Multilayer Perceptron

Hong Chang

Institute of Computing Technology, Chinese Academy of Sciences

Machine Learning Methods (Fall 2012)
Outline I

1. Introduction
2. Single Perceptron
3. Boolean Function Learning
4. Multilayer Perceptron
Artificial Neural Network

- **Cognitive scientists** and neuroscientists: the aim is to understand the functioning of the brain by building models of the natural neural networks.
- **Machine learning** researchers: the aim (more pragmatic) is to build better computer systems based on inspirations from studying the brain.
- A human brain has:
  - Large number \(10^{11}\) of neurons as processing units
  - Large number \(10^4\) of synapses per neuron as memory units
  - Parallel processing capabilities
  - Distributed computation/memory
  - High robustness to noise and failure
- **Artificial neural networks (ANN)** mimic some characteristics of the human brain, especially with regard to the computational aspects.
The output $f$ is weighted sum of the inputs $\mathbf{x} = (x_0, x_1, \ldots, x_d)^T$:

$$f = \sum_{j=1}^{d} w_j x_j + w_0 = \mathbf{w}^T \mathbf{x}$$

where $x_0$ is a special bias unit with $x_0 = 1$ and $\mathbf{w} = (w_0, w_1, \ldots, w_d)^T$ are called the connection weights.
What a Perceptron Does

- **Regression vs. classification:**

- To implement a linear discriminant function, we need the threshold function
  
  \[ s(a) = \begin{cases} 
  1 & \text{if } a > 0 \\
  0 & \text{otherwise} 
  \end{cases} \]

  to define the following decision rule:

  \[
  \text{Choose} \begin{cases} 
  C_1 & \text{if } s(w^T x) = 1 \\
  C_2 & \text{otherwise} 
  \end{cases}
  \]
Instead of using the threshold function to give a discrete output in \{0, 1\}, we may use the sigmoid function

\[ \text{sigmoid}(a) = \frac{1}{1 + \exp(-a)} \]

to give a continuous output in (0, 1):

\[ f = \text{sigmoid}(\mathbf{w}^T \mathbf{x}) \]

- The output may be interpreted as the posterior probability that the input \( \mathbf{x} \) belongs to \( C_1 \).
- Perceptron is cosmetically similar to logistic regression and least squares linear regression.
$K > 2$ Outputs

- $K$ perceptrons, each with a weight vector $\mathbf{w}_k$:

$$f_k = \sum_{j=1}^{d} w_{kj} x_j + w_0 = \mathbf{w}_k^T \mathbf{x} \text{ or } \mathbf{f} = \mathbf{W} \mathbf{x}$$

where $w_{kj}$ is the weight from input $x_j$ to output $y_k$, and each row of the $K \times (d + 1)$ matrix $\mathbf{W}$ is the weight vector of one perceptron.

- $\mathbf{W}$ performs a linear transformation from a $d$-dimensional space to a $K$-dimensional space.
Classification

- **Classification:**
  
  Choose $C_k$ if $f_k = \max_i f_i$

- If we need the *posterior probabilities* as well, we can use softmax to define $f_k$ as:

$$f_k = \frac{\exp(w_k^T x)}{\sum_{i=1}^K \exp(w_i^T x)}$$
Perceptron Learning

- **Learning mode:**
  - **Online learning:** instances seen one by one.
  - **Batch learning:** whole sample seen all at once.

- **Advantages of online learning:**
  - No need to store the whole sample.
  - Can adapt to changes in sample distribution over time.
  - Can adapt to physical changes in system components.

- The error function is not defined over the whole sample but on individual instances.

- Starting from randomly initialized weights, the parameters are adjusted a little bit at each iteration to reduce the error without forgetting what was learned previously.
Stochastic Gradient Descent

- If the error function is differentiable, gradient descent may be applied at each iteration to reduce the error.
- Gradient descent for online learning is also known as stochastic gradient descent.
- For regression, the error on a single instance \((x^{(i)}, y^{(i)})\):

\[
E(w|x^{(i)}, y^{(i)}) = \frac{1}{2}(y^{(i)} - f^{(i)})^2 = \frac{1}{2}(y^{(i)} - (w^T x^{(i)}))^2
\]

which gives the following online update rule:

\[
\Delta w^{(i)} = \eta (y^{(i)} - f^{(i)}) x^{(i)}
\]

where \(\eta\) is a step size parameter which is decreased gradually in time for convergence.
Binary Classification

- Logistic discrimination for a single instance \((x^{(i)}, y^{(i)})\), where \(y^{(i)} = 1\) if \(x^{(i)} \in C_1\) and \(y^{(i)} = 0\) if \(x^{(i)} \in C_2\), gives the output:

\[
f^{(i)} = \text{sigmoid}(w^T x^{(i)})
\]

- Likelihood:

\[
L = (f^{(i)})y^{(i)}(1 - f^{(i)})^{1 - y^{(i)}}
\]

- Cross-entropy error function:

\[
E(w|x^{(i)}, y^{(i)}) = -\log L = -y^{(i)} \log f^{(i)} - (1 - y^{(i)}) \log(1 - f^{(i)})
\]

- Online update rule:

\[
\Delta w_j^{(i)} = \eta(y^{(i)} - f^{(i)})x_j^{(i)}
\]
**K > 2 Classes**

- Softmax for a single instance \((x^{(i)}, y^{(i)})\), where \(y_k^{(i)} = 1\) if \(x^{(i)} \in C_k\) and 0 otherwise, gives the outputs:

\[
    f_k^{(i)} = \frac{\exp(w_k^T x^{(i)})}{\sum_l \exp(w_l^T x^{(i)})}
\]

- Likelihood:

\[
    L = \prod_k (f_k^{(i)})^{y_k^{(i)}}
\]

- Cross-entropy error function:

\[
    \mathcal{E}(\{w_k\}|x^{(i)}, y^{(i)}) = - \log L = - \sum_k y_k^{(i)} \log f_k^{(i)}
\]

- Online update rule:

\[
    \triangle w_{kj}^{(i)} = \eta (y_k^{(i)} - f_k^{(i)}) x_j^{(i)}
\]
Perceptron Learning Algorithm

For $k = 1, \ldots, K$
    For $j = 0, \ldots, d$
        \[ w_{kj} \leftarrow \text{rand}(-0.01, 0.01) \]

Repeat
    For all $(x^t, y^t) \in \mathcal{X}$ in random order
        For $k = 1, \ldots, K$
            \[ o_k \leftarrow 0 \]
            For $j = 0, \ldots, d$
                \[ o_k \leftarrow o_k + w_{kj}x_j^t \]
        For $k = 1, \ldots, K$
            \[ f_k \leftarrow \exp(o_k)/\sum_l \exp(o_l) \]
            For $k = 1, \ldots, K$
                For $j = 0, \ldots, d$
                    \[ w_{jk} \leftarrow w_{jk} + \eta(y_k^t - f_k)x_j^t \]

Until convergence

- **Update**=Learning Factor $\times$ (Desired Output - Actual Output) $\times$ Input
Online Learning Error of Perceptron

Theorem (Block, 1962, and Novikoff, 1962)

Let a sequence of examples \((x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\) be given. Suppose that \(\|x^{(i)}\| \leq M\) for all \(i\), and further that there exists a unit-length vector \(u (\|u\|_2 = 1)\) such that \(y^{(i)} \cdot (u^T x^{(i)}) \geq \gamma\) for all examples in the sequence (i.e., \(u^T x^{(i)} \geq \gamma\) if \(y^{(i)} = 1\) and \(u^T x^{(i)} \leq -\gamma\) if \(y^{(i)} = -1\), so that \(u\) separates the data with a margin of at least \(\gamma\)). Then the total number of mistakes that the perceptron algorithm makes on this sequence is at most \((M/\gamma)^2\).

- Given a training example \((x, y) (y \in \{-1, 1\})\), the perceptron learning rule updates the parameters as follows: If \(f_\theta(x) = y\), then it makes no change to the parameters. Otherwise, \(\theta \leftarrow \theta + yx\).

- Note that bound on the number of errors does not have an explicit dependence on the number of examples \(N\) in the sequence, or on the dimension \(d\) of the inputs.
Perceptron convergence theorem:
If there exists an exact solution (i.e., if the training data set is **linearly separable**), then the perceptron learning algorithm is guaranteed to find an exact solution in **finite number of steps**.

The number of steps required to achieve convergence could still be substantial.

For linearly separable data set, there may be many solutions.

For linearly nonseparable data set, the perceptron algorithm will never converge.
Illustration of Convergence of Perceptron
Learning Boolean AND

- Learning a Boolean function is a **two-class classification** problem.
- **AND** function with 2 inputs and 1 output:

\[
\begin{pmatrix}
  x_1 & x_2 & y \\
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
\end{pmatrix}
\]

- Perceptron for AND and its geometric interpretation:
Learning Boolean XOR

- A simple perceptron can only learn *linearly separable* Boolean functions such as AND and OR but not linearly nonseparable functions such as XOR.
- XOR function with 2 inputs and 1 output:

  \[
  \begin{pmatrix}
  x_1 & x_2 & y \\
  0 & 0 & 0 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0
  \end{pmatrix}
  \]
Learning Boolean XOR (2)

There do not exist $w_0, w_1, w_2$ that satisfy the following inequalities:

$$
\begin{align*}
    w_0 & \leq 0 \\
    w_2 + w_0 & > 0 \\
    w_1 + w_0 & > 0 \\
    w_1 + w_2 + w_0 & \leq 0
\end{align*}
$$

The VC dimension of a line in 2-D is 3. With 2 binary inputs there are 4 cases, so there exist problems with 2 inputs that are not solvable using a line.
A multilayer perceptron (MLP) has a hidden layer between the input and output layers.

MLP can implement nonlinear discriminants (for classification) and nonlinear regression functions (for regression).

We call this a two-layer network because the input layer performs no computation.

Hong Chang (ICT, CAS)
Illustration of the capability of a MLP

- Four functions: (a) $f(x) = x^2$; (b) $f(x) = \sin(x)$; (c) $f(x) = \|x\|$; (d) step function. $N = 50$ uniformly sampled.
- 3 hidden units with $\tanh$ activation functions, linear output units.
Forward Propagation

- **Input-to-hidden:**

  \[ z_h = \text{sigmoid}(w_h^T x) = \frac{1}{1 + \exp[-(\sum_{j=1}^{d} w_{hj} x_j + w_{h0})]} \]

  - The hidden units must implement a nonlinear function (e.g., sigmoid or hyperbolic tangent) or else it is equivalent to a simple perceptron.
  - Sigmoid can be seen as a continuous, differentiable version of thresholding.

- **Hidden-to-output:**

  \[ f_k = v_k^T z = \sum_{h=1}^{H} v_{kh} z_h + v_{k0} \]

  - **Regression:** linear output units.
  - **Classification:** a sigmoid unit (for \( K = 2 \)) or \( K \) output units with softmax (for \( K > 2 \)).
The hidden units make a **nonlinear transformation** from the \( d \)-dimensional input space to the \( H \)-dimensional space spanned by the hidden units.

In the new \( H \)-dimensional space, the output layer implements a **linear function**.

**Multiple hidden layers** may be used for implementing more complex functions of the inputs, but learning the network weights in such deep networks will be more complicated.
MLP for XOR

- Any Boolean function can be represented as a disjunction of conjunctions, e.g.,

\[ x_1 \text{ XOR } x_2 = (x_1 \text{ AND } \neg x_2) \text{ OR } (\neg x_1 \text{ AND } x_2) \]

which can be implemented by an MLP with one hidden layer.
MLP as a Universal Approximator

- The result for arbitrary Boolean functions can be extended to the continuous case.

- **Universal approximation:** An MLP with one hidden layer can approximate any nonlinear function of the input given sufficiently many hidden units.
Extension of the perceptron learning algorithm to multiple layers by error backpropagation from the outputs back to the inputs:

- Learning of hidden-to-output weights: like perceptron learning by treating the hidden units as inputs.
- Learning of the input-to-hidden weights: applying the chain rule to calculate the gradient:

\[
\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial f_k} \frac{\partial f_k}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}
\]
MLP Learning for Nonlinear Regression

- Assuming a single output:
  \[ f^{(i)} = f(x^{(i)}) = \sum_{h=1}^{H} v_h z_h^{(i)} + v_0 \]

  where \( z_h^{(i)} = \text{sigmoid}(w_h^T x^{(i)}) \).

- Error function over entire sample:
  \[ E(W, v | \mathcal{X}) = \frac{1}{2} \sum_i (y^{(i)} - f^{(i)})^2 \]

- Update rule for second-layer weights:
  \[ \Delta v_h = \eta \sum_i (y^{(i)} - f^{(i)}) z_h^{(i)} \]
MLP Learning for Nonlinear Regression (2)

- **Update rule for first-layer weights:**

\[
\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \sum_i \frac{\partial E}{\partial f^{(i)}} \frac{\partial f^{(i)}}{\partial z_h^{(i)}} \frac{\partial z_h^{(i)}}{\partial w_{hj}} = \eta \sum_i (y^{(i)} - f^{(i)}) v_h z_h^{(i)} (1 - z_h^{(i)}) x_j^{(i)}
\]

- \((y^{(i)} - f^{(i)}) v_h\) acts like the error term for hidden unit \(h\), which is backpropagated from the output to the hidden unit with the weight \(v_h\) reflecting the responsibility of the hidden unit.

- Either **batch learning** or **online learning** may be carried out.
Example

- Evolution of regression function and error over time
MLP Learning for Nonlinear Multiple-Output Regression

- Outputs:
  \[ f_k^{(i)} = \sum_{h=1}^{H} v_{kh}z_h^{(i)} + v_{k0} \]

- Error function:
  \[ E(W, V|\mathcal{X}) = \frac{1}{2} \sum_i \sum_k (y_k^{(i)} - f_k^{(i)})^2 \]

- Update rule for second-layer weights:
  \[ \Delta v_{kh} = \eta \sum_i (y_k^{(i)} - f_k^{(i)}) z_h^{(i)} \]

- Update rule for first-layer weights:
  \[ \Delta w_{hj} = \eta \sum_i \left[ \sum_k (y_k^{(i)} - f_k^{(i)}) v_{kh} \right] z_h^{(i)} (1 - z_h^{(i)}) x_j^{(i)} \]
Algorithm

Initialize all \( v_{kh} \) and \( w_{hj} \) to \( \text{rand}(-0.01, 0.01) \)
Repeat

For all \((x^t, y^t) \in \mathcal{X}\) in random order
  For \( h = 1, \ldots, H \)
    \( z_h \leftarrow \text{sigmoid}(w^T_h x^t) \)
  For \( k = 1, \ldots, K \)
    \( f_k = v^T_k z \)
  For \( k = 1, \ldots, K \)
    \( \Delta v_k = \eta(y^t_k - f^t_k)z \)
  For \( h = 1, \ldots, H \)
    \( \Delta w_h = \eta(\sum_k (y^t_k - f^t_k) v_{kh})z_h(1 - z_h)x^t \)
  For \( k = 1, \ldots, K \)
    \( v_k \leftarrow v_k + \Delta v_k \)
  For \( h = 1, \ldots, H \)
    \( w_h \leftarrow w_h + \Delta w_h \)

Until convergence
MLP Learning for Nonlinear Multi-Class Discrimination

- **Outputs:**

\[ f_k^{(i)} = f_k(x^{(i)}) = \frac{\exp(o_k^{(i)})}{\sum_l \exp(o_l^{(i)})} \]

which approximate the posterior probabilities \( P(C_k|x^{(i)}) \), where

\[ o_k^{(i)} = \sum_{h=1}^{H} v_{kh}z_h^{(i)} + v_{k0}. \]

- **Error function:**

\[ E(W, V|X) = -\sum_i \sum_k y_k^{(i)} \log f_k^{(i)} \]

- **Update rules:**

\[ \Delta v_{kh} = \eta \sum_i (y_k^{(i)} - f_k^{(i)}) z_h^{(i)} \]

\[ \Delta w_{hj} = \eta \sum_i \left[ \sum_k (y_k^{(i)} - f_k^{(i)}) v_{kh} \right] z_h^{(i)} (1 - z_h^{(i)}) x_j^{(i)} \]