Machine Learning

CSE6740/CS7641/ISYE6740, Fall 2012

Neural Networks

Le Song

Lecture 7, September 11, 2012
Based on slides from Eric Xing, CMU

Reading: Chap. 5 CB
Learning highly non-linear functions

\[ f : X \rightarrow Y \]

- \( f \) might be non-linear function
- \( X \) (vector of) continuous and/or discrete vars
- \( Y \) (vector of) continuous and/or discrete vars

The XOR gate

Speech recognition
Perceptron and Neural Nets

- From biological neuron to artificial neuron (perceptron)

- Activation function

\[ X = \sum_{i=1}^{n} x_i w_i \]

\[ Y = \begin{cases} 
+1, & \text{if } X \geq \omega_0 \\
-1, & \text{if } X < \omega_0 
\end{cases} \]

- Artificial neuron networks
  - supervised learning
  - gradient descent
Connectionist Models

- Consider humans:
  - Neuron switching time
    \( \sim 0.001 \text{ second} \)
  - Number of neurons
    \( \sim 10^{10} \)
  - Connections per neuron
    \( \sim 10^{4-5} \)
  - Scene recognition time
    \( \sim 0.1 \text{ second} \)
  - 100 inference steps doesn't seem like enough
    \( \rightarrow \) much parallel computation

- Properties of artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes
Abdominal Pain Perceptron

Male: 1
Age: 20
Temp: 37
WBC: 10
Pain Intensity: 1
Pain Duration: 1

Adjustable weights

Appendicitis
Diverticulitis
Ulcer
Duodenal Perforated
Pain Non-specific
Cholecystitis
Obstruction
Pancreatitis
Small Bowel
Myocardial Infarction Network

Duration Pain | Intensity Pain | Elevation ECG: ST | Smoker | Age | Male
---|---|---|---|---|---
2 | 4 | 1 | 1 | 50 | 1

Myocardial Infarction “Probability” of MI
The "Driver" Network

ALVINN
[Pomerleau 1993]
Perceptrons

Input units

Cough  Headache

Weights

No disease  Pneumonia  Flu  Meningitis

Output units

Δ rule
change weights to decrease the error

- what we got
what we wanted

error
Output of unit $j$:

$$o_j = \frac{1}{1 + e^{-(a_j + \theta_j)}}$$

Input to unit $j$:

$$a_j = \sum w_{ij} a_i$$

Input to unit $i$:

$$a_i$$

measured value of variable $i$
Jargon Pseudo-Correspondence

- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

Logistic Regression Model (the sigmoid unit)

- Inputs
  - Age: 34
  - Gender: 1
  - Stage: 4

- Output
  - “Probability of being Alive”: 0.6

- Coefficients: $a, b, c$
- Independent variables: $x_1, x_2, x_3$

- Dependent variable: $p$ Prediction

- Equation: $p = \frac{1}{1 + e^{-\sum_{i=1}^{3} a_i x_i}}$
The perceptron learning algorithm

- Recall the nice property of sigmoid function
  \[
  \frac{d\sigma}{dt} = \sigma(1 - \sigma)
  \]

- Consider regression problem \( f: X \rightarrow Y \), for scalar \( Y \):
  \[
  y = f(x) + \epsilon
  \]

- Let’s maximize the conditional data likelihood
  \[
  \vec{w} = \arg \max_{\vec{w}} \ln \prod_i P(y_i|x_i; \vec{w})
  \]
  \[
  \vec{w} = \arg \min_{\vec{w}} \sum_i \frac{1}{2} (y_i - \hat{f}(x_i; \vec{w}))^2
  \]
Gradient Descent

\[ \nabla E[\vec{w}] = \begin{bmatrix} \frac{\partial E}{\partial w_0}, & \frac{\partial E}{\partial w_1}, & \cdots & \frac{\partial E}{\partial w_n} \end{bmatrix} \]

Training rule:

\[ \Delta \vec{w} = -\eta \nabla E[\vec{w}] \]

i.e.,

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

\[ \begin{align*}
\frac{\partial E[\vec{w}]}{\partial w_j} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum (t_d - o_d)^2 \\
&= \frac{\partial}{\partial w_i} \frac{1}{2} \sum (t_d - o_d)^2
\end{align*} \]

\( x_d = \text{input} \)
\( t_d = \text{target output} \)
\( o_d = \text{observed unit output} \)
\( w_i = \text{weight i} \)
The perceptron learning rules

\[
\frac{\partial E_D[\vec{w}]}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_d (t_d - o_d)^2
\]
\[
= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)
\]
\[
= \sum_d (t_d - o_d)(- \frac{\partial o_d}{\partial w_i})
\]
\[
= - \sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_i} \frac{\partial net_d}{\partial w_i}
\]
\[
= - \sum_d (t_d - o_d) o_d (1 - o_d) x_d^i
\]

Batch mode:
Do until converge:

1. compute gradient \( \nabla E_D[\vec{w}] \)
2. \( \vec{w} = \vec{w} - \eta \nabla E_D[\vec{w}] \)

Incremental mode:
Do until converge:

- For each training example \( d \) in \( D \)
  1. compute gradient \( \nabla E_d[\vec{w}] \)
  2. \( \vec{w} = \vec{w} - \eta \nabla E_d[\vec{w}] \)
where

\( \nabla E_d[\vec{w}] = -(t_d - o_d) o_d (1 - o_d) x_d \)
MLE vs MAP

- Maximum conditional likelihood estimate

\[
\hat{w} = \arg \max_{\tilde{w}} \ln \prod_{i} P(y_i|x_i; \tilde{w})
\]

\[
\tilde{w} \leftarrow \tilde{w} + \eta \sum_d (t_d - o_d) o_d (1 - o_d) \tilde{x}_d
\]

- Maximum a posteriori estimate

\[
\hat{w} = \arg \max_{\tilde{w}} \ln p(\tilde{w}) \prod_{i} P(y_i|x_i; \tilde{w})
\]

\[
\tilde{w} \leftarrow \tilde{w} + \eta \left( \sum_d (t_d - o_d) o_d (1 - o_d) \tilde{x}_d - \lambda \tilde{w} \right)
\]
What decision surface does a perceptron define?

\[ f(x_1w_1 + x_2w_2) = y \]

\[
\begin{align*}
&f(0w_1 + 0w_2) = 1 \\
&f(0w_1 + 1w_2) = 1 \\
&f(1w_1 + 0w_2) = 1 \\
&f(1w_1 + 1w_2) = 0
\end{align*}
\]

\[ f(a) = \begin{cases} 
1, & \text{for } a > \theta \\
0, & \text{for } a \leq \theta
\end{cases} \]

\[
\begin{array}{|c|c|}
\hline
w_1 & w_2 \\
\hline
0.20 & 0.35 \\
0.20 & 0.40 \\
0.25 & 0.30 \\
0.40 & 0.20 \\
\hline
\end{array}
\]

some possible values for \( w_1 \) and \( w_2 \)
What decision surface does a perceptron define?

NAND

\[
f(x_1w_1 + x_2w_2) = y
\]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>Z (color)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
f(a) = \begin{cases} 
1, & \text{for } a > \theta \\ 
0, & \text{for } a \leq \theta 
\end{cases}
\]

some possible values for \(w_1\) and \(w_2\)
What decision surface does a perceptron define?

NAND

\[
f(a) = \begin{cases} 
1, & \text{for } a > \theta \\
0, & \text{for } a \leq \theta 
\end{cases}
\]

a possible set of values for \((w_1, w_2, w_3, w_4, w_5, w_6)\):
\((0.6, -0.6, -0.7, 0.8, 1, 1)\)
Non Linear Separation

**Meningitis**
- No cough
- Headache

**Flu**
- Cough
- Headache

**No disease**
- No cough
- No headache

**Pneumonia**
- Cough
- No headache

- No treatment
- Treatment
“Combined logistic models”

Inputs

Age
- 34
- 0.6

Gender
- 2
- 0.1

Stage
- 4
- 0.7

Output

“Probability of being Alive”

0.6

Independent variables

Weights

Hidden Layer

Dependent variable

Prediction
**Independent variables**

- Age: 34
- Gender: 2
- Stage: 4

**Weights**

- Input to Hidden Layer: 0.2, 0.3, 0.2
- Hidden Layer to Output: 0.5, 0.8

**Prediction**

- “Probability of being Alive”: 0.6

**Output**

- 0.6
Independent variables

- Age: 34
- Gender: 1
- Stage: 4

Weights

- Age: .6
- Gender: .2
- Stage: .3

Hidden Layer

- Weight 1: .5
- Weight 2: .8

Output

- “Probability of being Alive”: 0.6

Prediction

Dependent variable
Not really, no target for hidden units...

Age 34
Gender 2
Stage 4

Weights

Independent variables
Weights
Hidden Layer
Weights
Dependent variable
Prediction

"Probability of being Alive"
Perceptrons

\[ \mathbf{\tilde{w}} \leftarrow \mathbf{\tilde{w}} + \eta \sum_d (t_d - o_d) o_d (1 - o_d) \mathbf{x}_d \]

\( \Delta \text{rule} \) change weights to decrease the error

- what we got
- what we wanted
- error

weights

Input units

Output units

Cough       Headache

No disease  Pneumonia  Flu  Meningitis
Hidden Units and Backpropagation

- what we got
- what we wanted
\[ \text{error} \]

\[ \Delta \text{ rule} \]

Input units

Hidden units

Output units

Impulse

Backpropagation
Backpropagation Algorithm

- Initialize all weights to small random numbers

Until convergence, Do

1. Input the training example to the network and compute the network outputs

1. For each output unit \( k \)

   \[
   \delta_k \leftarrow o^2_k (1 - o^2_k) (t - o^2_k)
   \]

2. For each hidden unit \( h \)

   \[
   \delta_h \leftarrow o^1_h (1 - o^1_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k
   \]

3. Undate each network weight \( w_{i,j} \)

   \[
   w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \quad \text{where} \quad \Delta w_{i,j} = \eta \delta_j x^j
   \]
More on Backpropagation

- It is doing 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 $\alpha$

$$\Delta w_{i,j}(t) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(t - 1)$$

- Minimizes error over training examples
  - Will it generalize well to subsequent testing examples?
- Training can take thousands of iterations, $\rightarrow$ very slow!
- Using network after training is very fast
Learning Hidden Layer Representation

- A network:

- A target function:

- Can this be learned?
Learning Hidden Layer Representation

- A network:

- Learned hidden layer representation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden Values</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000000</td>
<td>.89 .04 .08</td>
<td>10000000</td>
</tr>
<tr>
<td>010000000</td>
<td>.01 .11 .88</td>
<td>01000000</td>
</tr>
<tr>
<td>001000000</td>
<td>.01 .97 .27</td>
<td>00100000</td>
</tr>
<tr>
<td>000100000</td>
<td>.99 .97 .71</td>
<td>00010000</td>
</tr>
<tr>
<td>000010000</td>
<td>.03 .05 .02</td>
<td>00001000</td>
</tr>
<tr>
<td>000001000</td>
<td>.22 .99 .99</td>
<td>00000100</td>
</tr>
<tr>
<td>000000100</td>
<td>.80 .01 .98</td>
<td>00000010</td>
</tr>
<tr>
<td>000000010</td>
<td>.60 .94 .01</td>
<td>00000001</td>
</tr>
</tbody>
</table>
Training

Sum of squared errors for each output unit

- Training on different datasets
Training

Hidden unit encoding for input 01000000
Expressive Capabilities of ANNs

- **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 arbitrary small error, by 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].
Application: ANN for Face Reco.

- The model
- The learned hidden unit weights

Typical input images

http://www.cs.cmu.edu/~tom/faces.html
Regression vs. Neural Networks

(2^3 - 1) possible combinations

Y = a(X_1) + b(X_2) + c(X_3) + d(X_1X_2) + ...
Minimizing the Error

Error surface

initial error

negative derivative

final error

local minimum

\( w_{\text{initial}} \rightarrow w_{\text{trained}} \)

positive change
Overfitting in Neural Nets

Overfitted model

“Real” model

Overfitted model

CHD

0

age

cycles

error

holdout

training
Alternative Error Functions

- Penalize large weights:

\[ E[\bar{w}] \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{k,d} - o_{k,d})^2 + \gamma \sum_{i,j} w_{j,i}^2 \]

- Training on target slopes as well as values:

\[ E[\bar{w}] \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{k,d} - o_{k,d})^2 + \mu \sum_{j \in inputs} \left( \frac{\partial t_{k,d}}{\partial x_d^j} - \frac{\partial o_{k,d}}{\partial x_d^j} \right) \]

- Tie together weights
  - E.g., in phoneme recognition
Artificial neural networks – what you should know

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizing sum of squared training errors
  - Gives MLE estimates of network weights if we assume zero mean Gaussian noise on output values
- Minimizing sum of sq errors plus weight squared (regularization)
  - MAP estimates assuming weight priors are zero mean Gaussian
- Gradient descent as training procedure
  - How to derive your own gradient descent procedure
- Discover useful representations at hidden units
- Local minima is greatest problem
- Overfitting, regularization, early stopping