Details of Backprop

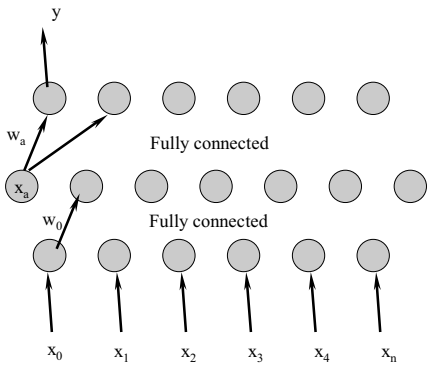
Plus Advice and Modifications

A little more derivation

- Recall:

$$\frac{\partial E}{\partial w_i} = -2\left(y - f(\vec{x})\right)\frac{\partial\left[F(\vec{w}^T\vec{x})\right]}{\partial\left(\vec{w}^T\vec{x}\right)}x_i$$

A multilayer net...



For weights in the output layer...

- Clearly…

$$\frac{\partial E}{\partial w_a} = -2\left(y - f(\vec{x})\right)\frac{\partial\left[F(\bullet)\right]}{\partial(\bullet)}x_a$$

- but what about the other layer, the "hidden" layer?

For weights in hidden layers:

$$\frac{\partial E}{\partial w_{\text{hidden}}} = -2\left(y - f(\vec{x})\right)\frac{\partial\left[F(\vec{w}_{\text{output}}{}^T\vec{x}_{\text{hidden}})\right]}{\partial\left(\vec{w}_{\text{output}}{}^T\vec{x}_{\text{hidden}}\right)}\frac{\partial\left[\vec{w}_{\text{output}}{}^T\vec{x}_{\text{hidden}}\right]}{\partial(w_{\text{hidden}})} =$$

$$-2\left(y - f(\vec{x})\right)\frac{\partial\left[F(\vec{w}_{\text{output}}{}^T x_{\text{hidden}})\right]}{\partial\left(\vec{w}_{\text{output}}{}^T x_{\text{hidden}}\right)}\frac{\partial\left[\vec{w}_{\text{output}}{}^T x_{\text{hidden}}\right]}{\partial(w_{\text{hidden}})} =$$

$$-2\left(y - f(\vec{x})\right)\frac{\partial\left[F\left(\vec{w}_{\text{output}}{}^T F\left(\vec{w}_{\text{hidden}}{}^T\vec{x}\right)\right)\right]}{\partial\left(\vec{w}_{\text{output}}{}^T F\left(\vec{w}_{\text{hidden}}{}^T\vec{x}\right)\right)}w_{\text{output}}\frac{\partial\left[F\left(\vec{w}_{\text{hidden}}{}^T\vec{x}\right)\right]}{\partial\left(\vec{w}_{\text{hidden}}{}^T\vec{x}\right)}x$$

The BP Trick...

- Let's call W_j the updates for the weights in layer j, x_j the input to this layer, a the actual network output, and F_j' the derivative of the activation function
- Let's say

$$\Delta W_j = c\vec{\Delta}_j\,\vec{x}_j^T$$

Backpropagating the deltas...

- Fast and loose notation here…
- For the output layer ($j=N$)

$$\vec{\Delta}_N = (\vec{y} - \vec{a})F_N'$$

- for all other layers

$$\vec{\Delta}_j = W_{j+1}^T \vec{\Delta}_{j+1} F_j'$$

Cool Trick for Sigmoids

- If

$$F(A) = \frac{1}{1+e^{-A}}$$

- then

$$\frac{\partial F(A)}{\partial A} = (1 - F(A))F(A)$$

So,

- Computer implementations of backprop, particularly with sigmoidal or linear units, or a mix of the two, are very easy to program…
- And 3 layer nets with BP should be very capable...

However...

- How do we represent inputs (how many input units)?
- How do we represent outputs (how many output units)?
- What's a good learning rate?
- **How many hidden neurons are necessary?**

Note:

- There are two basic applications of NNs we have discussed:
  - Categorization: which needs binary output that is appropriate for sigmoidal output units
  - Function Approximation: which needs output that is appropriate for linear output units

How to represent data:

- Any data set can be represented in sparse or dense format. . .
- In a neural net, a dense encoding means fewer weights.
- This means less computation.
- A sparse encoding means less "cross talk", which may lower training time.

What Learning Rate Should We Use?

- Clearly, the learning rate is critical to the progress of gradient search . . .
- However, for multilayer networks, there is no clear-cut guidance for setting learning rates
- Values between .001 and 10 are typical

The Momentum Method

- This is another heuristic scheme for "smoothing" gradient descent in backprop
- It updates the weights with a decaying average of previous weight updates
- The momentum method often results in fast convergence in incremental training, and can also help in batch training

How Many Hidden Units?

- Kolomogorov's theorem gives us a somewhat artificial bound for function approximating networks
- However, determining the actual number of units needed is in general, a key craft in designing neural nets...

In Function Approximation:

- The number of hidden units effects the complexity of the approximating function, therefore:
  - With too few nodes, we can have underfitting
  - With too many nodes, we can have overfitting

Other Backprop Tips:

- Random Restarts
- Noisy inputs
- Incrementally reducing error tolerances
- Adjust the number of hidden units
- These are all heuristics for problems that face all local optimization schemes
- Later in the class we will examine other, more global search processes

What to use as input:

- Sometimes the key to analyzing any data is appropriate pre-processing
- For Example:
  - Feature Extraction
  - Data Transformation (i.e. Fourier Transform)
  - Time Delayed Inputs

Feature Extraction:

- Consider extracting meaningful features from handwritten characters (Burr 1988)
- Apply a bilinear map to place the character in a standard grid
- Map points on the character to nearest regions of a template
- Assign indices to different template configurations to construct inputs
- 90-98% efficiency is obtained with this method

More Feature Extraction

- Consider distinguishing rocks and cylinders in sonar signals (Gorman & Sejnowski, 1985)
- Trained people are up to 93% accurate
- 100 training examples
- 60 expert-extracted features
- 12 hidden units yield 92% accuracy

Data Transformation

- Consider speech recognition: distinguishing b,d,e,v on a noisy line (Lang, 1987)
- People get 96% correct
- 700 sample of 150 millisecond data
- Apply 16 band FFT, over 12 time slices, for length 192 input vectors
- 4 output nodes, and one hidden unit, yield 93% accuracy

Time Delayed Inputs

- Consider mapping ASCII text to phonemes, to drive a voice generator (Sejnowski & Rosenberg, 1985)
- 29 characters to 21 phonemes a 5 stresses
- Input is seven characters, each with a one step delay, to map the middle character
- Trained with DECtalk, and expert system
- 80 hidden units

Next Time

- Self-organizing maps (something completely different)