Multi-Layer Perceptron

Mohammad Emtiyaz Khan
EPFL

Dec. 3, 2015

©Mohammad Emtiyaz Khan 2015
Multi-Layer Perceptron (MLP)

This is also known as feed-forward neural network and can be represented graphically as follows:

\[ x_n \rightarrow a_n^{(1)} \rightarrow z_n^{(1)} \rightarrow a_n^{(2)} \rightarrow z_n^{(2)} \rightarrow \ldots \rightarrow z_n^{(K-1)} \rightarrow y_n \]

where \( \{y_n, x_n\} \) is the \( n \)'th input-output pair, \( z_n^{(k)} \) is the \( k \)'th hidden vector, \( a_n^{(k)} \) is the corresponding activation. There are a total of \( K \) layers.

For the \( k \)'th layer, we obtain the \( m \)'th activation \( a_{mn}^{(k)} \) and the corresponding hidden variable \( z_{mn}^{(k)} \), as shown below:

\[
\begin{align*}
    a_{mn}^{(k)} &= \left( \beta_m^{(k)} \right)^T z_n^{(k-1)}, \\
    z_{mn}^{(k)} &= h \left( a_{mn}^{(k)} \right)
\end{align*}
\]

where \( z_n^{(k-1)} \) is the hidden vector for the previous layer. For the first layer, we set \( z_n^{(0)} = x_n \). For the last layer, we use a link function to map \( z_n^{(K-1)} \) to the output \( y_n \).

Note that a 1-Layer MLP is simply a generalization of linear/logistic regression.
Defining $\mathbf{B}^{(k)}$ as a matrix with rows $(\beta_m^{(k)})^T$, we can express the computation of activation and hidden vectors as follows:

$$
a_n^{(k)} = \mathbf{B}^{(k)} z_n^{(k-1)}, \quad z_n^{(k)} = h \left( a_n^{(k)} \right)
$$

In a more compact notation, we can express the input-output relationship as follows:

$$
\hat{y}_n = g((\beta^{(K-1)})^T \ast h(\mathbf{B}^{(K-2)} \ast h(\ast \ast \ast h(\mathbf{B}^{(1)} \ast \mathbf{x}_n))))
$$

where $g$ is an appropriate link function to match the output.

An illustration below shows reconstruction of the function $|x|$ at $N = 50$ data points sampled at the blue dots. The trained network has 2 layers and 3-hidden variables with $tanh()$ activation function.
Optimization and Back-propagation

We can learn parameters $B$ using stochastic gradient-descent.

Gradient computation can be complicated due to the deep structure of the network. We can use back-propagation to simplify the computation. The key-idea is to express the derivatives in terms of activations $a^{(k)}_n$ and hidden variables $z^{(k)}_n$ using the chain rule. Below is the outline of the algorithm:

$$ f = \sum_{n=1}^{N} \left( y_n - f(x_n) \right)^2 $$

Gradient computation:

$$ \frac{\partial f}{\partial B} = \sum_{n=1}^{N} \left[ x_n \right] $$

The backpropagation technique to compute the gradient term $\nabla_w E_i$ for a $L$-layer network can be summarized as follows:

$$ \frac{\partial f}{\partial B} = \sum_{n=1}^{N} \left[ x_n \right] $$

Forward Pass

Backward Pass

Use the chain-rule

$$ B_{i+1} = B_i - \delta_k \frac{\partial f}{\partial B} \bigg|_{B=B_i} $$

Figure 3.3: Forward pass (left) and backward pass (right) of error backpropagation algorithm for a two-layer MLP. Notice the symmetry between the two passes, and how information $g'(a^{(1)}_q)$ computed during the forward pass and stored locally at each node is recycled during the backward pass.
Step 1: Compute $\mathbf{a}_n^{(k)}$ and $\mathbf{z}_n^{(k)}$ using forward propagation.

Step 2: Compute $\delta_n^{(k)} := \partial \mathcal{L} / \partial \mathbf{a}_n^{(k)}$ using backward propagation:

$$\delta_n^{(k-1)} = \text{diag} \left[ \mathbf{h}'(\mathbf{a}_n^{(k)}) \right] \left( \mathbf{B}^{(k)} \right)^T \delta_n^{(k)}$$

Step 3: Compute $\partial \mathcal{L} / \partial \mathbf{B}^{(k)}$ using the above derivatives.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}^{(k)}} = \sum_n \delta_n^{(k)} \left( \mathbf{z}_n^{(k)} \right)^T$$

**Tricks**

Obtaining a good generalization error with neural networks and avoiding overfitting requires a lot of hacks and tricks. A good summary of these are given in Bottou’s paper “Stochastic gradient tricks”. In addition, initialization seems to play a huge role in improving the performance. See the following paper “On the importance of initialization and momentum in deep learning” by Ilya Sutskever et. al.