Neural networks

15-381 lecture notes, 23 March 2000

David Cohn <www.cs.cmu.edu/~cohn>

Chapter 19 of Russell & Norvig

After this lecture, you should understand:

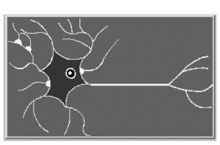
- what you can do with perceptrons
- what you can't do with perceptrons
- problems facing the traditional MLP
- some applications of neural networks
- how to use neural networks for regression

Biological inspiration for neural networks

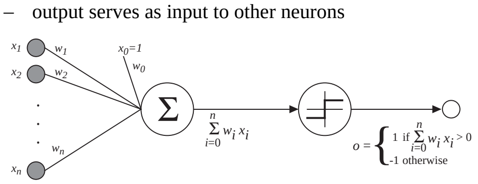
- Computers are good at things people aren't
  - massive, fast arithmetic calculation
  - tediously reliable computation
- People are good at things computers aren't
  - face/speech recognition
  - picking out a conversation in a noisy room
  - complex control problems - riding a bike, flying an airplane, driving a car
- People use a different kind of processor
  - neuron switching time ~100 Hz
  - number of neurons ~10^{10}
  - connections/neuron ~10^4 - 10^5
  - scene recognition time ~0.1 second
- 100 "cycles" to do scene recognition?
  - highly parallel, distributed computation
- How to emulate? *"Hey - let's make a computer that works like the brain does!"*
  - many neuron-like switching units
  - weighted interconnections between units
  - tune weights automatically, via "learning"

Discriminative classification redux: a single neuron

- Crude model of biological neuron
  - sums inputs from many dendrites, weighted by strength of synapse (connection)
  - if input exceeds action potential, neuron fires
  - axon carries neuron output to dendrites of many other neurons



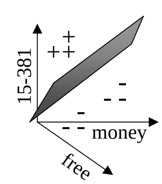
- Even cruder mathematical neuron
  - activation *a* is weighted sum of inputs $w_1x_1+w_2x_2+...+w_nx_n$
  - if *a* greater than threshold w_0, set output to *1*, otherwise *0*
  - output serves as input to other neurons



- Rosenblatt 1962 called these "perceptrons"

What does a perceptron do?

- Represents some useful functions
  - AND, OR, MAJORITY, NAND, ...
- Example: text classification using vector space model

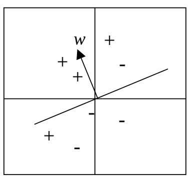


- Problem: requires *finding* weights that separate data

Perceptron learning rule

- First, a mathematical convenience
  - treat threshold w_0 uniformly with rest of weights - pretend "bias" input x_0 is always "1"
  - can now write as vectors $w^T x > 0$
- Want to align hyperplane with boundary
  - treat each example x as vector
  - weight vector w is normal of hyperplane
  - if x is misclassified as positive, add x (or fraction of x) to w
  - if x is misclassified as negative, subtract x (or fraction of x) from w
  - Mathematically: $w \leftarrow w + \eta x$ if x^+ and $w^T x < 0$

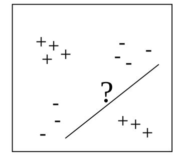
    $w \leftarrow w - \eta x$ if x^- and $w^T x > 0$



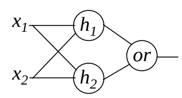
- Step size η usually 0< η<1
- Guarantee: if there is a separating hyperplane, and if η is small enough, w will converge to it
- *Lack of guarantee: it may take a long time*

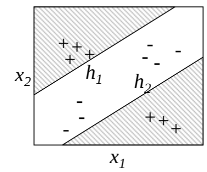
What perceptrons can't do

- Can't represent anything that's not linearly separable
  - fairly obvious, no?



- What do we do?
  - follow the "neural" model, stack perceptrons in a network
  - now we can represent all boolean functions with single "hidden" layer (but might require exponential number of hidden units)



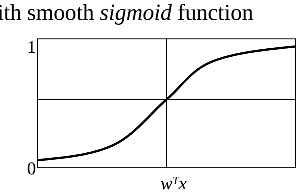


- But - now how do we apply our perceptron learning algorithm to learn functions from data?
  - "no clear way to extend perceptron learning algorithm to multilayer networks" - Minsky & Papert
  - plunged the field into despair and disrepute for 20 years
- But then somebody remembered calculus!

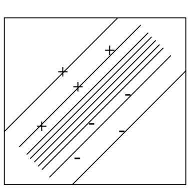
Gradient descent learning for perceptrons

- Hillclimbing on error is an old trick
  - compute $\partial E/\partial w$, set $w \leftarrow w - \eta\, \partial E/\partial w$
  - take steps in direction to decrease error
- But derivative of threshold-based perceptron is uninformative
- Replace hard threshold with smooth *sigmoid* function

$$\hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$$



- Now output never 0 or 1, but varies smoothly between
  - usually set thresholds - y > 0.9 means "1", < 0.1 means "0"
  - given certain statistical assumptions, can use as estimate of *probability* that y is 1 or 0



Gradient descent

- Define error of perceptron as sum squared difference between output and label

$$E = \sum_d \left(y_d - \sigma(w^T x_d)\right)^2$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_d \left(y_d - \sigma(w^T x_d)\right)^2$$

$$= \sum_d \frac{\partial}{\partial w_j} \left(y_d - \sigma(w^T x_d)\right)^2$$

$$= \sum_d 2\left(y_d - \sigma(w^T x_d)\right) \frac{\partial}{\partial w_j} \left(y_d - \sigma(w^T x_d)\right)$$

$$= \sum_i 2\left(y_i - \sigma(w^T x)\right) \frac{-\partial \sigma(w^T x)}{\partial w_j}$$

$$= \sum_d 2\left(y_d - \sigma(w^T x_d)\right) \frac{-\partial \sigma(w^T x_d)}{\partial w^T x_d} \frac{\partial w^T x_d}{\partial w_j}$$

- Two useful derivatives: $\dfrac{\partial \sigma(o)}{\partial o} = o(1-o)$

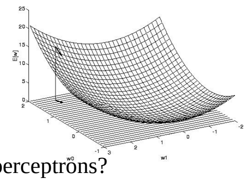
  $\dfrac{\partial w^T x_d}{\partial w_j} = x_{d,j}$

- Which lets us write

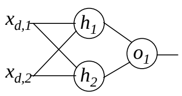
$$\frac{\partial E}{\partial w_j} = \sum_d 2\left(y_d - \sigma(w^T x_d)\right) \cdot \sigma(w^T x_d) \cdot (1 - \sigma(w^T x_d)) x_{d,j}$$

Backpropagation - gradient descent for MLPs

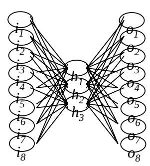
- We now have a different learning rule for perceptrons
  - follows error surface to minimum



- Can we alter it to train multilayer perceptrons?
  - define o_k to mean output of neuron k,
  - initialize weights $w_{i,j}$ to small random values
  - for each output unit k, define $\delta_k = o_k(1- o_k)(y_k - o_k)$
  - for each intermediate ("hidden") unit h, define $\delta_h = o_h(1- o_h)\Sigma_k w_{h,k}\, \delta_k$
  - update network weights $w_{i,j} \leftarrow w_{i,j} + \eta\delta_h x_{d,i}$
    - batch learning: accumulate $\Delta w_{i,j}$ over all training patterns, change all at once
    - incremental learning: apply pattern x_d, update weights, apply next...

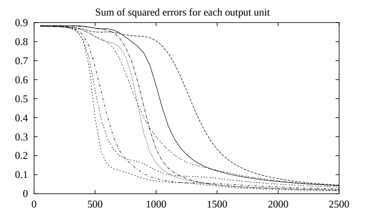


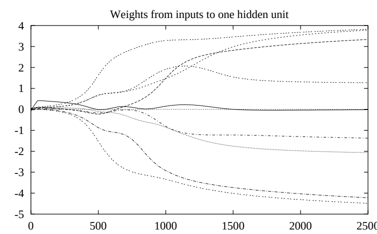
- Apply to arbitrary depth networks with arbitrary numbers of hidden or output units



Backpropagation - training

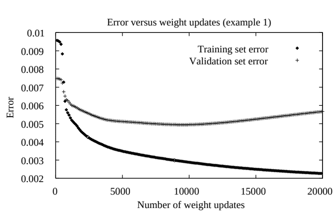
- Finds *local*, not *global* minimum training error
  - gradient descent on error surface seeks "bottom of bowl"
  - real error surface not very bowl-like, with many local minina
- In practice, seems to work fairly well
  - common trick: multiple restarts
- How will it generalize?
  - use a validation set, or estimate using VC dimension bounds

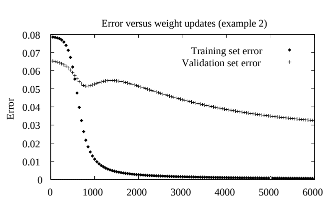




Backpropagation - overfitting

- Initially, most neurons in "linear" region of sigmoid, don't add much to representational power
  - as we continue training, more and more of these are pushed out of linear region, add complexity to model
  - by training too long, may overfit - fit training data well, but not generalize well
- How to prevent overfitting?





Some backprop tricks

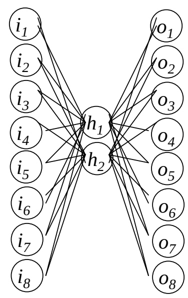
- Training can be painfully slow
- Using it (after training) can be fairly quick

- How to speed up "vanilla" backprop?
  - One trick: add a "momentum" term to weight change
    - $\Delta w_{i,j}(t) = \eta\delta_h x_{d,i} + \alpha\, \Delta w_{i,j}(t-1)$
  - Adaptive step sizes
    - if $sgn(\Delta w_{i,j}(t)) = sgn(\Delta w_{i,j}(t-1))$, set $\eta = 1.5\eta$
    - else set $\eta = \eta/2$
  - Quasi-Newton methods
    - estimate second derivative of error surface (via multiple steps or differentiation)
    - approximate as parabolic bowl, jump to expected minimum
  - Conjugate gradient methods
    - take steps expected to minimize error in independent directions

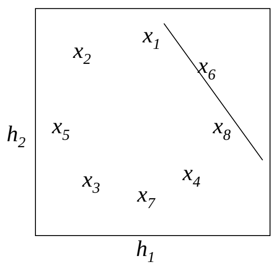
Example neural net: the autoencoder

- Get network to replicate its inputs, fed through a bottleneck

10000000→10000000
01000000→01000000
00100000→00100000
00010000→00010000
00001000→00001000
00000100→00000100
00000010→00000010
00000001→00000001

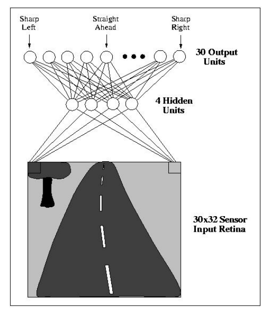


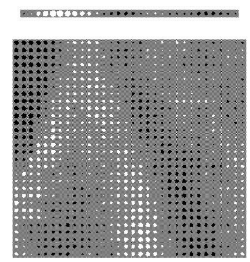
- What hidden unit representation does backprop find?



Some neural net applications

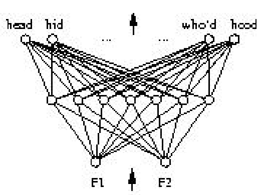
- Driving a car - steering to stay on the road
  - ALVINN *[Pomerleau]*
  - "No Hands Across America"

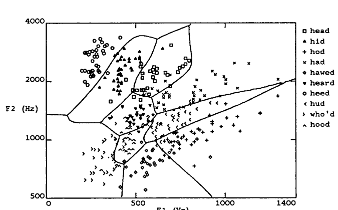




Some neural net applications

- Speech recognition
  - convert raw acoustic signal to *formants* (resonant frequencies in signal)
  - classify sound according to first two formants





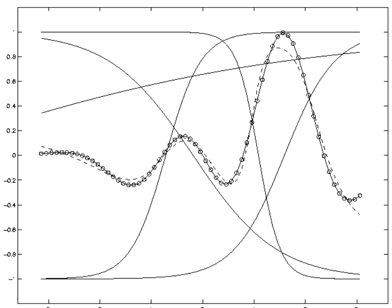
Some neural net applications

- Face detection/recognition

- Evaluation functions in game tree search
  - Neurogammon *[Tesauro]* learned to score board states in backgammon
  - Note: requires more than simple supervised learning - "value" of state assumes optimal play, which we can't guarantee
  - Now best computer backgammon player in the world - *very* close to best human

Neural networks for regression

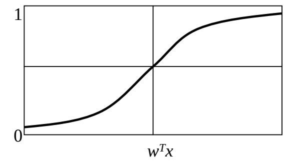
- Rather than training outputs to 0 or 1, train to intermediate values
  - can scale y, or omit non-linearity in output layer to get arbitrary range outputs
  - Question: can we just get rid of *all* the nonlinearities? It would simplify the math...
  - using single hidden layer, can represent every bounded continuous function to arbitrarily small error
    - might have to be a *big* hidden layer
    - often advantage to having several smaller layers
    - trade-off: “deeper” networks take longer to propagate error back and train



Alternative nonlinearities

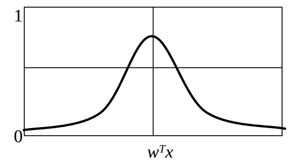
- We’ve been using sigmoid:

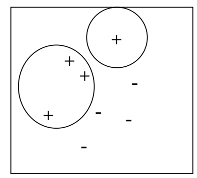
$$\hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$$



- Could also use Gaussian-like bump
  - *radial basis functions*

$$\hat{y} = \exp\left(-\left(w^T x\right)^2 / w_{\text{var}}\right)$$





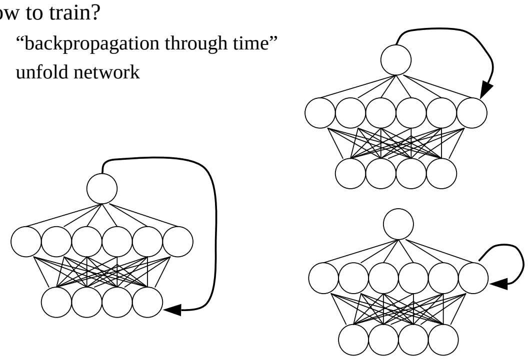
- In fact, can use *any* differentiable nonlinearity!
  - which ones are best depends on
    - ease of computation
    - match of “selection” function with problem boundaries

Recurrent networks - “What happens next?”

- We can use a nnet to predict what’s going to happen at the next moment/minute/day
  - can we extrapolate to future moments/minutes/days?
- One way: train a network for each interval we want
- Another way: feed network outputs back into network
  - either to inputs, or to hidden units
  - can even feed internal outputs back

- Run forward to predict time series

- How to train?
  - “backpropagation through time”
  - unfold network



Some things to ponder

- When might we want to use neural networks?
- When might we want to avoid using them?

- How to choose our neural net architecture?
  - how many layers?
  - how many hidden units?
  - what nonlinearities?

Things we've left uncovered

- Nnets that adapt their structure
- Alternative error functions
  - penalizing large weights
  - encouraging unused weights to go to zero
  - training on slope of target functions
- Alternative forms of neural networks
  - unsupervised nnets: Kohonen's neural maps
  - ART1,...,ARTn - attempts at cognitively plausible neural nets
- Neural networks as statistical models
  - yes, you can use a nnet as a generative model!
  - close cousins to Bayes nets