Neural Network Backpropagation

3-2-16
Recall from Monday...

Perceptrons can only classify linearly separable data.
Multi-layer networks

- Can represent any boolean function.
- We don’t want to build them by hand, so we need a way to train them.
- Algorithm: backpropagation.
  - You’ve already seen this in action in yesterday’s lab.
Backpropagation networks

- Backpropagation can be applied to any directed acyclic neural network.
- Activation functions must be differentiable.
- Activation functions should be non-linear.
  - OK
  - not OK
- Layered networks allow training to be parallelized within each layer.
Sigmoid activation functions

- We want something like a threshold.
  - Neuron is inactive below the threshold; active above it.
- We need something differentiable.
  - Required for gradient descent.

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
Gradient descent

- Define the squared error at each output node as: \( E(\vec{w}) = (t - o)^2 \)
- Update weights to reduce error.
  - Take a step in the direction of steepest descent:
    \[
    w_i = w_i - \eta \frac{\partial E}{\partial w_i}
    \]
Computing the error gradient

\[ E(\vec{w}) = (t - o)^2 \quad o = \sigma(\vec{w} \cdot \vec{x}) \]

\[ \sigma(\vec{w} \cdot \vec{x}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}} \]

\[ \vec{w} \cdot \vec{x} = \sum_{i} w_i x_i \]

\[ \frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)) \]

... algebra ensues ...

\[ \frac{\partial E}{\partial w_i} = -o(1 - o)(t - o)x_i \]
Gradient descent step for output nodes

\[ w_i = w_i + \eta o(1 - o)(t - o)x_i \]

\[ t = 0.9 \quad \bar{w} \cdot \bar{x} = 0.8 \quad o \approx 0.70 \]

\[ w_0 = 1 + 0.5 \cdot 0.7(1 - 0.7)(.9 - .7)2 = 1.04 \]

\[ w_1 = -1 + 0.5 \cdot 0.7(1 - 0.7)(.9 - .7)1.2 = -0.97 \]
Backpropagation

Key idea: at hidden units, use the next-layer change instead of the error function.

- Determine the node’s contribution to its successors.

\[ \sum_{k \in \text{next layer}} w_{hk} \delta_k \]

\[ \delta_k = o_k (1 - o_k) (t_k - o_k) \]

- Update incoming weights using this “error”

\[ \delta_h = o_h (1 - o_h) \sum_{k \in \text{next layer}} w_{hk} \delta_k \]

\[ \Delta w_i = \eta \delta_h x_i \]
Backpropagation algorithm

for 1:training runs

    for example in training_data:
       run example through network
       compute error for each output node
       for each layer (starting from output):
           for each node in layer:
               gradient descent update on incoming weights
Exercise: run a backprop step on this network