Neural networks

- Reading: Russell and Norvig, ch. 18.7, Wikipedia pages on *artificial neural network*, *activation function*, *backpropagation*
- Basic network unit
- Network organization
- Learning
- Backpropagation

Motivation

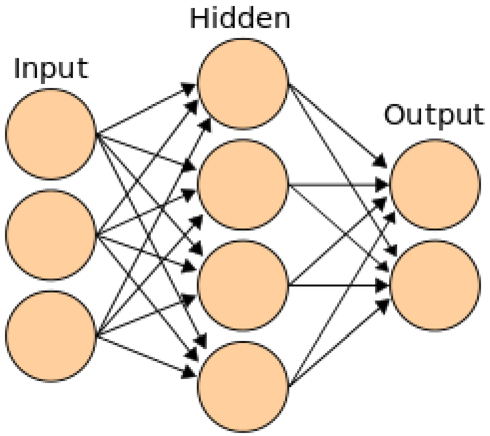
- Simulate human neurons with simple model
  - Nodes with inputs and outputs
    - Directed graph
  - Inputs and outputs model idea of human synapses firing
    - Signals typically vary from 0 to 1
    - Function at node started as step or sigmoid function
    - Now many more options: see https://en.wikipedia.org/wiki/Activation_function

Motivation

- Nodes organized in “layers”
  - Computation is synchronized, with node output at time t used as input for successor nodes at time $t+1$
  - Typically, each layer has its own purpose
    - Wikipedia example: for facial recognition, initial layers check features (pupil, iris, eyebrows), later layers check more abstract features (like eyes or nose), last layers check objective (face)
- Weights used in computation of function, placed on connections
  - Learning measures error for training data points against expected results
  - Learning modeled by modifying weights

Basic network

- Graph of McCulloch-Pitts “unit”s



  - [By en:User:Cburnett [GFDL (http://www.gnu.org/copyleft/fdl.html)]

Computing basic functions

- McCulloch-Pitts
  - includes a constant input a_0 = -1 with a *bias weight* on this input
  - Applies activation function g to weighted sum of inputs
    - For step function, bias weight is effectively a threshold
    - Ex: for nonnegative input weight, if weighted sum > bias weight, output = 1, else output = 0
  - With 1 input, input weight = -1, bias weight = -0.5 produces NOT of input
    - Input = 0, weighted sum = 0.5 + 0 yields output of 1

Computing basic functions

- With 2 inputs (each = 0 or 1) and both input weights = 1
  - Bias weight = 1.5 produces AND of inputs
    - Both inputs = 1, weighted sum = -1.5 + 1 + 1 = 0.5 $\Rightarrow$ $g(0.5) = 1$ (true)
    - Either input = 0, weighted sum <= -1.5 + 0 + 1 = -0.5 $\Rightarrow$ $g(-0.5) = 0$ (false)
  - Bias weight = 0.5 produces OR of inputs
    - Output = 0 only if both inputs = 0, weighted sum = -0.5

Network organization

- If only one direction (*feed-forward*)
  - Only implement functions, no internal state
  - For single-layer network (*perceptrons*), creates linear decision boundary
    - Cannot represent functions like XOR (and therefore addition), or restaurant waiting problem
    - On the other hand, can learn fairly complex functions like majority function easily
      - For N inputs, make each input weight 1/N
- If cycles allowed, can store information (like flip-flop circuits)

Network organization (cont'd)

- 2-layer perceptrons can represent any continuous function
- 3-layer perceptrons can represent any function

Learning

- If actual output is less than expected output, want to increase weights
- If actual is greater than expected, want to decrease weights
- Three concerns:
  - How much to increase/decrease
  - Which weights to adjust
  - In particular, how should weights for hidden layers be adjusted
    - When so many weights affect result, hard to assign adjustments to individual components

Gradient-based approach

- Define a *cost function* based on difference between expected and predicted output
  - Common function is (square of difference)/2
- Change weight i by $\alpha * a * \Delta$
  - α = *learning rate* = coefficient used to adjust how quickly weights change
  - a_i = input i
  - Δ = error * derivative of g with respect to input

Backpropagation

- Propagate errors backwards, to assign portion of responsibility for error to each preceding node
  - Output layer uses formula on previous slide
  - For hidden layers, Δ = weighted sum of Δ for successors * derivative of g with respect to input
  - You can find a detailed example at: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
- [Backpropagation does not seem to correlate to the way real neurons function – in general, artificial neural networks have diverged from the original inspiration]

Improvements

- An adaptive learning rate to improve rate of convergence and avoid oscillating between weights
- Adding an *inertia* factor representing the rate of change can help move past plateaus
  - Plateau implies gradient approaches 0, inertia causes weights to continue to change in direction of last change

Learning modes

- *Stochastic* learning = change weight after trying each training input
  - Final weights will not model individual inputs perfectly – can interpret this as each input including “noise”
    - Less chance of getting stuck in local minimum
- *Batch* learning = change weight based on aggregate information after trying collection of training inputs
  - Change in weights will be more stable, faster
- Mixed approach (small batches) is common

Result quality

- Depends on domain
- Ex: handwriting of digits recognition
  - Has reached 0.9% error
  - Best algorithm at 0.6%