Today's Agenda

- Inductive learning
- Decision trees
- Break
- Bayesian learning
- Neural nets

Inductive Learning

Supervised Learning

- Given: Training examples \( (x_i, y_i) \) for some unknown function \( f \).
- Find: A good approximation to \( f \).

Example Applications

- Credit risk assessment
  - \( x \): Proportion of customer and proposed purchase
  - \( f(x) \): Approve purchase or not.
- Disease diagnosis
  - \( x \): Proportion of patient (symptoms, lab tests)
  - \( f(x) \): Disease or no, be recommended therapy
- Face recognition
  - \( x \): Bitmap picture of person’s face
  - \( f(x) \): Name of the person.
- Automatic steering
  - \( x \): Bitmap picture of road surface in front of car.
  - \( f(x) \): Degrees to turn the steering wheel.

Appropriate Applications for Supervised Learning

- Situations where there is no human expert
  - \( f(x) \): Predicted binding strength to HIV protease inhibitors
- Situations where humans can perform the task but can't describe how they do it.
  - \( x \): Bitmap picture of hand-written character
  - \( f(x) \): Act as role of the character
- Situations where the desired function is changing frequently
  - \( x \): Description of stock prices and trends for last 30 days.
  - \( f(x) \): Recommended stock transactions
- Situations where each user needs a customized function
  - \( x \): Invoicing e-mail message
  - \( f(x) \): Importance score for presenting to user (or deleting without presenting).

A Learning Problem

```
Example: x_1, x_2, x_3, x_4, y
1 0 0 1 0 0
2 0 1 0 0
3 0 0 1 1
4 1 0 1 1
5 0 1 1 0
6 1 1 0 0
7 0 1 0 1
```
Bias in Learning

- Hypothesis space
- Preferences over hypothesis
- Other prior knowledge

Without bias learning is impossible!
Decision Trees

Variable-sized hypothesis space
Number of possible hypotheses grows with depth of tree

Training Examples

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play/Tennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
How can this algorithm be viewed as a state-space search problem?

Entropy

\[ \text{Entropy}(S) = \text{expected number of bits needed to encode class } (\oplus \text{ or } \ominus) \text{ of randomly drawn member of } S \text{ (under optimal, shortest-length code)} \]

Why?

Information theory: optimal length code assigns
\[ \log_2 p \text{ bits to message having probability } p. \]

So, expected number of bits to encode \( \oplus \text{ or } \ominus \) of random member of \( S \):
\[ p_\oplus (-\log_2 p_\oplus) + p_\ominus (-\log_2 p_\ominus) \]
\[ \text{Entropy}(S) = -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus. \]

Information Gain

\[ \text{Gain}(S, A) = \text{expected reduction in entropy due to sorting on } A \]

\[ \text{Gain}(S, A) \equiv \text{Entropy}(S) - \sum_{\text{sorts}(A)} \frac{|S'|}{|S|} \text{Entropy}(S') \]

Training Examples

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Normal</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

Selecting the Next Attribute

Which attribute is the best classifier?
Hypothesis Space Search by ID3

- Hypothesis space is complete!
  - Target function surely in there...
- Outputs a single hypothesis (which one?)
  - Can’t play 20 questions...
- No backtracking
  - Local minima...
- Statistically-based search choices
  - Robust to noisy data...
- Inductive bias: approx. “prefer shortest tree”

Occam’s Razor

Why prefer short hypotheses?
Argument in favor:
- Fewer short hypotheses than long hypotheses.
  - A short hypothesis may be more likely to be a coincidence
- A short hypothesis may be more likely to be an underlying model
Argument opposed:
- There are many ways to define small sets of hypotheses
  - E.g., all trees with a prime number of nodes that use attributes beginning with “Z”
- What’s so special about small sets based on size of hypothesis?

Overfitting in Decision Trees

Consider adding noisy training example #15:
Sunny, Hot, Normal, Strong, Yes/No
What effect on earlier tree?

Overfitting

Consider error of hypothesis \( h \) over
- Training data: \( error_{train}(h) \)
- Entire distribution \( D \) of data: \( error_{opt}(h) \)

Hypothesis \( h \in H \) overfits training data if there is an alternative hypothesis \( h' \in H \) such that

\[
error_{train}(h) < error_{train}(h')
\]
and

\[
error_{opt}(h) > error_{opt}(h')
\]
Overfitting in Decision Tree Learning

Avoiding Overfitting

How can we avoid overfitting?
- stop growing when data split not statistically significant
- grow full tree, then post-prune

How to select “best” tree:
- Measure performance over training data
- Measure performance over separate validation data set
- MDL: minimize
  \[ \text{size}(\text{tree}) + \text{size}(<\text{misprediction rate}(\text{tree})) \]

Reduced-Error Pruning

Split data into training and validation set
Do until further pruning is harmful:
1. Evaluate impact on validation set of pruning each possible node (plus those below it)
2. greedily remove the one that most improves validation set accuracy
   - produces smallest version of most accurate subtree
   - What if data is limited?

Effect of Reduced-Error Pruning

Attributes with Costs

Consider
- medical diagnosis, BloodTest has cost $150
- robotics, Width from 1 ft has cost 23 sec.

How to learn a consistent tree with low expected cost?
One approach: replace gain by
- Tan and Schlimmer (1990)
  \[ \text{Gain}(S,A) \quad \text{Cost}(A)^w \]
- Nunez (1988)
  \[ \frac{\text{Gain}(S,A) - 1}{\text{Cost}(A) + 1}^w \]
  where \( w \in [0,1] \) determines importance of cost

Scaling Up

- ID3, C4.5, etc. assume data fits in main memory (OK for up to hundreds of thousands of examples)
- SPRINT, SLIQ: multiple sequential scans of data (OK for up to millions of examples)
- VFDT: at most one sequential scan (OK for up to billions of examples)
Ensembles of Classifiers

- Idea: instead of training one classifier (decision tree)
- Train \( k \) classifiers and let them vote
  - Only helps if classifiers disagree with each other
  - Trained on different data
  - Use different learning methods
- Amazing fact: can help a lot!

How voting helps

- Assume errors are independent
- Assume majority vote
- Probability majority is wrong = area under binomial dist

Assume errors are independent
Assume majority vote
Probability majority is wrong = area under binomial dist

* If individual area is 0.3
* Area under curve for \( \geq 1 \) wrong is 0.026
* Order of magnitude improvement!

Constructing Ensembles

- Bagging
  - Run classifier \( k \) times on \( m \) examples drawn randomly with replacement from the original set of \( n \) examples
- Cross-validated committees
  - Divide examples into \( k \) disjoint sets
  - Train on \( k \) sets corresponding to original minus \( 1/k \)-th
- Boosting (Shapire)
  - Maintain a probability distribution over set of training examples
  - On each iteration, use distribution to sample
  - Use error rate to modify distribution
  - Create harder and harder learning problems

Summary

- Inductive learning
- Decision trees
  - Representation
  - Tree growth
  - Heuristics
  - Overfitting and pruning
  - Scaling up
- Ensembles

Break!

Bayesian Learning
Bayes' Theorem

\[ P(h|D) = \frac{P(D|h)P(h)}{P(D)} \]

- \( P(h) \) = prior probability of hypothesis \( h \)
- \( P(D) \) = prior probability of training data \( D \)
- \( P(h|D) \) = probability of \( h \) given \( D \)
- \( P(D|h) \) = probability of \( D \) given \( h \)

Choosing Hypotheses

Find most probable hypothesis given training data

**Maximum a posteriori hypothesis** \( h_{MAP} \):

\[
h_{MAP} = \arg \max_{h \in H} P(h|D)
\]

\[
= \arg \max_{h \in H} \frac{P(D|h)P(h)}{P(D)}
\]

\[
= \arg \max_{h \in H} P(D|h)P(h)
\]

Assuming \( P(h) = P(h_i) \) we can further simplify, and choose the **Maximum likelihood (ML)** hypothesis

\[
h_{ML} = \arg \max_{h \in H} P(D|h_i)
\]

Example

Does patient have cancer or not?

A patient takes a lab test and the result comes back positive. The test returns a correct positive result in only 98% of the cases in which the disease is actually present, and a correct negative result in only 97% of the cases in which the disease is not present. Furthermore, 0.008 of the entire population have this cancer.

\[
P(\text{cancer}) =
\]
\[
P(\neg\text{cancer}) =
\]
\[
P(+|\text{cancer}) =
\]
\[
P(\neg+|\neg\text{cancer}) =
\]
\[
P(+|\neg\text{cancer}) =
\]
\[
P(\neg+|\text{cancer}) =
\]
\[
P(\text{cancer}+) =
\]
Brute-Force MAP Hypothesis Learner

1. For each hypothesis $h$ in $H$, calculate the posterior probability
   \[ P(h|D) = \frac{P(D|h)P(h)}{P(D)} \]
2. Output the hypothesis $h_{MAP}$ with the highest posterior probability
   \[ h_{MAP} = \arg\max_{h \in H} P(h|D) \]

Evolution of Posterior Probabilities

Learning a Real-Valued Function

Consider any real-valued target function $f$
Training examples $(x_i, d_i)$, where $d_i$ is noisy training value
- $d_i = f(x_i) + \epsilon_i$
- $\epsilon_i$ is random variable (noise) drawn independently for each $x_i$ according to some Gaussian distribution with mean $0$
Then the maximum likelihood hypothesis $h_{ML}$ is the one that minimizes the sum of squared errors:
   \[ h_{ML} = \arg\min_{h \in H} \sum_{i=1}^{m} (d_i - h(x_i))^2 \]

Most Probable Classification of New Instances

So far we’ve sought the most probable hypothesis given the data $D$ (i.e., $h_{MAP}$)
Given new instance $x$, what is its most probable classification? Not $h_{MAP}(x)$!
Consider:
- Three possible hypotheses:
  \[ P(h_1|D) = 0.4, \ P(h_2|D) = 0.3, \ P(h_3|D) = 0.3 \]
- Given new instance $x$,
  \[ h_1(x) = +, \ h_2(x) = -, \ h_3(x) = - \]
- What’s most probable classification of $x$?

Bayes Optimal Classifier

Bayes optimal classification:
   \[ \arg\max_{h \in H} \sum_{y \in \mathcal{Y}} P(y|h_i)P(h_i|D) \]
Example:
\[ P(h_1|D) = 0.4, \ \ P(-h_1) = 0, \ \ P(+h_1) = 1 \]
\[ P(h_2|D) = 0.3, \ \ P(-h_2) = 1, \ \ P(+h_2) = 0 \]
\[ P(h_3|D) = 0.3, \ \ P(-h_3) = 1, \ \ P(+h_3) = 0 \]
Classify instance $D$ as:
Naive Bayes Classifier

Assume target function \( f : X \rightarrow V \), where each instance \( x \) described by attributes \( (a_1, a_2, \ldots, a_n) \).

Most probable value of \( f(x) \) is:

\[
\hat{v}_{MAP} = \text{argmax}_{v_j \in V} P(v_j | a_1, a_2, \ldots, a_n) = \text{argmax}_{v_j \in V} \frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, a_2, \ldots, a_n)} = \text{argmax}_{v_j \in V} P(a_1, a_2, \ldots, a_n | v_j) P(v_j)
\]

Naive Bayes assumption:

\[
P(a_1, a_2, \ldots, a_n | v_j) = \prod_{i=1}^{n} P(a_i | v_j)
\]

which gives

Naive Bayes classifier:

\[
\hat{v}_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^{n} P(a_i | v_j)
\]

---

Naive Bayes Algorithm

NaiveBayes.Learn(examples)

For each target value \( v_j \)

\( \hat{P}(v_j) \) -- estimate \( P(v_j) \)

\( \hat{P}(a_i | v_j) \) -- estimate \( P(a_i | v_j) \)

Classify New Instance \( x \)

\[
\hat{v}_{NB} = \text{argmax}_{v_j \in V} \hat{P}(v_j) \prod_{i=1}^{n} \hat{P}(a_i | v_j)
\]

---

Naive Bayes: Example

Consider PlayTennis again, and new instance

\( \text{Outlook = sun, Temp = cool, Humid = high, Wind = strong} \)

Want to compute:

\[
\hat{v}_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^{n} P(a_i | v_j)
\]

\[
P(y) \cdot P(\text{sun} | y) \cdot P(\text{cool} | y) \cdot P(\text{high} | y) \cdot P(\text{strong} | y) = .005
\]

\[
P(n) \cdot P(\text{sun} | n) \cdot P(\text{cool} | n) \cdot P(\text{high} | n) \cdot P(\text{strong} | n) = .021
\]

\( \rightarrow v_{NB} = n \)

---

Learning to Classify Text

Why?

- Learn which news articles are of interest
- Learn to classify web pages by topic

Naive Bayes is among most effective algorithms

What attributes shall we use to represent text documents?

---

Learning to Classify Text

Target concept \( \text{Interesting?} : \text{Document} \rightarrow \{+, -\} \)

1. Represent each document by vector of words:
   one attribute per word position in document

2. Learning: Use training examples to estimate

   - \( P(+) \)
   - \( P(-) \)
   - \( P(\text{doc} | +) \)
   - \( P(\text{doc} | -) \)
Naive Bayes conditional independence assumption

\[ P(\text{doc}|v_j) = \prod_{i=1}^{\text{length(doc)}} P(a_i = w_k|v_j) \]

where \( P(a_i = w_k|v_j) \) is probability that word in position \( i \) is \( w_k \), given \( v_j \)

One more assumption:

\[ P(a_i = w_k|v_j) = P(a_m = w_k|v_j), \forall i, m \]

**Learn_Naive_Bayes_Text(Examples, V)**

1. Collect all words & tokens that occur in *Examples*
   - Vocabulary -- all distinct words & tokens in *Examples*
   - Compute all probabilities \( P(v_j) \) and \( P(w_k|v_j) \)
   - For each target value \( v_j \) in \( V \) do:
     - \( \text{docs}_j \) -- *Examples* for which the target value is \( v_j \)
     - \( P(v_j) = \frac{|\text{docs}_j|}{|\text{Examples}|} \)
     - \( \text{Text}_j \) -- concatenate all members of \( \text{docs}_j \)
     - \( n \) -- total number of words in \( \text{Text}_j \) (counting duplicate words multiple times)
     - for each word \( w_k \) in *Vocabulary*
       - \( n_k \) -- number of times word \( w_k \) occurs in \( \text{Text}_j \)
       - \( P(w_k|v_j) = \frac{n_k + 1}{n + |V\text{ocabulary}|} \)

**Classify_Naive_Bayes_Text(Doc)**

- positions -- all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return \( v_{NB} \), where

\[ v_{NB} = \arg \max_{v_j \in V} P(v_j) \prod_{i \in \text{positions}} P(a_i = w_k|v_j) \]

**Example: 20 Newsgroups**

Given 1000 training documents from each group

Learn to classify new documents according to which newsgroup it came from

- comp.graphics
- comp.os.ms-windows.misc
- comp.sys.mac.hardware
- comp.windows.x
- alt.atheism
- soc.religion.christian
- talk.politics.misc
- talk.politics.mideast
- talk.politics.guns
- misc.forsale
- rec.autos
- rec.apparel
- rec.booking
- rec.cooking
- rec.sport.baseball
- rec.sport.hockey
- sci.atmospheric
- sci.crypt
- sci.electronics
- sci.med
- sci.space

Naive Bayes: 89% classification accuracy

**Learning Curve for 20 Newsgroups**

Accuracy vs. Training set size (1/3 withheld for test)

**Learning Bayesian Networks**

Several variants of this learning task

- Network structure might be known or unknown
- Training examples might provide values of all network variables, or just some

If structure known and no missing values, it’s as easy as training a Naive Bayes classifier
The EM Algorithm

Suppose structure known, variables partially observable

E.g., observe ForestFire, Storm, BusTourGroup, Thunder, but not Lightning, Campfire...

Initialize parameters ignoring missing information
Repeat until convergence:

E step: Calculate expected vals of unobserved variables, assuming current parameter values

M step: Calculate new parameter values to maximize probability of data (observed & estimated)

Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples:</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>?</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialization: $P(B|A) = P(C|B) = P(A) = P(B|\sim A) = P(C|\sim B) = \ldots = 0$

E-step: $P(?) = P(B|A, \sim C) = P(A|B, \sim C) = \ldots = 0$

M-step: $P(B|A) = P(C|B) = P(A) = P(B|\sim A) = P(C|\sim B) = \ldots = 0$

E-step: $P(?) = 0$ (converged)

Unknown Structure

Search:
- Initial state: empty network, prior network
- Operators: Add arc, delete arc, reverse arc
- Evaluation: Posterior probability

Bayesian Learning: Summary

- Optimal prediction
- Naive Bayes learner
- Text classification
- Bayesian networks
- EM algorithm

Neural Networks

Preview

- Perceptrons
- Gradient descent
- Multilayer networks
- Backpropagation
Connectionist Models

Consider humans:
- Neuron switching time \( \sim .001 \text{ second} \)
- Number of neurons \( \sim 10^{10} \)
- Connections per neuron \( \sim 10^{4 \sim 5} \)
- Scene recognition time \( \sim .1 \text{ second} \)
- 100 inference steps doesn’t seem like enough

\( \Rightarrow \) Much parallel computation

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

Perceptron

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

Sometimes we’ll use simpler vector notation:
\[ o(\vec{z}) = \begin{cases} 
1 & \text{if } \vec{w} \cdot \vec{z} > 0 \\
-1 & \text{otherwise.}
\end{cases} \]

Decision Surface of a Perceptron

Represents some useful functions
- What weights represent \( g(x_1, x_2) = \text{AND}(x_1, x_2) \)?
- But some functions not representable
- All not linearly separable
- Therefore, we’ll want networks of these...
Perceptron Training Rule

\[ w_i \leftarrow w_i + \Delta w_i \]

where

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

Where:
- \( t = c(x) \) is target value
- \( o \) is perceptron output
- \( \eta \) is small constant (e.g., 0.1) called **learning rate**

Gradient Descent

To understand, consider **simpler linear unit**, where

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

Let’s learn \( w_i \)'s that minimize the squared error

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

Where \( D \) is set of training examples

Gradient:

\[ \nabla E[w] = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right] \]

Training rule:

\[ \Delta w = -\eta \nabla E[w] \]

I.e.:

\[ \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \]

Gradient Descent

\[ \frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d} (t_d - o_d)^2 \]

= \[ \frac{1}{2} \sum_{d} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \]

= \[ \frac{1}{2} \sum_{d} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \]

= \[ \sum_{d} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - w \cdot x_d) \]

\[ \frac{\partial E}{\partial w_i} = \sum_{d} (t_d - o_d)(-x_{i,d}) \]
Gradient Descent

\textsc{Gradient-Descent}\((\text{training\_examples}, \eta)\)

Initialize each \(w_i\) to some small random value

Until the termination condition is met, Do

1. Initialize each \(\Delta w_i\) to zero.
2. For each \((\vec{x}, t)\) in \text{training\_examples}, Do
   - Input instance \(\vec{x}\) to unit and compute output \(\hat{y}\)
   - For each linear unit weight \(w_i\), Do
     \[
     \Delta w_i \leftarrow \Delta w_i + \eta(t - \hat{y})x_i
     \]
3. For each linear unit weight \(w_i\), Do
   \[
   w_i \leftarrow w_i + \Delta w_i
   \]

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 \(H\)

Batch vs. Incremental Gradient Descent

**Batch Mode** Gradient Descent:

Do until convergence

1. Compute the gradient \(\nabla E_D(\vec{w})\)
2. \(\vec{w} \leftarrow \vec{w} - \eta \nabla E_D(\vec{w})\)

**Incremental Mode** Gradient Descent:

Do until convergence

For each training example \(d\) in \(D\)

1. Compute the gradient \(\nabla E_d(\vec{w})\)
2. \(\vec{w} \leftarrow \vec{w} - \eta \nabla E_d(\vec{w})\)

\[
E_D(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - \hat{y}_d)^2
\]

\[
E_d(\vec{w}) = \frac{1}{2} (t_d - \hat{y}_d)^2
\]

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

Multilayer Networks of Sigmoid Units

Sigmoid Unit

\[
s(\vec{z}) = \frac{1}{1 + e^{-\vec{z}}}
\]

\(s(x)\) is the sigmoid function

\[
\frac{d}{dx} s(x) = s(x)(1 - s(x))
\]
We can derive gradient descent rules to train

- One sigmoid unit
- Multilayer networks of sigmoid units → Backpropagation

But we know:

\[
\frac{\partial E}{\partial \text{net}_d} = \frac{\partial E}{\partial \text{net}_d} \cdot \frac{\partial \text{net}_d}{\partial o_d} = o_d(1 - o_d)
\]

\[
\frac{\partial \text{net}_d}{\partial w_i} = \frac{\partial (\sigma \cdot x_d)}{\partial w_i} = x_{i,d}
\]

So:

\[
\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d(1 - o_d) x_{i,d}
\]

Let: \( \delta_i = -\frac{\partial E}{\partial \text{net}_i} \)

\[
\delta_j = -\frac{\partial E}{\partial \text{net}_j} = o_j(1 - o_j) \sum_{k \in \text{output}} \delta_k w_{kj}
\]

\[
\text{Error Gradient for a Sigmoid Unit}
\]

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

\[
= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial \text{net}_i} (t_d - o_d)^2
\]

\[
= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial \text{net}_i} (t_d - o_d)
\]

\[
= \sum_{d \in D} (t_d - o_d) \left( \frac{\partial \text{net}_i}{\partial o_d} \right)
\]

\[
= -\sum_{d \in D} (t_d - o_d) \frac{\partial \text{net}_i}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial \text{net}_i}
\]

\[
\text{Backpropagation Algorithm}
\]

Initialize all weights to small random numbers

Until convergence, Do

1. Input it to network and compute network outputs
2. For each output unit \( k \)
   \( \delta_k = o_k(1 - o_k)(t_k - o_k) \)
3. For each hidden unit \( h \)
   \( \delta_h = o_h(1 - o_h) \sum_{k \in \text{output}} w_{hk} \delta_k \)
4. Update each network weight \( w_{ij} \)
   \( w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \)
   where \( \Delta w_{ij} = \eta \delta_j x_{i,j} \)

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 \( \alpha \)
  \( \Delta w_{ij}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{ij}(n-1) \)
- Minimizes error over training examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations → slow!
- Using network after training is very fast
Learning Hidden Layer Representations

A target function:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>10000000</td>
</tr>
<tr>
<td>01000000</td>
<td>01000000</td>
</tr>
<tr>
<td>00100000</td>
<td>00100000</td>
</tr>
<tr>
<td>00010000</td>
<td>00010000</td>
</tr>
<tr>
<td>00001000</td>
<td>00001000</td>
</tr>
<tr>
<td>00000100</td>
<td>00000100</td>
</tr>
<tr>
<td>00000010</td>
<td>00000010</td>
</tr>
<tr>
<td>00000001</td>
<td>00000001</td>
</tr>
</tbody>
</table>

Can this be learned?

Learned hidden layer representation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>.89 .04 .08</td>
<td>10000000</td>
</tr>
<tr>
<td>01000000</td>
<td>.01 .11 .88</td>
<td>01000000</td>
</tr>
<tr>
<td>00100000</td>
<td>.01 .97 .27</td>
<td>00100000</td>
</tr>
<tr>
<td>00010000</td>
<td>.99 .97 .71</td>
<td>00010000</td>
</tr>
<tr>
<td>00001000</td>
<td>.03 .05 .92</td>
<td>00001000</td>
</tr>
<tr>
<td>00000100</td>
<td>.22 .99 .99</td>
<td>00000100</td>
</tr>
<tr>
<td>00000010</td>
<td>.80 .01 .98</td>
<td>00000010</td>
</tr>
<tr>
<td>00000001</td>
<td>.90 .94 .01</td>
<td>00000001</td>
</tr>
</tbody>
</table>

Training

Sum of squared errors for each output unit

Training

Hidden unit encoding for input 01000000

Training

Weights from inputs to any hidden unit
Convergence of Backpropagation

Gradient descent to some local minimum
- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train multiple nets with different initial weights

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

Expressiveness of Neural Nets

Boolean functions:
- Every Boolean function can be represented by network with single hidden layer
- But might require exponential (in number of inputs) hidden units

Continuous functions:
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers

Overfitting in Neural Nets

Error versus weight updates (example 1)

Overfitting Avoidance

Penalize large weights:
\[ E(d) = \frac{1}{2} \sum_{x \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 + \gamma \sum_{j} \sigma_{ji}^2 \]

Train on target slopes as well as values:
\[ E(d) = \frac{1}{2} \sum_{x \in D} \sum_{k \in \text{outputs}} \left( (t_{kd} - o_{kd})^2 + \mu \sum_{i \in \text{inputs}} \left( \frac{\partial t_{kd}}{\partial o_d} \frac{\partial o_d}{\partial o_{ki}} \right)^2 \right) \]

Weight sharing
Early stopping

Neural Networks: Summary

- Perceptrons
- Gradient descent
- Multilayer networks
- Backpropagation