)^2$$

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

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

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

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



Calculus Reminder

- Chain rule for one variable: $\frac{\partial f \circ g}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$
- Chain rule for: $f : \Re^n \to \Re^k, g : \Re^m \to \Re^n$

$$J_x(f \circ g) = J_{g(x)}(f) J_x(g) = (k \times n)(n \times m)$$

- For k=1, m=1

$$J_x(f \circ g) = \sum_{i=1}^{n} \frac{\partial f}{\partial g(x)_i} \frac{\partial g(x)_i}{\partial x}$$

Propagating Errors

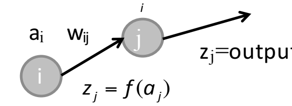
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \delta_j z_i$$

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

- For output units (assuming no weights on outputs)

$$\frac{\partial E}{\partial a_j} = \delta_j = y - t \qquad a_j = \sum_i w_{ij} z_i$$

- For hidden units



Chain rule

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

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 **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 δs to hidden units

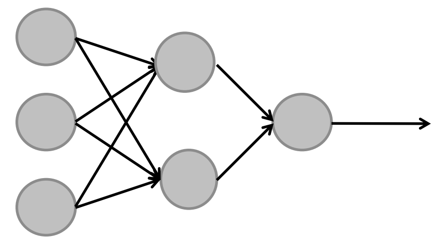
$$\delta_j = \sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial a_j} = h'(a_j)\sum_k w_{kj}\delta_k$$

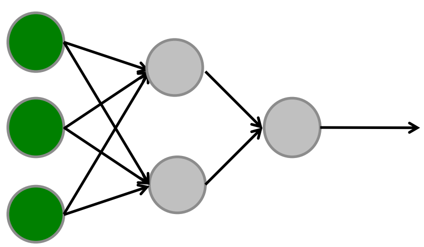
- Compute gradient update: $\frac{\partial E}{\partial w_{ij}} = \delta_j a_i$

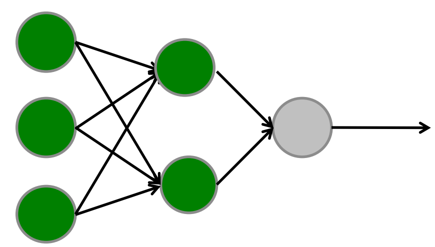
Summary of Gradient Update

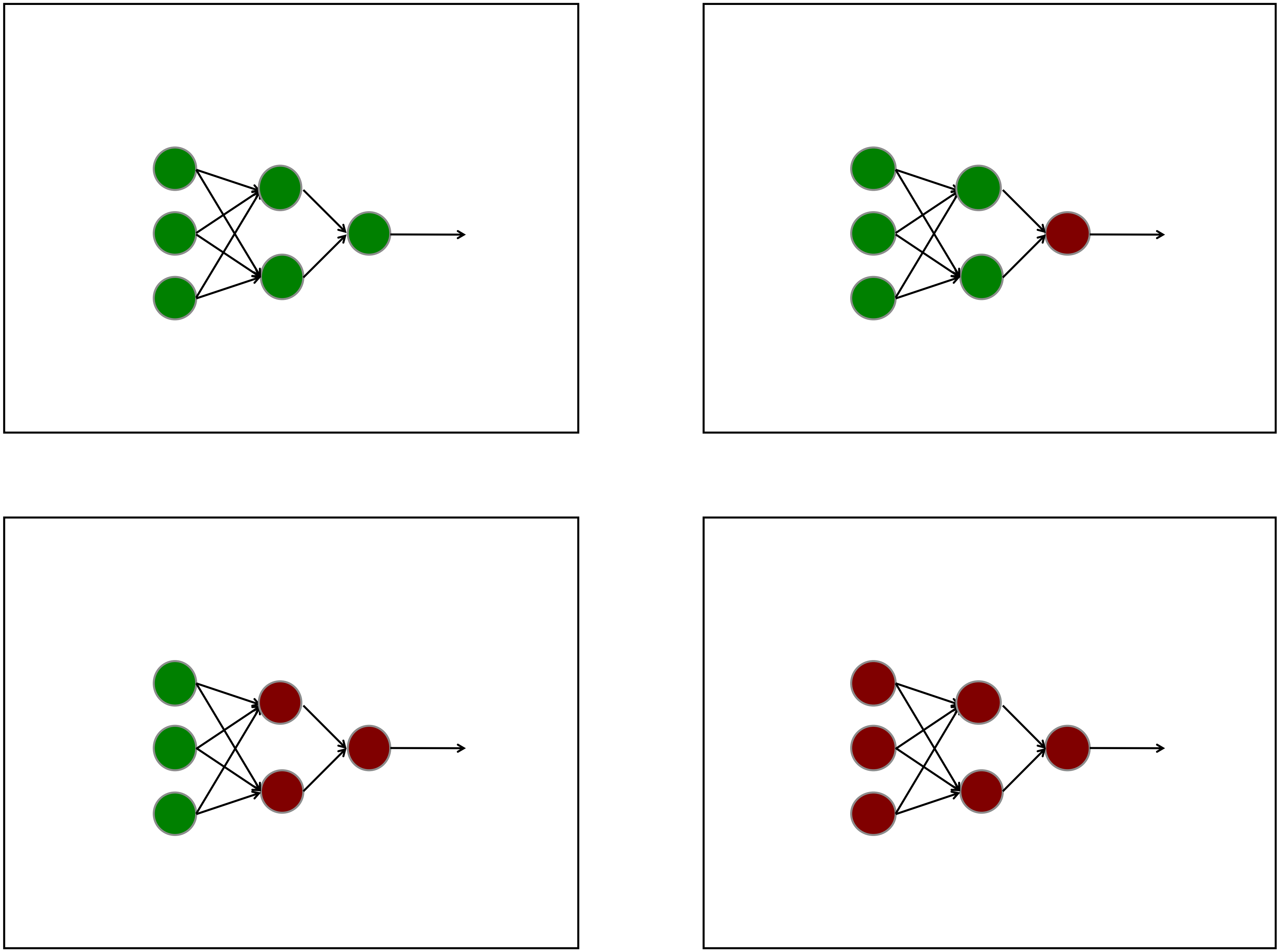
- Gradient calculation, parameter update have recursive formulation
- Decomposes into:
  - Local message passing
  - No transcendentals:
    - $h'(x)=1-h(x)^2$ for $\tanh(x)$
    - $H'(x)=h(x)(1-h(x))$ for logistic sigmoid
- Highly parallelizable
- Biologically plausible(?)

- 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

Backprop 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 ≠ 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

NN History Through the Second Coming

- Second wave of interest in neural networks lost research momentum in the 1990s – though still continued to enjoy many practical applications
- Neural network tricks were not sufficient to overcome competing methods:
  - Support vector machines
  - Clever feature selection methods wrapped around simple or linear methods
- 2000-2010 was an era of linear + special sauce
- What changed?

Deep Networks

- Not a learning algorithm, but a family of techniques
  - Training sometimes done in stages, rather than monolithically, with different layers of the network getting training separately
  - Sometimes combines ideas from supervised and unsupervised learning, with middle layers trained to do some kind of feature compression
  - Clever crafting of network structure – convolutional nets
- Exploit massive computational power
  - Parallel computing
  - GPU computing
  - Very large data sets (can reduce overfitting)

Deep Networks Today

- Still on the upward swing of the hype pendulum
- State of the art performance for many tasks:
  - Speech recognition
  - Object recognition
  - Playing video games
- Controversial:
  - Hype, hype, hype! (but it really does work well in many cases!)
  - Theory lags practice
  - Collection of tricks, not an entirely a science yet
  - Results are not human-interpretable

Conclusions

- Neural nets are a general function approximation architecture
- Gradient has nice properties, permitting highly parallelizable training
- Historically wild swings in popularity
- Currently on upswing due to clever changes in training methods, use of parallel computation, and large data sets