Chapter 4: Artificial Neural Networks

CS 536: Machine Learning
Littman (Wu, TA)

Administration

iCML-03: instructional Conference on Machine Learning
http://www.cs.rutgers.edu/~mlittman/courses/ml03/iCML03/

Weka assignment
http://www.cs.rutgers.edu/~mlittman/courses/ml03/hw1.pdf
Artificial Neural Networks

[Read Ch. 4]
[Review exercises 4.1, 4.2, 4.5, 4.9, 4.11]

• Threshold units
• Gradient descent
• Multilayer networks
• Backpropagation
• Hidden layer representations
• Example: Face Recognition
• Advanced topics

Connectionist Models

Consider humans:

• Neuron switching time ~ .001 second
• Number of neurons ~ $10^{10}$
• Connections per neuron ~ $10^{4-5}$
• Scene recognition time ~ .1 second
• 100 inference steps doesn't seem like enough
  • much parallel computation
Artificial Networks

Properties of artificial neural nets (ANNs):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

When to Consider ANNs

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant
ANNs: Example Uses

Examples:
- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction
- Backgammon [Tesauro]

ALVINN drives on highways
Perceptron

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Or, more succinctly: $$o(x) = \text{sgn}(w \cdot x)$$

Perceptron Decision Surface

A single unit can represent some useful functions
- What weights represent
  $$g(x_1, x_2) = \text{AND}(x_1, x_2)$$?
But some functions not representable
- e.g., not linearly separable
- Therefore, we'll want networks of these...
Perceptron training rule

\[ w_i \gets w_i + \delta w_i \]

where

\[ \delta w_i = \delta (t-o) x_i \]

Where:

- \( t = c(x) \) is target value
- \( o \) is perceptron output
- \( \delta \) is small constant (e.g., .1) called the learning rate (or step size)

Perceptron training rule

Can prove it will converge

- If training data is linearly separable
- and \( \delta \) sufficiently small
Gradient Descent

To understand, consider simpler linear unit, where

\[ o = w_0 + w_1x_1 + \ldots + w_nx_n \]

Let's learn \( w_i \)'s to minimize squared error

\[ E[w] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \]

Where \( D \) is set of training examples

Error Surface
Gradient Descent

Gradient
\[ \nabla E [w] = [\partial E/\partial w_0, \partial E/\partial w_1, \ldots, \partial E/\partial w_n] \]

Training rule:
\[ \Delta w = -\eta \nabla E [w] \]
in other words:
\[ \Delta w_i = -\eta \partial E/\partial w_i \]

Gradient of Error

\[ \partial E/\partial w_i \]
\[ = \partial/\partial w_i \frac{1}{2} d (t_d - o_d)^2 \]
\[ = \frac{1}{2} d \partial/\partial w_i (t_d - o_d)^2 \]
\[ = \frac{1}{2} d 2 (t_d - o_d) \partial/\partial w_i (t_d - o_d) \]
\[ = d (t_d - o_d) \partial/\partial w_i (t_d - w x_d) \]
\[ = d (t_d - o_d) (-x_{i,d}) \]
Gradient Descent Code

\texttt{Gradient-Descent}(training examples, $\square$)

\begin{itemize}
  \item Each training example is a pair of the form $<x, t>$, where $x$ is the vector of input values, and $t$ is the target output value. $\square$ is the learning rate (e.g., .05).
  \item Initialize each $w_i$ to some small random value
  \item Until the termination condition is met, Do
    \begin{itemize}
      \item Initialize each $\triangle w_i$ to zero.
      \item For each $<x, t>$ in training examples, Do
        \begin{itemize}
          \item Input the instance $x$ to the unit and compute the output $o$
          \item For each linear unit weight $w_i$, Do
            \begin{itemize}
              \item $w_i \triangleleft w_i + \square (t-o)x_i$
            \end{itemize}
        \end{itemize}
    \end{itemize}
\end{itemize}

Summary

Perceptron training rule will succeed if
\begin{itemize}
  \item Training examples are linearly separable
  \item Sufficiently small learning rate $\square$
\end{itemize}

Linear unit training uses gradient descent
\begin{itemize}
  \item Guaranteed to converge to hypothesis with minimum squared error
  \item Given sufficiently small learning rate $\square$
  \item Even when training data contains noise
  \item Even when training data not $H$ separable
**Stochastic Gradient Descent**

**Batch mode** Gradient Descent:
Do until satisfied
1. Compute the gradient $\bar{\nabla} E_D[w]$
2. $w \leftarrow w - \bar{\nabla} E_D[w]$

**Incremental mode** Gradient Descent:
Do until satisfied
- For each training example $d$ in $D$
  1. Compute the gradient $\bar{\nabla} E_d[w]$
  2. $w \leftarrow w - \bar{\nabla} E_d[w]$

**More Stochastic Grad. Desc.**

$E_D[w] = 1/2 \sum_{d \in D} (t_d - o_d)^2$

$E_d[w] = 1/2 (t_d - o_d)^2$

*Incremental Gradient Descent* can approximate *Batch Gradient Descent* arbitrarily closely if $\bar{\nabla}$ set small enough
Multilayer Networks

Decision Boundaries
Sigmoid Unit

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

Derivatives of Sigmoid Functions

Nice property:
\[
d\sigma(x)/dx = \sigma(x) (1-\sigma(x))
\]

We can derive gradient decent rules to train

- One sigmoid unit
- Multilayer networks of sigmoid units \( \blacklozenge \) Backpropagation
Error Gradient for Sigmoid

\[ \frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \partial_d (t_d-o_d)^2 \]
\[ = 1/2 \partial_d \frac{\partial}{\partial w_i} (t_d-o_d)^2 \]
\[ = 1/2 \partial_d 2 (t_d-o_d) \frac{\partial}{\partial w_i} (t_d-o_d) \]
\[ = \partial_d (t_d-o_d) (-\partial o_d/\partial w_i) \]
\[ = - \partial_d (t_d-o_d) (\partial o_d/\partial net_d \partial net_d/\partial w_i) \]

Even more...

But we know:
\[ \frac{\partial o_d}{\partial net_d} = \frac{\partial (o_d \cdot x_d)}{\partial w_i} = \frac{\partial w}{\partial x_d} \frac{\partial o_d}{\partial net_d} = o_d (1 - o_d) \]
\[ \frac{\partial net_d}{\partial w_i} = \frac{\partial (w \cdot x_d)}{\partial w_i} = x_{i,d} \]
So:
\[ \frac{\partial E}{\partial w_i} = - \partial_d (t_d-o_d) o_d (1 - o_d) x_{i,d} \]
Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, Do
  • For each training example, Do
    1. Input the training example to the network and compute the network outputs
    2. For each output unit $k$
       $\delta_k = o_k(1 - o_k)(t_k - o_k)$
    3. For each hidden unit $h$
       $\delta_h = o_h(1 - o_h) \sum_{k \text{ in outputs}} w_{h,k} \delta_d$
    4. Update each network weight $w_{ij}$
       $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$ where $\Delta w_{ij} = \delta_j x_{ij}$

More on Backpropagation

• Gradient descent over entire network weight vector
• Easily generalized to arbitrary directed graphs
• Will find a local, not necessarily global error minimum
  – In practice, often works well (can run multiple times)
More more

- Often include weight *momentum* 
  \[ w_{ij}(n) = \alpha_{ij}x_{ij} + \beta_{ij}w_{ij}(n-1) \]

- Minimizes error over training examples
  - Will it generalize well to subsequent examples?

- Training can take thousands of iterations
  - slow!

- Using network after training is very fast