CS145: INTRODUCTION TO DATA MINING

6: Vector Data: Neural Network

Instructor: Yizhou Sun
yzsun@cs.ucla.edu

October 22, 2017
## Methods to Learn: Last Lecture

<table>
<thead>
<tr>
<th>Classification</th>
<th>Vector Data</th>
<th>Set Data</th>
<th>Sequence Data</th>
<th>Text Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logistic Regression; Decision Tree; KNN <strong>SVM</strong>; NN</td>
<td></td>
<td></td>
<td>Naïve Bayes for Text</td>
</tr>
<tr>
<td>Clustering</td>
<td>K-means; hierarchical clustering; DBSCAN; DBSCAN; Mixture Models</td>
<td></td>
<td></td>
<td>PLSA</td>
</tr>
<tr>
<td>Prediction</td>
<td><strong>Linear Regression</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GLM*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequent Pattern</td>
<td>Apriori; FP growth</td>
<td></td>
<td>GSP; PrefixSpan</td>
<td></td>
</tr>
<tr>
<td>Mining</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similarity Search</td>
<td></td>
<td></td>
<td></td>
<td>DTW</td>
</tr>
</tbody>
</table>
# Methods to Learn

<table>
<thead>
<tr>
<th></th>
<th>Vector Data</th>
<th>Set Data</th>
<th>Sequence Data</th>
<th>Text Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classification</strong></td>
<td>Logistic Regression; Decision Tree; KNN SVM; NN</td>
<td></td>
<td></td>
<td>Naïve Bayes for Text</td>
</tr>
<tr>
<td><strong>Clustering</strong></td>
<td>K-means; hierarchical clustering; DBSCAN; DBSCAN; Mixture Models</td>
<td></td>
<td></td>
<td>PLSA</td>
</tr>
<tr>
<td><strong>Prediction</strong></td>
<td>Linear Regression; GLM*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Frequent Pattern Mining</strong></td>
<td>Apriori; FP growth</td>
<td></td>
<td>GSP; PrefixSpan</td>
<td></td>
</tr>
<tr>
<td><strong>Similarity Search</strong></td>
<td></td>
<td></td>
<td></td>
<td>DTW</td>
</tr>
</tbody>
</table>
Neural Network

• Introduction
• Multi-Layer Feed-Forward Neural Network
• Summary
Artificial Neural Networks

• 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 -> parallel computation

• Artificial neural networks
  • Many neuron-like threshold switching units
  • Many weighted interconnections among units
  • Highly parallel, distributed process
  • Emphasis on tuning weights automatically
An $n$-dimensional input vector $\mathbf{x}$ is mapped into variable $y$ by means of the scalar product and a nonlinear function mapping.
Perceptron Training Rule

- If loss function is: \( l = \frac{1}{2} \sum_i (t_i - o_i)^2 \)

For each training data point \( x_i \):

\[
\mathbf{w}_{new} = \mathbf{w}_{old} + \eta (t_i - o_i)x_i
\]

- \( t \): target value (true value)
- \( o \): output value
- \( \eta \): learning rate (small constant)
Neural Network

• Introduction

• Multi-Layer Feed-Forward Neural Network

• Summary
A Multi-Layer Feed-Forward Neural Network

A two-layer network

\[ y = g(W^{(2)}h + b^{(2)}) \]

\[ h = f(W^{(1)}x + b^{(1)}) \]

Output vector

Output layer

Hidden layer

Input layer

Input vector: \( x \)

Nonlinear transformation, e.g. sigmoid transformation

Bias term

Weight matrix
Sigmoid Unit

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

Property: \[ \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \]

Will be used in learning
Activation functions

- **Step function**
  \[step_t(x) = \begin{cases} 
  1 & x > t \\
  0 & \text{otherwise}
  \end{cases}\]

- **Sign function**
  \[sign(x) = \begin{cases} 
  +1 & x \geq 0 \\
  -1 & \text{altrimenti}
  \end{cases}\]

- **Sigmoid function**
  \[sigmoide(x) = \frac{1}{1 + e^{-x}}\]
How A Multi-Layer Neural Network Works

• The **inputs** to the network correspond to the attributes measured for each training tuple

• Inputs are fed simultaneously into the units making up the **input layer**

• They are then weighted and fed simultaneously to a **hidden layer**

• The number of hidden layers is arbitrary, although usually only one

• The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction

• The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer

• From a math point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any continuous function
Defining a Network Topology

• Decide the **network topology**: Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*

• Normalize the **input** values for each attribute measured in the training tuples

• **Output**, if for classification and more than two classes, one output unit per class is used

• Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a different network topology or a different set of initial weights
Learning by Backpropagation

- Backpropagation: A neural network learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples
- Also referred to as connectionist learning due to the connections between units
Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value.

- For each training tuple, the weights are modified to minimize the loss function between the network's prediction and the actual target value, say mean squared error.

- Modifications are made in the “backwards” direction: from the output layer, through each hidden layer down to the first hidden layer, hence “backpropagation”.
Example of Loss Functions

- Hinge loss
- Logistic loss
- Cross-entropy loss
- Mean square error loss
- Mean absolute error loss
**A Special Case**

- **Activation function:** Sigmoid
  \[ O_j = \sigma(\sum_i w_{ij} O_i + \theta_j) \]

- **Loss function:** mean square error
  \[ J = \frac{1}{2} \sum_j (T_j - O_j)^2, \]

  \( T_j \): true value of output unit \( j \);
  \( O_j \): output value
Backpropagation Steps to Learn Weights

- Initialize weights to small random numbers, associated with biases
- Repeat until terminating condition meets
  - For each training example
    - Propagate the inputs forward (by applying activation function)
      - For a hidden or output layer unit $j$
        - Calculate net input: $I_j = \sum_i w_{ij}O_i + \theta_j$
        - Calculate output of unit $j$: $O_j = \sigma(I_j) = \frac{1}{1+e^{-I_j}}$
    - Backpropagate the error (by updating weights and biases)
      - For unit $j$ in output layer: $Err_j = O_j(1-O_j)(T_j-O_j)$
      - For unit $j$ in a hidden layer: $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$
      - Update weights: $w_{ij} = w_{ij} + \eta Err_j O_i$
      - Update bias: $\theta_j = \theta_j + \eta Err_j$
  - Terminating condition (when error is very small, etc.)
More on the output layer unit $j$

• Recall:

$$J = \frac{1}{2} \sum_j (T_j - o_j)^2, \quad o_j = \sigma(\sum_i w_{ij} o_i + \theta_j)$$

• Chain rule of first derivation

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial O_j} \frac{\partial O_j}{\partial w_{ij}} = -(T_j - o_j) o_j (1 - o_j) o_i$$

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial O_j} \frac{\partial O_j}{\partial \theta_j} = -(T_j - o_j) o_j (1 - o_j)$$

Denoted as $\text{Err}_j$!
More on the hidden layer unit $j$

- Let $i, j, k$ denote units in input layer, hidden layer, and output layer, respectively.

\[ J = \frac{1}{2} \sum_k (T_k - o_k)^2, \quad o_k = \sigma \left( \sum_j w_{jk} o_j + \theta_k \right), \quad o_j = \sigma \left( \sum_i w_{ij} o_i + \theta_j \right) \]

- Chain rule of first derivation

\[
\frac{\partial J}{\partial w_{ij}} = \sum_k \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}}
\]

\[ = - \sum_k (T_k - o_k) o_k (1 - o_k) w_{jk} o_j (1 - o_j) o_i \]

Note: $\frac{\partial J}{\partial o_k} = -(T_k - o_k), \frac{\partial o_k}{\partial o_j} = o_k (1 - o_k) w_{jk}, \frac{\partial o_j}{\partial w_{ij}} = o_j (1 - o_j) o_i$

\[
\frac{\partial J}{\partial \theta_j} = \sum_k \frac{\partial J}{\partial o_k} \frac{\partial o_k}{\partial o_j} \frac{\partial o_j}{\partial \theta_j} = -Err_j
\]
A multilayer feed-forward neural network

Initial Input, weight, and bias values
Example

• Input forward:

Table 9.2: The net input and output calculations.

<table>
<thead>
<tr>
<th>Unit j</th>
<th>Net input, $I_j$</th>
<th>Output, $O_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$0.2 + 0 - 0.5 - 0.4 = -0.7$</td>
<td>$1/(1 + e^{0.7}) = 0.332$</td>
</tr>
<tr>
<td>5</td>
<td>$-0.3 + 0 + 0.2 + 0.2 = 0.1$</td>
<td>$1/(1 + e^{-0.1}) = 0.525$</td>
</tr>
<tr>
<td>6</td>
<td>$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$</td>
<td>$1/(1 + e^{0.105}) = 0.474$</td>
</tr>
</tbody>
</table>

• Error backpropagation and weight update:

Table 9.3: Calculation of the error at each node.

<table>
<thead>
<tr>
<th>Unit j</th>
<th>$\text{Err}_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$</td>
</tr>
<tr>
<td>5</td>
<td>$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$</td>
</tr>
<tr>
<td>4</td>
<td>$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$</td>
</tr>
</tbody>
</table>

assuming $T_6 = 1$

Table 9.4: Calculations for weight and bias updating.

<table>
<thead>
<tr>
<th>Weight or bias</th>
<th>New value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{46}$</td>
<td>$-0.3 + (0.9)(0.1311)(0.332) = -0.261$</td>
</tr>
<tr>
<td>$w_{56}$</td>
<td>$-0.2 + (0.9)(0.1311)(0.525) = -0.138$</td>
</tr>
<tr>
<td>$w_{14}$</td>
<td>$0.2 + (0.9)(-0.0087)(1) = 0.192$</td>
</tr>
<tr>
<td>$w_{15}$</td>
<td>$-0.3 + (0.9)(-0.0065)(1) = -0.306$</td>
</tr>
<tr>
<td>$w_{24}$</td>
<td>$0.4 + (0.9)(-0.0087)(0) = 0.4$</td>
</tr>
<tr>
<td>$w_{25}$</td>
<td>$0.1 + (0.9)(-0.0065)(0) = 0.1$</td>
</tr>
<tr>
<td>$w_{34}$</td>
<td>$-0.5 + (0.9)(-0.0087)(1) = -0.508$</td>
</tr>
<tr>
<td>$w_{35}$</td>
<td>$0.2 + (0.9)(-0.0065)(1) = 0.194$</td>
</tr>
<tr>
<td>$\theta_{6}$</td>
<td>$0.1 + (0.9)(0.1311) = 0.218$</td>
</tr>
<tr>
<td>$\theta_{5}$</td>
<td>$0.2 + (0.9)(-0.0065) = 0.194$</td>
</tr>
<tr>
<td>$\theta_{4}$</td>
<td>$-0.4 + (0.9)(-0.0087) = -0.408$</td>
</tr>
</tbody>
</table>
Efficiency and Interpretability

- **Efficiency** of backpropagation: Each iteration through the training set takes $O(|D| \times w)$, with $|D|$ tuples and $w$ weights, but # of iterations can be exponential to $n$, the number of inputs, in worst case.

- For easier comprehension: **Rule extraction** by network pruning*
  - Simplify the network structure by removing weighted links that have the least effect on the trained network.
  - Then perform link, unit, or activation value clustering.
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers.

- **Sensitivity analysis**: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules.
  - E.g., If $x$ decreases 5% then $y$ increases 8%
Neural Network as a Classifier

• Weakness
  • Long training time
  • Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  • Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network

• Strength
  • High tolerance to noisy data
  • Successful on an array of real-world data, e.g., hand-written letters
  • Algorithms are inherently parallel
  • Techniques have recently been developed for the extraction of rules from trained neural networks
  • Deep neural network is powerful
Digits Recognition Example

• Obtain sequence of digits by segmentation

\[ \begin{array}{cccccc}
S & 0 & 4 & 1 & 9 & 2 \\
\end{array} \]

• Recognition (our focus)

\[ \begin{array}{cccccc}
5 & 0 & 4 & 1 & 9 & 2 \\
\end{array} \]

\[ \begin{array}{cccccc}
5 & \rightarrow & 5 \\
\end{array} \]
Digits Recognition Example

• The architecture of the used neural network

• What each neurons are doing?

Input image ➔ Activated neurons detecting image parts ➔ Predicted number 0
Towards Deep Learning*

Deep neural network

(Diagram of a deep neural network with labeled layers: input layer, hidden layer 1, hidden layer 2, hidden layer 3, output layer.)
Deep Learning References

- http://neuralnetworksanddeeplearning.com/
- http://www.deeplearningbook.org/
Neural Network

• Introduction
• Multi-Layer Feed-Forward Neural Network
• Summary
Summary

• Neural Network
  • Feed-forward neural networks; activation function; loss function; backpropagation