Perceptrons

Steven J Zeil
Old Dominion Univ.
Fall 2010

Neural Networks

- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: $10^{10}$
- Large connectivity: $10^5$
- Parallel processing
- Robust

Computing via NN

- Not so much an attempt to imitate the brain as inspired by it
- A model for massive parallel processing
- Simplest building block: the perceptron
Introduction: Neural Networks

The Perceptron

- Using Perceptrons
- Training

Multilayer Perceptrons

- Structure

Training MLPs

- Backpropagation
- Improving Convergence
- OverTraining
- Tuning Network Size

Applying MLPs

- Structuring Networks
- Dimensionality Reduction
- Time Delay Neural Networks
- Recurrent Networks

Introduction: Neural Networks

The Perceptron

- Using Perceptrons
- Training

Multilayer Perceptrons

- Structure

Training MLPs

- Backpropagation
- Improving Convergence
- OverTraining
- Tuning Network Size

Applying MLPs

- Structuring Networks
- Dimensionality Reduction
- Time Delay Neural Networks
- Recurrent Networks

Basic Uses

- Linear regression
- Linear discriminant between 2 classes
  - Use multiple perceptrons for \( K > 2 \) classes

Perceptron Output Functions

- Many perceptrons have a “post-processing” function at the output node.
- A common choice is the threshold:
  \[
  y = \begin{cases} 
  1 & \text{if } \vec{w}^T \vec{x} > 0 \\
  0 & \text{ow} 
  \end{cases}
  \]

Useful for classification.
Sigmoid Output Functions

- Useful when we need differentiation or the ability to estimate posterior probs.

\[ y = \text{sigmoid}(o) = \frac{1}{1 + \exp[-\vec{w}^T \vec{x}]} \]

K Classes

- \( o_i = \vec{w}_i^T \vec{x} \)
- Use softmax:
  \[ y_i = \frac{\exp o_i}{\sum_k \exp o_k} \]
- Choose \( C_i \) if \( y_i = \max_k y_k \)

Training

- Allows online (incremental) training rather than the usual batch
  - No need to store whole sample
  - Adjusts to slow changes in the problem domain
- Incremental form of gradient-descent: update in direction of gradient after each training instance
- LMS update:
  \[ \Delta w_{ij}^t = \eta \left( r_i^t - y_i^t \right) x_j^t \]
  - \( \eta \) is the learning factor - size controls rate of convergence and stability

Update Rule: Regression

- Error function is
  \[ E^t(\vec{w} | \vec{x}^t, r^t) = \frac{1}{2} (r^t - \vec{w}^T \vec{x}^t)^2 \]
  with gradient components
  \[ \frac{\partial E^t}{\partial w_i^t} = -(r^t - \vec{w}^T \vec{x}^t)x_i^t = -(r^t - y^t)x_i^t \]
- Therefore to move in the direction of the gradient
  \[ \Delta w_{ij}^t = \eta \left( r_i^t - y_i^t \right) x_j^t \]
### Update Rule: Classification

\[ \Delta w_{ij}^t = \eta (r^t_i - y^t_i) x^t_j \]

- For \( K = 2 \),
  \[ y^t_i = \text{sigmoid}(\vec{w}^T \vec{x}) \]
  leads to the same update function as for regression
- For \( K > 2 \), softmax leads to the same update as well.

### Example: Learning Boolean Functions

- Example: spreadsheet
  - demonstrates that perceptrons can learn linearly separable functions (AND, OR, NAND, \ldots)
  - but cannot learn XOR
    - Minsky & Papert, 1969
    - Nearly halted all work on neural networks until 1982

### Multilayer Perceptrons

- Adds one or more hidden layers
  \[ y_i = \vec{v}^T \vec{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0} \]
  \[ z^h = \text{sigmoid}(\vec{w}^T \vec{x}) \]
  \[ = \frac{1}{1 + \exp \left( - \left( \sum_{j=1}^{d} w_{hj} x_j + w_{h0} \right) \right)} \]
  (Rumelhart et al. 1986)
Learning XOR

MLP as a Universal Approximator

Any function with continuous inputs and outputs can be approximated by an MLP

- Given two hidden layers, can use one to divide input domain and the other to compute a piecewise linear regression function
- Hidden layers may need to be arbitrarily wide

Training MLPs: Backpropagation

\[ y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_i h z_h + v_i 0 \]

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

\[ = \frac{1}{1 + \exp\left(-\left(\sum_{j=1}^d w_{hj} x_j + w_{h0}\right)\right)} \]

Given the \( z \) values, we could train the \( \mathbf{v} \) as we do a single-layer perceptron.

\[ \Delta v_h = \eta \sum_t (r_t - y^t) z_h^t \]
Backpropagation (cont.)

\[
\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}}
\]
\[
= \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}
\]
\[
= -\eta \sum_t (r_t - y^t) \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}
\]
\[
= -\eta \sum_t (r_t - y^t) v_h \frac{\partial z_h}{\partial w_{hj}}
\]
\[
= -\eta \sum_t (r_t - y^t) v_h z_h^t (1 - z_h^t) x_j^t
\]

Applying Backpropagation

- Batch learning: make multiple passes over entire sample
  - Update \( \vec{v} \) and \( \vec{w} \) after each entire pass
  - Each pass is called an epoch
- Online learning: one pass, smaller \( \eta \)

Example of Batch Learning
Multiple hidden Levels

- Multiple hidden levels are possible
- Backpropagation generalizes to any number of levels.

Improving Convergence

- Momentum: Attempts to damp out oscillations by averaging in the “trend” of prior updates
  \[ \Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1} \]
  \[ 0.5 \leq \alpha < 1.0 \]
- Adaptive Learning rate: Keep \( \eta \) large when learning is going on, decreasing it later
  \[ \Delta \eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases} \]

Note that increase is arithmetic, but decrease is geometric.

OverTraining

MLPs are subject to overtaining

- partly due to large number of parameters
- but also is a function of training time
  - \( w_i \) start near zero - in effect the parameters are ignored
  - Early training steps move the more important attributes’ weights away from zero
  - As training continues, we start fitting to noise by moving the weights of less important attributes away from zero
    - In effect, adding more parameters to the model over time.
Tuning Network Size

- Destructive: remove units or connections that are unnecessary.
- Constructive: add units or connections to add performance

Destructive Tuning

Weight decay:
Give each weight a tendency to decay towards zero unless it is refreshed by additional training examples:

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

Equivalent to gradient descent training with error function:

\[ E' = E + \frac{\lambda}{2} \sum_i w_i^2 \]

penalizing solutions with large numbers of non-zero weights.

Constructive Tuning

- Train initial small network.
- If error is high, add a hidden unit and retrain
  - Dynamic Node Creation
  - Cascade Correlation
Dynamic Node Creation

- Start with a hidden layer with one hidden unit.
- New nodes added to that layer:
  - Never increases the number of layers.
- Weights of new unit are started randomly.
- Already-trained weights start from their trained values.

Cascade Correlation

- Each new node becomes the only node in a new layer.
  - Connected to all of the existing hidden units and to all inputs.
- Weights of new unit are started randomly.
- Already-trained weights are frozen at their trained values.

Applying MLPs

1. Introduction: Neural Networks
2. The Perceptron
   - Using Perceptrons
   - Training
3. Multilayer Perceptrons
   - Structure
4. Training MLPs
   - Backpropagation
   - Improving Convergence
   - OverTraining
   - Tuning Network Size
5. Applying MLPs
   - Structuring Networks
   - Dimensionality Reduction
   - Time Delay Neural Networks
   - Recurrent Networks

Structuring Networks

When we have knowledge of input structure (e.g., vision):
- Pixels are arranged in rectangular arrays.
- Locally correlated structures (e.g., edges) are important.
  - A hierarchical cone.
Weight Sharing

Take advantage of uniformity over a spatial dimension.

Introduction: Neural Networks

The Perceptron

Multilayer Perceptrons

Training MLPs

Applying MLPs

Hints

Prior knowledge of equivalent cases, e.g., invariance to common graphic transforms:

- Use to auto-expand training set ("virtual examples")
- Reduce equivalent cases to a canonical form during pre-processing
- Incorporate into network structure (e.g., weight sharing)
- Augment the error function to penalize violations of the equivalence

\[ E' = E + \lambda_h E_h \]

where

\[ E_h = \begin{cases} 
( g(x|\theta) - g(x'|\theta) )^2 & \text{if } x' \simeq x \\
0 & \text{otherwise}
\end{cases} \]

Dimensionality Reduction

- Looking at weights of trained MLP can give hints as to which input attributes are significant
- In any MLP, if the number of units in the first hidden layer is less than the number of inputs, we are doing dimensionality reduction.
- In an auto-associator, we train an MLP to generate its own inputs.
  - Using an intermediate hidden layer of fewer units than the number of inputs.
- In essence, performs principal components analysis
  - Weight vectors span the same space and the principle eigenvectors

Linear Auto-Associator

Looking at weights of trained MLP can give hints as to which input attributes are significant

In any MLP, if the number of units in the first hidden layer is less than the number of inputs, we are doing dimensionality reduction.

In an auto-associator, we train an MLP to generate its own inputs.

Using an intermediate hidden layer of fewer units than the number of inputs.

In essence, performs principal components analysis

- Weight vectors span the same space and the principle eigenvectors
Non-Linear Auto-Associator

- Nonlinear dimensionality reduction

Decoder

Encoder

$x_0 = +1$  $x_1$  $x_d$

$y_1$  $y_d$

Nonlinear

Time Delay Neural Networks

For learning time sequences

Recurrent Networks

In effect, adds limited memory to MLPs
Train by unfolding (similar to loop unrolling)