Overview

- Multilayer feed-forward networks
- Gradient descent search
- Backpropagation learning rule
- Evaluation of backpropagation
- Applications of backpropagation

Research talk: Today 3pm, Zach 105B (by the Instructor) – *Neural Basis of Analogical Processing*

Multilayer Feed-Forward Networks

- Proposed in the 1950s
- Proper procedure for training the network came later (1969) and became popular in the 1980s: **back-propagation**

Back-Propagation Learning Rule

- Back-prop is basically a **gradient descent** algorithm.
- The tough problem: output layer has explicit error measure, so finding the error surface is trivial. However, for the hidden layers, how much error each connection eventually cause at the output nodes is hard to determine.
- Backpropagation determines how to distribute the **blame** to each connection.

Gradient Descent

- We want to minimize the total error $E$ by tweaking the network weights.
- $E$ depends on $W_i$, thus by adjusting $W_i$, you can reduce $E$.
- Figuring out how to simultaneously adjust weights $W_i$ for all $i$ at once is practically impossible, so use an iterative approach.
- A sensible way is to reduce $E$ with respect to one weight $W_i$ at a time, proportional to the gradient (or slope) at that point.
Gradient Descent (cont’d)

- For weight $W_i$ and error function $E$, to minimize $E$, $W_i$ should be changed according to $W_i \leftarrow W_i + \Delta W_i$:
  \[
  \Delta W_i = \alpha \times \left( -\frac{\partial E}{\partial W_i} \right),
  \]
  where $\alpha$ is the learning rate parameter.

- $E$ can be a function of many weights, thus the partial derivative is used in the above:
  \[
  E(W_1, W_2, ..., W_i, ... W_n, ...)
  \]

Hidden to Output Weights

- Error function
  \[
  E = \frac{1}{2} \sum_i (E_i)^2
  \]
  \[
  E_i = T_i - O_i
  \]
  \[
  = T_i - g\left( \sum_j W_j, i a_j \right),
  \]
  where $g(\cdot)$ is the sigmoid activation function, and $T_i$ the target.

Hidden to Output Weights (cont’d)

- For easier calculation later on, we can rewrite:
  \[
  \frac{\partial E}{\partial W_{j,i}} = -(T_i - O_i) \times g'(\sum_j W_j, i a_j) \times a_j
  \]
  \[
  = -a_j \times (T_i - O_i) \times g'(\sum_j W_j, i a_j)
  \]
  \[
  = -a_j \times \Delta_i
  \]
  
- It is easy to verify $g'(x) = g(x)(1 - g(x))$ from $g(x) = \frac{1}{1 + e^{-x}}$, so we can reuse the $g(x)$ value from the feed-forward phase in the feedback weight update.
Hidden to Output Weight Update

\[
\frac{\partial E}{\partial W_{kj,i}} = -a_j \times \Delta_i,
\]
we get the update rule:
\[
W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i,
\]
where \(\Delta_i = (T_i - O_i) \times g'(\sum_j W_{j,i} a_j).

However, this is too complex.

Use the chain rule for easier calculation of the partial derivative:
\[
\frac{\partial E}{\partial W_{k,j}} = \frac{\partial E}{\partial a_j} \times \frac{\partial a_j}{\partial W_{k,j}}
\]
where this is \(\Delta_i\)
\[
= -\sum_i \left( (T_i - O_i) g'(\sum_j W_{j,i} a_j) W_{j,i} \right) \times \left( \sum_k W_{k,j} I_k \right) I_k
\]
\[
= -\sum_i \left( \Delta_i W_{j,i} \right) g'(\sum_k W_{k,j} I_k) I_k
\]

From \(\frac{\partial E}{\partial W_{k,j}}\), we can rename \(\sum_i \left( \Delta_i W_{j,i} \right) g'(\sum_k W_{k,j} I_k)\) to be \(\Delta_j\), then the whole equation becomes:
\[
\frac{\partial E}{\partial W_{k,j}} = -\Delta_j I_k
\]
Thus the update rule becomes:
\[
W_{k,j} \leftarrow W_{k,j} + \alpha \times \Delta_j \times I_k
\]
Back-Propagation: Summary

- Weight update:
  \[ \Delta W_{x,y} \propto \Delta y \times \text{Input}_x \]

- The \( \Delta s \):
  \[ \Delta_x = \text{Error}_x \times g'(\text{WeightedSum}_x) \]

Thus, each node has its own \( \Delta \) and that is used to update the weights. These \( \Delta s \) are backpropaged for weight updates further below.

General Case: More Than 2 Layers

- In general, the same rule for back-propagating \( \Delta s \) apply for multiple layer networks with more than two layers.

- That is, \( \Delta \) for a deep hidden unit can be determined by the product of the weighted sum of feedback \( \Delta s \) and the first derivative of feedforward weighted sum at the current unit.

Backpropagation Algorithm

1. Pick \((\text{Input}, \text{Target})\) pair.

2. Using input, activate hidden and output layers through feed-forward activation.

3. At the output node, calculate the error \((T_i - O_i)\), and from that calculate the \( \Delta s \).

4. Update weights to the output layer, and backpropagate the \( \Delta s \).

5. Successively update hidden layer weights until input layer has been reached.

6. Repeat step 1–5 until the total error goes below a set threshold.

Technical Issues in Training

- Batch vs. online training
  - Batch: accumulate weight updates for one epoch, and then update
  - Online: immediately apply weight updates after one input-output pair.

- When to stop training
  - Training set: use for training
  - Validation set: determine when to stop
  - Test set: use for testing performance
Problems With Backprop

- Learning can be extremely slow: introduce momentum, etc.
- Network can be stuck in local minima: this is a common problem for any gradient-based method.

Other issues are: how to introduce new batches of data after the training has been completed.

Backprop Application

- Speech generation: NetTALK (Sejnowski and Rosenberg, 1987)
- Character recognition: LeCun (1989)
- Driving a car: ALVINN, etc.

and many other Engineering applications – control, etc.

Demo: NetTALK

- I want to I want to go to my grandmother’s ....
- friend, sent, around, not, red, soon, doubt, key, attention, lost

Key Points

- Basic concept of a multi-layer feed-forward network.
- How hidden units know how much error they caused.
- Backprop is a gradient descent algorithm.
- Drawbacks of backprop.
Next Time

- Self-organizing maps.