Applied inductive learning - Lecture 5

Louis Wehenkel

Department of Electrical Engineering and Computer Science
University of Liège

Montefiore - Liège - October 17, 2005

Find slides: http://montefiore.ulg.ac.be/~lwh/AIA/
Batch-mode vs Online-mode Supervised Learning

Motivations for Artificial Neural Networks

Linear ANN Models
   Single neuron models
   Single layer models

Nonlinear ANN Models
   Representation capacity of multilayer perceptrons
   Learning algorithms for multilayer perceptrons
   Other nonlinear models and learning algorithms

Wrap up discussion
Objects (or observations): $LS = \{o_1, \ldots, o_N\}$

Attribute vector: $a^i = (a_1(o_i), \ldots, a_n(o_i))^T$, $\forall i = 1, \ldots, N.$

Outputs: $y^i = y(o_i)$ or $c^i = c(o_i)$, $\forall i = 1, \ldots, N.$

LS Table

<table>
<thead>
<tr>
<th>$o$</th>
<th>$a_1(o)$</th>
<th>$a_2(o)$</th>
<th>$\ldots$</th>
<th>$a_n(o)$</th>
<th>$y(o)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_1^1$</td>
<td>$a_2^1$</td>
<td>$\ldots$</td>
<td>$a_n^1$</td>
<td>$y^1$</td>
</tr>
<tr>
<td>2</td>
<td>$a_1^2$</td>
<td>$a_2^2$</td>
<td>$\ldots$</td>
<td>$a_n^2$</td>
<td>$y^2$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$N$</td>
<td>$a_1^N$</td>
<td>$a_2^N$</td>
<td>$\ldots$</td>
<td>$a_n^N$</td>
<td>$y^N$</td>
</tr>
</tbody>
</table>

Focus for this lecture on numerical inputs, and numerical outputs (classes will be encoded numerically if needed).
In batch-mode
- Samples provided and processed together to construct model
- Need to store samples (not the model)
- Classical approach for data mining

In online-mode
- Samples provided and processed one by one to update model
- Need to store the model (not the samples)
- Classical approach for adaptive systems

But both approaches can be adapted to handle both contexts
- Samples available together can be exploited one by one
- Samples provided one by one can be stored and then exploited together
Motivations for Artificial Neural Networks

Intuition: biological brain can learn, so let’s try to be inspired by it to build learning algorithms.

- Starting point: single neuron models
  - perceptron, LTU and STU for linear supervised learning
  - online (biologically plausible) learning algorithms
- Complexify: multilayer preceptrons
  - flexible models for non-linear supervised learning
  - universal approximation property
  - iterative training algorithms based on non-linear optimization
- ...other neural network models of importance
Single neuron models

The biological neuron:

Human brain: $10^{11}$ neurons, each with $10^4$ synapses
Memory (knowledge): stored in the synapses
Hard threshold unit...

A simple (simplistic) mathematical model of the biological neuron

\[ g(a(o)) = \text{sgn} \left\{ w_0 + w^T a(o) \right\} \]
\[ = \text{sgn} \left\{ w'^T a'(o) \right\} \]

Parameters to adapt to problem: \( w' \)
...and the perceptron learning algorithm

1. For binary classification: \( c(o) = \pm 1 \).
2. Start with an arbitrary initial weight vector, e.g. \( w'_0 = 0 \).
3. Consider the objects of the LS in a cyclic or random sequence.
4. Let \( o_i \) be the object at step \( i \), \( c(o_i) \) its class and \( a(o_i) \) its attribute vector.
5. Adjust the weight by using the following correction rule,

\[
w'_{i+1} = w'_i + \eta_i \left( c(o_i) - g_i(a(o_i)) \right) a'(o_i).
\]

- \( w'_i \) changes only if \( o_i \) is not correctly classified
- it is changed in the right direction (\( \eta_i > 0 \) is the learning rate)
- at any stage, \( w'_i \) is a linear combination of the \( a(o_i) \) vectors
Geometrical view of update equation

\[ a_1 = \eta a_0 + w_i \]

\[ a_0 = 1 \]

Louis Wehenkel
Geometrical view of update equation

\[ a_0 = 1 \]

\[ a(o), c(o) = +1 \]
Geometrical view of update equation

\[ a_0 = 1 \]

\[ a(o), c(o) = +1 \]

Updated hyperplane

\[ 2\eta a(o) \]

\[ w_i \]
The input/output function $g(a)$ of such a device is computed by

$$
g(a(o)) \triangleq f(w_0 + w^T a(o)) = f(w'^T a'(o))$$

where the activation function $f(\cdot)$ is assumed to be differentiable. Classical examples of activation functions are the sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)},$$

and the hyperbolic tangent

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)},$$
... and gradient descent

Find vector $\mathbf{w}'^T = (w_0, \mathbf{w}^T)$ minimizing the square error (TSE)

$$TSE(LS, \mathbf{w}') = \sum_{o \in LS} (g(a(o)) - y(o))^2 = \sum_{o \in LS} \left( f(\mathbf{w}'^T a'(o)) - y(o) \right)^2.$$ 

The gradient with respect to $\mathbf{w}'$ is computed by

$$\nabla_{\mathbf{w}'} TSE(LS, \mathbf{w}') = 2 \sum_{o \in LS} (g(a(o)) - y(o)) f'(\mathbf{w}'^T a'(o))a'(o),$$

where $f'(\cdot)$ denotes the derivative of the activation function $f(\cdot)$.

The gradient descent method works by iteratively changing the weight vector by a term proportional to $-\nabla_{\mathbf{w}'} TSE(LS, \mathbf{w}')$. 
... and stochastic online gradient descent

**Fixed** step gradient descent in online-mode:

1. For binary classification: \( c(o) = \pm 1 \).
2. Start with an arbitrary initial weight vector, e.g. \( w'_0 = 0 \).
3. Consider the objects of the LS in a cyclic or random sequence.
4. Let \( o_i \) be the object at step \( i \), \( c(o_i) \) its class and \( a(o_i) \) its attribute vector.
5. Adjust the weight by using the following correction rule,

\[
\begin{align*}
\mathbf{w}'_{i+1} &= \mathbf{w}'_i - \eta_i \nabla_{\mathbf{w}'} SE(o_i, \mathbf{w}'_i) \\
&= \mathbf{w}'_i + 2\eta_i [c(o_i) - g_i(a(o_i))] f'(\mathbf{w}'_i^T a'(o_i))a'(o),
\end{align*}
\]

\((SE(o, w')\) is the contribution of object \( o \) in \( TSE(LS, w')\).\)
Theoretical properties

- Convergence of the perceptron learning algorithm
  - If $LS$ is linearly separable: converges in a finite number of steps.
  - Otherwise: converges with infinite number of steps, if $\eta_i \to 0$.

- Convergence of the online or batch gradient descent algorithm
  - if $\eta_i \to 0$ (slowly), and infinite number of steps, same solution
  - if $f(\cdot)$ linear, finds same solution as linear regression

NB: slow $\eta_i \to 0$ means

- $\lim_{m \to \infty} \sum_{i=1}^{m} \eta_i = +\infty$
- $\lim_{m \to \infty} \sum_{i=1}^{m} \eta_i^2 < +\infty$
Single layer models

- NB: the number of layers, refers to the number of connection layers.
- A single layer model is composed of several perceptrons.
- These work completely independently.
- They can be trained independently.
- Interpretations:
  - hyperplanes in Euclidean space
  - boolean operators (e.g. Logical AND, OR etc.)
Representation capacity of multilayer perceptrons

- Classification:
  - geometrical insight in the representation capacity
  - threelayer hard threshold units
    - First layer: can define a collection of hyperplanes/semiplanes
    - Second layer: can define arbitrary intersections of semiplanes
    - Third layer: can define arbitrary union of intersections of semi-planes
  - Conclusion: with a sufficient number of units, very complex regions can be described

- Regression:
  - function approximation insight
  - twolayer soft threshold units
    - One-dimensional input space illustration
    - First layer, defines $K$ offset and scale parameters $\alpha_i, \beta_i, i = 1 \ldots K$: responses $f(\alpha_i x + \beta_i)$
    - Second (linear layer): $\hat{y}(x) = b_0 + \sum_{i=1}^{K} b_i f(\alpha_i x + \beta_i)$
Learning algorithms for multilayer perceptrons

First assume that only the last layer has to be learned.
Then assume also the other layers need to be learned.
Backpropagation of derivatives
Second order optimization methods
Radial basis functions etc.
Relation with tree-based methods, linear models and KNN
Overfitting in multilayer perceptrons