3. Neural Networks

Table of Content
1. Introduction
2. Definitions
3. Neural Network Types
4. Learning
5. Advantages & Disadvantages
6. Self-Organizing Map (SOM)

§3.1 Introduction

A neuron

A network of neurons
§3.1 Introduction

What are neural networks (NNT)?

What problem do they solve?

The system: Learn to classify objects
§3.1 Introduction

- NNT are used in:
  - Applications in function fitting
  - Pattern recognition
  - Clustering
  - Time series analysis …

- Application areas:
  - Aerospace
  - Automotive
  - Banking
  - Defence
  - Electronics
  - Entertainment
  - Financial
  - Industrial
  - Insurance
  - Manufacturing
  - Medical
  - Oil and gas
  - Speech & writing
  - Telecommunications
  - Transportation
  - Robotics
  - Securities
  - ...
§3 Neural Networks

3.2 Definitions

- Transfer functions
  - Step function: \( y = f(x) = \begin{cases} 1, & w \cdot x + b > \theta \\ 0, & w \cdot x + b < \theta \end{cases} \)
  - Linear combination: \( y = f(x) = b + \sum_{i=1}^{n} w_i \cdot x_i \)
  - Sigmoid: \( y = f(x) = \frac{1}{1 + e^{-w_0 \cdot x - w_1 \cdot b}} \)

- Without bias

- With bias
§3.2 Definitions

- Multi-Layer Neural Network

![Diagram of Multi-Layer Neural Network]

§3.3 Neural Network Types

Types of NNTs

- Single-layer Feed-Forward
- Multi-layer
- Recurrent
  - Parallel
  - Series-Parallel

§3.3 Neural Network Types

- Single-layer Feed-Forward
  - One layer
  - Information (signals) is only fed forward in the network
  - No back-propagation
§3 Neural Network Types

3.3 Neural Network Types

\* Single-layer Feed-Forward

\[ w = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \]
\[ b = 0 \]

\* Perceptron classification

\* MLP (Multi Layer Perceptron)
§3.3 Neural Network Types

- Layer-Recurrent Network (LRN)

§3.4 Learning

- Learning types
  - Supervised
  - Unsupervised
  - Hybrid

§3.4 Learning

- Algorithms of supervised learning
  - Neural network (Multilayer perceptron)
  - Support Vector Machine
  - Linear regression
  - Logistic regression
  - Naïve Bayes
  - Linear discriminant analysis
  - Decision trees
  - K-nearest neighbour algorithm
§3.4 Learning

- Steps of supervised learning:
  - Determine the type of training examples
  - Gather a training set
  - Determine the input feature representation of the learned function
  - Determine the structure of the learned function and corresponding learning algorithm
  - Train the network
  - Evaluate the accuracy of the learned function

- Learning algorithms
  - Learning vector quantization (LVQ)
  - Least mean square error (LMS)
  - Perceptron learning rule
  - Back propagation network (BPN)

- Perceptron learning rule
  - Set of training examples \( \{(x_1,y_1), (x_n, y_n)\} \)
  - A learning algorithm seeks a function \( g: X \rightarrow Y \)
    - \( X \) is the input space
    - \( Y \) is the output space

\[
\begin{array}{c||c}
X: \text{Input Vector} & \\
\hline
\text{XXX} & \text{white} \\
\text{YYY} & \text{black} \\
\hline
\end{array}
\]

\( Y = \text{Target (tag, output)} \)
§3.4 Learning

- Perceptron learning rule
  - $x$: input vector
  - $y = f(x)$: output from the perceptron
  - $b$: bias term
  - $D = \{(x_1, d_1), \ldots, (x_n, d_n)\}$: training set
    - $x_j$: $n$-dimensional input vector
    - $d_j$: the desired output of the perceptron
  - $w$: weight vector
  - $w(t)$: weight at time $t$
  - $\alpha$: learning rate

- Initialize all weights $w_i(0)$
  - E.g., to 0 or a random value
- Give a threshold $\gamma$
- For each sample $j$ in the training set $D$
  - Calculate the actual output: $y_j(t) = f(w(t) \cdot x_j)$
  - Adapt the weights: $w_i(t+1) = w_i(t) + \alpha (d_j - y_j(t)) \cdot x_{ij}$
- Repeat previous step until $d_j - y_j(t) < \gamma$
  or a predetermined number of iterations has been performed

- The training set $D$ is said to be linearly separable, if
  $\exists \gamma, \exists w \forall 1 < j < n: (w \cdot x_j + b) > \gamma$
- If the data set is linearly separable, then the algorithm converges after a finite number of steps.
- Otherwise, it will not converge.
- Therefore, normally a fixed number of steps is performed.
§3.4 Learning

- Learning and training process in NNTs

- Problems of supervised learning:
  - Trade-off between bias & variance
  - Function complexity and amount of training data
  - Dimensionality of the input space
  - Noise in the output values

- Other factors:
  - Heterogeneity of the data
  - Redundancy in the data
  - Presence of interactions and non-linearities

- Generalizations:
  - Use semi-supervised learning
  - Use active learning
  - Structured prediction
  - Learn to rank
§3.4 Learning

Training types of NNTs

- Back propagation network (BPN)
- Radial basis function network (RBF)
- Levenberg-Marquardt network (LMN)
- Hopfield network

§3.4 Learning

- Back propagation network (BPN)
  - Collect data
  - Create the network
  - Configure the network
  - Initialize the weights and biases
  - Train the network
  - Validate the network (post-training analysis)
  - Use the network

§3.4 Learning

- Back propagation network training
  - Propagation phase
    - Forward propagation of training instance of input
    - Backward propagation of output activations
    - Generates deltas between output and hidden neurons
  - Weight update phase
    - For each weight
      - Get gradient: multiply output delta and input activation
      - Subtract a ratio of the gradient from the weight
    - Repeat both phases until the performance of the network is good enough
§3.4 Learning

- Back propagation network training
  - Online learning
    - Propagation and weight update follow each other
      - Requires more updates
  - Batch learning
    - Many propagations are performed before a weight update occurs
      - Requires more memory

- Gradient computation:
  - Momentum
  - Levenberg-Marquardt
  - Conjugate Gradient
  - Variable Learning Rate
  - Steepest Descent
  - BFGS quasi-Newton
  - Powell-Beale conjugate gradient
  - Fletcher-Powell conjugate gradient
  - Polak-Ribière conjugate gradient
  - Gradient descent with adaptive learning rule
  - Gradient descent with momentum
  - One step secant
  - Resilient gradient descent

- Example
§3.4 Learning

- Limitations of back-propagation learning
  - The convergence obtained is very slow
  - The convergence is not guaranteed
  - The result may generally converge to any local minimum on the error surface, since stochastic gradient descent exists on a non-linear surface.

§3.5 Advantages and Disadvantages

- Advantages of neural networks
  - Non-linearity
  - Relation between input with output
  - Error-tolerance
  - Have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft, to detecting credit card fraud

- Disadvantages
  - Initialization
    - Affects the convergence rate
  - Interpretation
    - No understanding how it works (black box)
  - Require a large diversity of training for real-world operation
  - Could be over-trained

  - Better use hybrid models combining neural networks and symbolic approaches
§3.6 Self-Organizing Maps

- Techniques for reducing the dimensions of data
  - Self Organizing Maps (SOM)
    - Also called Kohonen networks
      - [http://www.scholarpedia.org/article/Kohonen_network]
  - Principle Component Analysis (PCA)
  - Multi-Dimensional Scaling (MDS)
  - ...

A simple two-dimensional Kohonen network

Kohonen network of 4 X 4 nodes
Which is the output layer

colors are classified
§3.6 Self-Organizing Maps

- Define an ordered mapping
- Projection from a set of data items onto a regular, usually two-dimensional grid
- A model $m_i$ is associated with each grid node
- These models are computed by the SOM algorithm
- A data item will be mapped to the node whose model is most similar to the data item
  - E.g., in terms of a distance metric
- Normally, the model is a weighted local average of the given data items in the data space
  - The model of nearby nodes are more similar than those of nodes that are further away

§3.6 Self-Organizing Maps

- Input: $x(t) = [x_1(t), ..., x_n(t)]$
  - $t$ is the index of the data item
- Model: $m_i(t) = [m_{i1}(t), ..., m_{in}(t)]$
  - $t$ denotes the iteration of updating $m$
- Update: $m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t) \cdot (x(t) - m_i(t))$
  - $\alpha(t)$: size of the correction, decreases with $t$
  - $i$: model under processing
  - $c$: index of the model having the smallest distance from $x(t)$
  - $h_{ci}(t)$: smoothing kernel (neighborhood function)
§3.6 Self-Organizing Maps

- $h_c(t)$: smoothing kernel (neighborhood function)
  - $k = c \rightarrow h_c(t) = 1$
  - $h_c(t)$ decreases with the increase of the distance between the models $m_i$ and $m_c$
  - The spatial width of the kernel should decrease over $t$

- The functions of the step index (see Kohonen)
  - Determine the convergence
  - Must be chosen very delicately

- Initialization of $m_c(t)$ is a problem (see Kohonen)

§3.6 Self-Organizing Maps

- Improved algorithm
  - Batch Map
  - An order of magnitude faster
  - For every node $j$ in the grid
    - Compute the average $s_j$ of all input items $s(t)$ that have $m_j$ as closest model
    - Compute the new models as: $m_i = \frac{1}{n_j} \sum_j m_j s_j(t)$
    - $n_j$: number of input items mapped into node $j$
    - The index $j$ runs over the neighbors of node $i$