Artificial Intelligence Programming

More Neural Networks

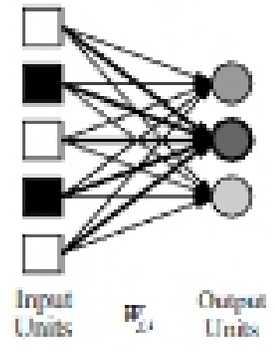
Chris Brooks

Department of Computer Science
University of San Francisco

Neural networks - refresher

- A network is composed of layers of nodes
  - Input, hidden, output layers in feedforward nets
- An input is applied to the input units
- The resulting output is the value of the function the net computes.

Perceptrons - refresher



- Single-layer networks (perceptrons) are the simplest form of NN.
- Easy to understand, but computationally limited.
- Each input unit is directly connected to one or more output units.

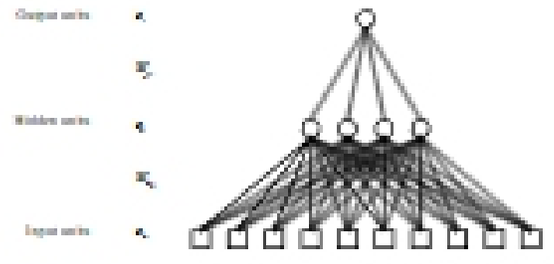
Delta rule - refresher

- The appeal of perceptrons is the ability to automatically learn their weights in a supervised fashion.
- The weight updating rule is known as the **delta rule**.
- $\Delta w_i = \alpha \sum_{d \in D}(t_d - o_d)x_{id}$
- Where D is the training set, t_d is expected output and o_d is actual output.

Multilayer Networks

- While perceptrons have the advantage of a simple learning algorithm, their computational limitations are a problem.
- What if we add another "hidden" layer?
- Computational power increases
  - With one hidden layer, can represent any continuous function
  - With two hidden layers, can represent any function
- Problem: How to find the correct weights for hidden nodes?

Multilayer Network Example



Backpropagation

- Backpropagation is an extension of the perceptron learning algorithm to deal with multiple layers of nodes.
- Nodes use sigmoid activation function, rather than the step function
  - $g(input_i) = \frac{1}{1+e^{-input_i}}$.
  - $g'(input_i) = g(input_i)(1 - g(input_i))$ (good news here - to compute g', we just need g)
- We will still "follow the gradient", where g' gives us the gradient.

More on computing error

- Recall that our goal is to minimize the error function.
- To do this, we want to change the weights so as to reduce the error.
- We define error as a function of the weights like so:
  - $E(\mathbf{w}) = expected - actual$
  - $E(\mathbf{w}) = expected - g(input)$
- So, to determine how to change the weights, we compute the derivative of error with respect to the weights.
  - This tells us the slope of the error curve at that point.

More on computing error

- $E(\mathbf{w}) = expected - g(input)$
- $E(\mathbf{w}) = expected - g(\mathbf{w} \cdot \mathbf{i})$
- $\frac{dE}{dW} = 0 - g'(input)\mathbf{i}$
- $\delta_w = \frac{dE}{dW} = -g(input) * (1 - g(input)) * \mathbf{i}$

Updating hidden-output weights

- Each weight is updated by $\alpha * \Delta_i$
- $W_{j,i} = W_{j,i} + \alpha * a_j * \Delta_i$

Backpropagation

- Updating input-hidden weights:
- Idea: each hidden node is responsible for a fraction of the error in δ_i.
- Divide δ_i according to the strength of the connection between the hidden and output node.
- For each hidden node j
- $\delta_j = g(input)(1 - g(input)) \sum_{i \in outputs} W_{j,i}\delta_i$
- Update rule for input-hidden weights:
- $W_{k,j} = W_{k,j} + \alpha * input_k * \delta_j$

Backpropagation Algorithm

- The whole algorithm can be summed up as:
  While not done:
    for d in training set
      Apply inputs of d, propagate forward.
      for node i in output layer
        $\delta_i = output * (1 - output) * (y_{exp} - output)$
      for each hidden node j
        $\delta_j = output * (1 - output) * \sum W_{j,i}\delta_i$
      Adjust each weight
        $W_{j,i} = W_{j,i} + \alpha * \delta_i * input_j$

Theory vs Practice

- In the definition of backpropagation, a single update for all weights is computed for all data points at once.
  - Find the update that minimizes total sum of squared error.
- Guaranteed to converge in this case.
- Problem: This is often computationally space-intensive.
  - Requires creating a matrix with one row for each data point and inverting it.
- In practice, updates are done incrementally instead.

Stopping conditions

- Unfortunately, incremental updating is not *guaranteed* to converge.
- Also, convergence can take a long time.
- When to stop training?
  - Fixed number of iterations
  - Total error below a set threshold
  - Convergence - no change in weights

Comments on Backpropagation

- Also works for multiple hidden layers
- Backpropagation is only guaranteed to converge to a local minimum
  - May not find the absolute best set of weights
- Low initial weights can help with this
  - Makes the network act more linearly - fewer minima
- Can also use random restart - train multiple times with different initial weights.

Momentum

- Since backpropagation is a hillclimbing algorithm, it is susceptible to getting stuck in plateaus
  - Areas where local weight changes don't produce an improvement in the error function.
- A common extension to backpropagation is the addition of a momentum term.
  - Carries the algorithm through minima and plateaus.
- Idea: remember the "direction" you were going in, and by default keep going that way.
- Mathematically, this means using the second derivative.

Momentum

- Implementing momentum typically means remembering what update was done in the previous iteration.
- Our update rule becomes:
- $\Delta w_{ji}(n) = \alpha \delta_j x_{ji} + \beta \Delta w_{ji}(n-1)$
- To consider the effect, imagine that our new delta is zero (we haven't made any improvement)
- Momentum will keep the weights "moving" in the same direction.
- Also gradually increases step size in areas where gradient is unchanging.
  - This speeds up convergence, helps escape plateaus and local minima.

Design issues

- One difficulty with neural nets is determining how to encode your problem
  - Inputs must be 1 and 0, or else real-valued numbers.
  - Same for outputs
- Symbolic variables can be given binary encodings
- More complex concepts may require care to represent correctly.