Artificial Neural Networks

- 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 step does not seem like enough
- must use lots of parallel computation!

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 Neural Networks

- 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

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

ALVINN drives 70 mph on highways

Perceptron

\[ \sigma = \begin{cases} 1 & \text{if } \sum w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases} \]

Sometimes we will use simpler vector notation:

\[ a(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases} \]

Decision Surface of Perceptron

Represents some useful functions
- What weights represent \( g(x_1, x_2) = AND(x_1, x_2) \)?
- But some functions not representable
  - e.g., not linearly separable
  - therefore, we will want networks of these ...
**Perceptron Training Rule**

- \( w_j \leftarrow w_j + \Delta w_j \)
- \( \Delta w_j = \eta (t - o) x_j \)
- \( t = c(\mathbf{x}) \) is target value
- \( o \) is perceptron output
- \( \eta \) is small constant (e.g., .1)

Can prove it will converge
- If training data is linearly separable
- And \( \eta \) is sufficiently small

**Gradient Descent**

To understand, consider simple linear unit, where

\[ o = w_0 + w_1 x_1 + \ldots + w_n x_n \]

Idea: learn \( w_i \)'s that minimize the squared error

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

Where \( D \) is the set of training examples

**Gradient Descent**

\[
\nabla E[\mathbf{\hat{w}}] = \begin{bmatrix}
\frac{\partial E}{\partial w_0} \\
\frac{\partial E}{\partial w_1} \\
\vdots \\
\frac{\partial E}{\partial w_n}
\end{bmatrix}
\]

Training rule: \( \Delta w_j = -\eta \nabla E[\mathbf{\hat{w}}] \)

i.e.,

\[ \Delta w_j = -\eta \nabla \frac{E[\mathbf{\hat{w}}]}{\partial w_j} \]

**Gradient Descent**

\[
\frac{\partial E}{\partial w_j} = \sum_d \left( t_d - o_d \right) (x_{d,j} - \mathbf{\hat{w}} \cdot \mathbf{x}_d)
\]

**Gradient Descent**

**GRADIENT-DESCENT**(training examples, \( \eta \))

Each training example is a pair of the form \( \langle \mathbf{x}, t \rangle \), where \( \mathbf{x} \) is the vector of input values and \( t \) is the target output value. \( \eta \) is the learning rate (e.g., .05).

- Initialize each \( w_j \) to some small random value
- Until the termination condition is met, do
  - Initialize each \( \Delta w_j \) to zero
  - For each \( \langle \mathbf{x}, t \rangle \) in training examples, do
    * Input the instance \( \mathbf{x} \) and compute output \( o \)
    * For each linear unit weight \( w_j \), do
      \( \Delta w_j \leftarrow \Delta w_j + \eta (t(\mathbf{x}) - o) x_j \)
      * For each linear unit weight \( w_j \), do
        \( w_j \leftarrow w_j + \Delta w_j \)
Summary
Perceptron training rule guaranteed to succeed if
- Training examples are linearly separable
- Sufficiently small learning rate $\eta$

Linear unit training rule uses gradient descent
- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate $\eta$
- Even when training data contains noise
- Even when training data not separable by $H$

Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied:
1. Compute the gradient $\nabla_{\mathbf{w}} E_L[\mathbf{w}]$
2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E_L[\mathbf{w}]$

Incremental mode Gradient Descent:
Do until satisfied:
- For each training example $d$ in $D$
1. Compute the gradient $\nabla_d E_L[\mathbf{w}]$
2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_d E_L[\mathbf{w}]$

Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if $\eta$ made small enough

Multilayer Networks of Sigmoid Units

Multilayer Decision Space

Sigmoid Unit

The Sigmoid Function

Sort of a rounded step function
Unlike step function, can take derivative (makes learning possible)
Error Gradient for a Sigmoid Unit

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{1}{2} \sum_k (\sigma_j - o_j)^2$$

But we know:

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial \sigma_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial \omega_{ij}}$$

$$= \frac{1}{2} \sum_k (\sigma_j - o_j)^2 \sigma_j(1 - \sigma_j) \sum_i \frac{\partial \sigma_i}{\partial \text{net}_i}$$

$$= \frac{1}{2} \sum_k (\sigma_j - o_j)^2 \sigma_j(1 - \sigma_j) \sum_i \frac{\partial \sigma_i}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial \omega_{ij}}$$

$$= \frac{1}{2} \sum_k (\sigma_j - o_j)^2 \sigma_j(1 - \sigma_j) \sum_i \frac{\partial \sigma_i}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial \omega_{ij}}$$

Backpropagation Algorithm

Initialize all weights to small random numbers. Until satisfied, do

1. For each training example, do
   1.1. Input the training example and compute the outputs
   1.2. For each output unit $k$
      $$\delta_k^t \leftarrow \sigma_j(1 - \sigma_j) H_k - o_k$$
   1.3. For each hidden unit $h$
      $$\delta_h^t \leftarrow \sigma_j(1 - \sigma_j) \sum_k w_{kj} \delta_k$$
   1.4. Update each network weight $w_{ij}$
      $$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$
      where
      $$\Delta w_{ij} = \eta \delta_j x_{ij}$$

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)
- Often include weight momentum $\alpha$
  $$\Delta w_{ij}(n) = \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1)$$
- Minimizes error over training examples
- Will it generalize well to subsequent examples?
- Training can take thousands of iterations — slow!
  - Using network after training is fast

Learning Hidden Layer Representations

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>0.89</td>
</tr>
<tr>
<td>01000000</td>
<td>0.11</td>
</tr>
<tr>
<td>00100000</td>
<td>0.97</td>
</tr>
<tr>
<td>00010000</td>
<td>0.71</td>
</tr>
<tr>
<td>00001000</td>
<td>0.05</td>
</tr>
<tr>
<td>00000100</td>
<td>0.99</td>
</tr>
<tr>
<td>00000010</td>
<td>0.80</td>
</tr>
<tr>
<td>00000001</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Output Unit Error during Training

Sum of squared errors for each output unit
Hidden Unit Encoding

Input to Hidden Weights

Convergence of Backpropagation
Gradient descent to some local minimum
• Perhaps not global minimum
• Momentum can cause quicker convergence
• Stochastic gradient descent also results in faster convergence
• Can train multiple networks and get different results (using different initial weights)

Nature of convergence
• Initialize weights near zero
• Therefore, initial networks near-linear
• Increasingly non-linear functions as training progresses

Expressive Capabilities of ANNs
Boolean functions:
• Every Boolean function can be represented by network with a single hidden layer
• But that might require an exponential (in the number of inputs) hidden units

Continuous functions:
• Every bounded continuous function can be approximated with arbitrarily small error by a network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
• Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]

Overfitting in ANNs

Overfitting in ANNs

Expressive Capabilities of ANNs
Boolean functions:
• Every Boolean function can be represented by network with a single hidden layer
• But that might require an exponential (in the number of inputs) hidden units

Continuous functions:
• Every bounded continuous function can be approximated with arbitrarily small error by a network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
• Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]
Neural Nets for Face Recognition

90% accurate learning head pose, and recognizing 1-of-20 faces

Typical Input Images

Learned Network Weights

Learned Weights

Typical Input Images

Alternative Error Functions

Penalize large weights:

\[ E(\hat{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{output}} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ij}^2 \]

Train on target slopes as well as values:

\[ E(\hat{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{output}} \left( t_{kd} - o_{kd} \right)^2 + \mu \left( \frac{\partial t_{kd}}{\partial x_{d'}} - \frac{\partial o_{kd}}{\partial x_{d'}} \right)^2 \]

Tie together weights:

- e.g., in phoneme recognition

Recurrent Networks

Feedforward Network

Recurrent Network

unrolled in time

Neural Network Summary

- physiologically (neurons) inspired model
- powerful (accurate), slow, opaque (hard to understand resulting model)
- bias: preferential
  - based on gradient descent
  - finds local minimum
  - effect by initial conditions, parameters
- neural units
  - linear
  - linear threshold
  - sigmoid

Neural Network Summary (cont)

- gradient descent
  - convergence
- linear units
  - limitation: hyperplane decision surface
  - learning rule
- multilayer network
  - advantage: can have non-linear decision surface
  - backpropagation to learn
    - backprop learning rule
- learning issues
  - units used
Neural Network Summary (cont)

- learning issues (cont)
  - batch versus incremental (stochastic)
  - parameters
    - initial weights
    - learning rate
    - momentum
  - cost (error) function
    - sum of squared errors
    - can include penalty terms
- recurrent networks
  - simple
  - backpropagation through time