**Tutorial on artificial neural network**

Loc Nguyen

Loc Nguyen’s Academic Network, Vietnam

Email: ng\_phloc@yahoo.com

Homepage: www.locnguyen.net

# Abstract

**Keywords:** artificial neural network (ANN).

|  |  |
| --- | --- |
|  | () |

# 1. Introduction

Artificial neural network (ANN) is the mathematical model based on biological neural network but neural network (NN) in this research always indicates artificial neural network. NN consists of a set of processing units which communicate together by sending signals to each other over a number of weighted connections (Kröse & Smagt, 1996, p. 15). Each *unit* is also called neuron, cell, node, or variable which is quantified by a real variable. Each weighted connection, which is considered a neural cord, is often quantified by a real number called *weight*. According to Kröse & Smagt, each unit is responsible for receiving input from neighbors or external sources and using this input to compute an output signal which is propagated to other units (Kröse & Smagt, 1996, p. 15). The most important thing here is that the signal propagation is done by the means of weighed connections which are imitated as biological neurotransmission with neurons and neural cords. According to Kröse & Smagt (Kröse & Smagt, 1996, pp. 15-16), there are three types of units:

* *Input units* receive data from outside the network. These units structure the *input layer*.
* *Hidden units* own input and output signals that remain within the neural network. These units structure the hidden layer. There can be one or more *hidden layers*.
* *Output units* send data out of the network. These units structure the *output layer*.

Units in NN are also considered variables. The figure (Wikipedia, Artificial neural network, 2009) below shows the simplest structure of an artificial neural network with three layers such as input layer, hidden layer, and output layer. The structure of neural network is often called the topology.

Diagram

Description automatically generated

**Figure 1.1.** Simplest topology of neural network with three layers such as input layer, hidden layer, and output layer

Note that the main reference of this research report is the book “An Introduction to Neural Networks” by Ben Kröse and Patrick van der Smagt (Kröse & Smagt, 1996).

According to Daniel Rios (Rios), there are two main topologies (structures) of neural network:

* *Feed-forward NN* is directed acyclic graphic in which flow of signal from input units to output units is one-way flow and so, there is no feedback connection.
* *Recurrent NN* is the one whose graph (topology) contains cycles and so, there are feedback connections.

It is necessary to evolve NN by modifying the weights of connections so that they become more accurate. In other words, such weights should not be fixed by experts. NN should be trained by feeding it teaching patterns and letting it change its weights. This is learning process or training process. According to Daniel Rios, there are three types of learning methods (Rios):

* *Supervised learning*: According to Daniel Rios (Rios), the network is trained by matching its input and its output patterns. These patterns are often known as classes which can be represented by binary values, integers for nominal indices, or real numbers.
* *Unsupervised learning*: The network is trained in response to clusters of patterns behind the input. According to Daniel Rios (Rios), there is no a priori set of categories into which the patterns are to be classified.
* *Reinforcement learning*: The learning algorithms receive partially information along with input from environments and then, adjust partially and progressively the weighted connections by adaptive way to such input. According to Daniel Rios (Rios), reinforcement learning is the intermediate form between supervised learning and unsupervised learning.

This introduction section focuses on supervised learning in which input and output are realistic quantities (real numbers). For NN, the essence of supervised learning is to improve weighted connections by matching input and output. Given unit *i*, let *xi* and *yi* denote *input* and *output* of unit *i*, which are real numbers. In NN literature, a unit will be activated if its output is determined and so the output *yi* is also called *activation* of unit *i*. If a unit is input unit (in input layer) then its input contributes to input of NN. If a unit is output unit (in output layer) then its output contributes to output of NN. Each connection between two successive units such as unit *i* and unit *j* is defined by the weight *wij* determining effect of unit *i* on unit *j*. In the normal topology, an output unit is the composition of other hidden units which in turn are the compositions of others input units. The composition (aggregation) of a unit is represented as a weighted sum which will be evaluated to determine the output of this unit. The process of computing the output of a unit includes two following steps (Han & Kamber, 2006, p. 331):

* An adder called *summing function* sums up all the inputs multiplied by their respective weights. It is essential to compute the weighted sum. This activity is referred to as linear combination.
* An *activation function* controls the amplitude of the output of the neuron. This activity aims to determine the output. Note that the output of the previous units is the input of current unit.

Figure 1.2 (Han & Kamber, 2006, p. 331) describes the process of computing the output.

**Figure 1.2.** Process of computing output of a unit

For example, as seen in figure 1.2, given a concerned unit *k*, suppose there are previous units whose outputs *yj* (s) are considered as inputs of unit *k*. According to the process of computing the output of a unit, we have following equation (Han & Kamber, 2006, p. 331), (Kröse & Smagt, 1996, pp. 16-17) for computing the output value of a unit.

|  |  |
| --- | --- |
|  | () |

Or shortly:

The equation above for output processing is called *propagation rule*. Note, *wjk* is the weight of the connection from unit *j* to unit *k* and *θj* is the bias of unit *j* while *fj*(.) is the activation function acting on unit *j*. If all units use the same form of activation function, we can denote *f*(.) = *fj*(.).

In a distributed environment, NN can be evolved asynchronously when the computing processes on different units can be computed by distributed way. Given time point *t*, the propagation rule at time point *t* + 1 is rewritten as follows:

|  |  |
| --- | --- |
|  | () |

As a convention, input units in input layer are indexed by *i* (for instance, *xi* and *yi*), hidden units in hidden layer are indexed by *h* (for instance, *xh* and *yh*), and output units in output layer are indexed by *o* (for instance, *xo* and *yo*). Therefore, indices *j*, *k*, *l*, etc. indicate normal units having both input and output. For input units, we assume that *xi* = *yi* and *θi*=0. A NN is valid if it has two or more layers and so there is a convention that a *n*-layer has *n*+1 actual layers, which means that the input layer is not counted for this convention. This is reasonable because the output processing equation is not applied to input units. The simplest NN is single layer NN owning one input layer and one output layer.

Output values of units are arbitrary, but they should range from 0 to 1 (sometimes –1 to 1 range). In general, every unit has following aspects:

* Each unit *k* has input *xk* and output *yk*. Moreover, let *vk* be the actual value of unit *k* from environment or database. The actual value *vk* can be equal to or different from the output *yk* with note that *yk* is derived from the propagation rule.
* A set of inputs connects to it. Each connection is quantified by a weight *wjk*.
* A bias value *θk* will be added to the weighted sum.
* The weighted sum is computed by summing up all the inputs modified by their respective weights. Summing function or adder is responsible for this summing task.
* Its output *yk* is the outcome of activation function *f*(.) on weighted sum. Activation function is crucial factor in NN. The combination of summing function and activation function constitutes the propagation rule, but the propagation rule can be more complicated with some enhancements.

The activation function *f*(*.*) is the squashing function which “squashes” a large weighted sum into possible smaller values ranging from 0 to 1 (sometimes –1 to 1 range). There are three types of activation function (Rios):

* *Threshold function* takes on value 0 if weighted sum is less than 0 and otherwise. The formula of threshold function is:
* *Piecewise-linear function* takes on values according to amplification factor in a certain region of linear operation. The formula of piecewise-linear function is:
* *Sigmoid function* or logistic function takes on values in range [0, 1] or [–1, 1]. The formula of sigmoid function is:

|  |  |
| --- | --- |
|  | () |

Where *e*(.) or exp(.) denotes exponent function. Logistic function is the most popular activation function.

Recall that the essence of learning NN (training NN) is to improve weighted connections by matching input and output. Given a weight *wjk* from unit *j* to unit *k*, a new version of *wjk* after learning process at time point *t* is updated by weight deviation Δ*wjk* as follows:

Or shortly:

|  |  |
| --- | --- |
|  | () |

The equation above is called *weight update rule* and hence, weight update rule focuses on how to calculate the weight deviation Δ*wjk*. Learning NN algorithms also improve biases beside improving weights. Given bias *θk* of unit *k*, a new version of *θk* after learning process at time point *t* is updated by bias deviation Δ*θk* as follows:

Or shortly:

|  |  |
| --- | --- |
|  | () |

The equation above is called *bias update rule* and hence, bias update rule focuses on how to calculate the bias deviation Δ*θk*. In general, a normal learning NN algorithm needs to specify both weight update rule and bias update rule because both of them determine the propagation rule.

The most popular learning NN algorithm is back-propagation algorithm, but we should skim some simpler learning algorithms first. Two common simpler learning algorithms are Perceptron and Adaline. Both of them are based on *Hebbian rule* and *delta rule*. Hebbian rule indicates that Δ*wjk* is proportional to product of output of unit *j* and output of unit *k* as follows (Kröse & Smagt, 1996, p. 18):

|  |  |
| --- | --- |
|  | () |

Where the positive constant γ is called learning rate specifying the power of proportional, which relates to speed of learning process. Both *yj* and *yk* are results of the propagation rule. Let *vk* be the actual value of unit *k* from environment or database, delta rule indicates that Δ*wjk* is proportional to product of output value of unit *j* and output deviation of unit *k* as follows (Kröse & Smagt, 1996, p. 18):

|  |  |
| --- | --- |
|  | () |

Obviously, Hebbian rule and delta rule are weight update rules.

Recall that the most popular NN algorithm is back-propagation algorithm whereas two simpler learning algorithms are Perceptron and Adaline. Perceptron algorithm is used to train a simple single layer NN called Perceptron. For instance, Perceptron has some input units and one output unit. Without loss of generality, Perceptron has two input units whose (input) values are denoted *x*1 and *x*2 and one output unit whose (output) value is denoted *y* with note that *y* is binary {–1, 1} and bias of the output unit is *θ*, as seen in figure 1.3 (Kröse & Smagt, 1996, p. 23).

**Figure 1.3.** Perceptron topology

As a convention, we can call input unit *x*1, input unit *x*2, output unit *y*, and bias *θ* although they are actual values. The propagation rule of Perceptron is (Kröse & Smagt, 1996, p. 23):

|  |  |
| --- | --- |
|  | () |

Which is, indeed, a binary classifier for supervised learning whose inputs are *x*1 and *x*2 and whose output is the binary class {–1, 1}. Classification equation from the Perceptron propagation rule is *w*1*x*1 + *w*2*x*2 + *θ* = 0. The weight update rule of Perceptron is:

Let *v* {–1, 1} be the actual value of unit *y* from environment or database, Perceptron learning algorithm calculates the weight deviation Δ*wi* as follows (Kröse & Smagt, 1996, pp. 24-25):

|  |  |
| --- | --- |
|  | () |

Therefore, the weight update rule of Perceptron is slightly similar to Hebbian rule. The bias update rule of Perceptron is:

Perceptron learning algorithm calculates the bias deviation Δ*θi* as follows (Kröse & Smagt, 1996, p. 25):

|  |  |
| --- | --- |
|  | () |

For example, with initialized values *w*1 = 1, *w*2 = 1, and *θ* = 0, given sample *x*1 = 1, *x*2 = 2, and *v* = 1, Perceptron weights and biases are updated as follows:

# References

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques* (2nd Edition ed.). (J. Gray, Ed.) San Francisco, CA, USA: Morgan Kaufmann Publishers, Elsevier.

Kröse, B., & Smagt, P. v. (1996). *An Introduction to Neural Networks* (8th Edition ed.). Amsterdam, The Netherlands: University of Amsterdam.

Rios, D. (n.d.). *Introduction to Neural Networks.* Retrieved 2009, from Neuro AI website: http://www.learnartificialneuralnetworks.com/introduction-to-neural-networks.html

Wikipedia. (2009, January 4). *Artificial neural network*. (Wikimedia Foundation) Retrieved 2009, from Wikipedia website: http://en.wikipedia.org/wiki/Artificial\_neural\_network

**Contents**

[Abstract 1](#_Toc131183221)

[1. Introduction 1](#_Toc131183222)

[References 5](#_Toc131183223)