Overview of supervised machine learning in neural network

Comp 3190 Research Project

Tan Jian Hau, Nicholas

7952446

Abstract

This work introduces the fundamentals of supervised machine learning, focusing on the structure and functionality of a single artificial neural node and an entire neural network. It explores various types of artificial neural nodes and their roles within the network. Additionally, it provides an in-depth examination of the training process for supervised learning models, highlighting key algorithms such as the perceptron rule and backpropagation. This comprehensive overview offers a foundational understanding of supervised machine learning and its underlying mechanisms.

Introduction

According to Russell & Robert, supervised learning in the computer world is a way to develop a system by tweaking the algorithm on each training step in an effort to make the output results as close to the desired results, or so-called target. The training dataset used in supervised learning consists of pairs of input patterns and targets (labeled dataset). On each training iteration step it first uses the input patterns to make a prediction and later compares the prediction with the target, then modifies the algorithm to minimize the difference between the prediction and the target, known as the error, and this is where the "learning" comes from, is like a "teacher" supervises the system to get the correct prediction output. The system studies the relationship between the inputs and targets to obtain a good mapping function [Russell & Robert, 1999].

Supervised learning is commonly used for classification and regression tasks. Classification is considered to assign a label (or category) to a given input, and regression is considered as to predict a numeric value from a given input. Unlike unsupervised learning, it aims to detect patterns using an unlabelled dataset, meaning without having a target in the training dataset [Choi et al., 2020].

In this paper, supervised machine learning in artificial neural network will be focused, meaning the artificial neural network is the learning system.

1 Structure of artificial neural network

According to Gurney, Artificial Neural Network (ANN) is inspired by the biological neural network. Quick look at the biological neural network, neurons are interconnected from one to another. Explaining in simplified form, in the biological brain, each incoming "fire strength" is evaluated by multiplying the incoming signal and its weighted value, and the neuron "fire" according to a threshold value for the total incoming "fire strength" [Gurney, 1997]. The Hebbian Theory suggests "nerves that fire together, wire together", meaning the connections between neurons strengthen when there are correlated outputs in the biological neural network on those neurons, just like in ANN, each connection in ANN is weighted such that it will give a desired output [Choi et al., 2020].

By Gurney, each artificial neuron is referred to as a node. Artificial neural networks can be multi-layered or single-layered nets. N-layered nets have a layer of input nodes, another layer of output nodes, and (N-1) layers of hidden nodes between the input and output layer. For a single-layered ANN, there is only an input and output layer. Each node takes in one or more inputs from previous layer nodes (depending on how many nodes it is connected from the previous layer) or possibly directly from the training dataset if it is a first layer node. The input value can be any real number ranging from 0 to 1, or it also can be a binary, possibly only either a 0 or 1. Each node takes in input from its previous connected nodes and produces output to its following connected nodes. For a node receiving n input signals, the input signals are denoted as x_1, x_2, ..., x_n. Each corresponding input signals have its own weight, denoted as w_1, w_2, ..., w_n, and it can be negative or positive. The subscript of the input signal and weight are for ordering purposes, where x_i and w_i are the ith input and weight. The activation α is calculated by summing the product of the input signals and the weights, showed in (2.1).

$$\alpha = X_1W_1 + X_2W_2 + \cdots + X_iW_i + \cdots + X_nW_n \tag{1.1}$$

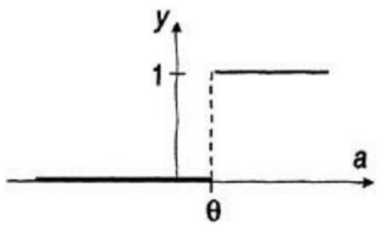
Each product of input signal and weight can either be negative or positive, and so does activation α. Then, the activation α will be passed to a function, comparing the activation with a threshold, producing a high output value (conventionally 1) if exceeds the threshold, or else 0 [Gurney, 1997].

1.1 Threshold logic unit (TLU)

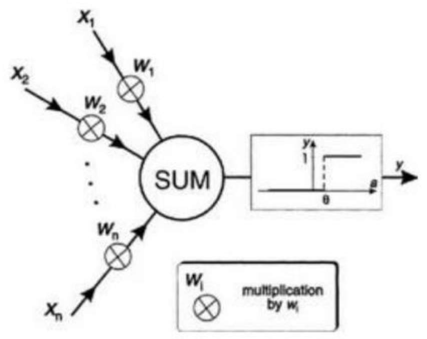
The threshold logic unit (TLU) is first proposed by McCulloch and Pitts [McCulloch & Pitts 1943]. By Gurney, in threshold logic unit (TLU), the activation α will be passed to a step function, if the activation α is greater than or equal to a threshold value θ, the node will give an output value of 1, and or else 0. The output value y calculation equation is

$$y = \begin{cases} 1\ if\ \alpha\ \geq\ \theta \\ 0\ if\ \alpha\ < \theta \end{cases} \tag{1.2}$$

Expressing y in graphical form



Expressing TLU In graphical form



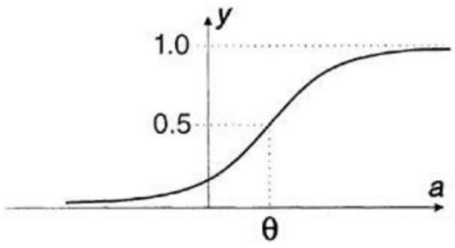
[Gurney, 1997].

1.2 Non-binary signals

Gurney also mentioned the limitation of TLU when dealing with non-binary signals. So far, the signals dealing here in TLU are binary, either 1 or 0. If the training dataset is in the form of non-binary, it is impossible to have continuously graded signals using the step function as an activation function. In order for the output y to depend smoothly on activation α, Gurney suggested “softening” the step function into a “squashing” continuous function. One such continuous function is logistic sigmoid (known as "sigmoid"). The mathematical and graphic expression is shown below, which ρ determines the shape of the function.

$$\mathrm{y} = \sigma(a) = \frac{1}{1 + e^{-(a-\theta)/\rho}} \tag{1.3}$$

Sigmoid function in graphical form



ANN uses a sigmoidal output relation known as the semilinear type, where the activation function is a linear relation with a non-zero slope [Gurney, 1997].

2 Training

According to Gurney, during the training process, the training dataset is presented repeatedly, and small changes are made to the weights and threshold value at each presentation in an effort to make the output to be or at least closer to the target. The degree of change on the weights and threshold is controlled by the learning rate, denoted as η. On the other hand, the guideline on the procedure of changing the weights and threshold is called the learning rule [Gurney, 1997]. In this chapter, the perceptron rule and the backpropagation will be focused.

2.1 Perceptron rule

2.1.1 Perceptron

Perceptron is introduced by Rosenblatt [Rosenblatt, 1961]. It is a binary classifier consisting of two layers of neurons, where it has a layer of inputs that is connected to a TLU output layer [Zhou, 2021].

2.1.2 Adjusting weight and threshold

We will focus on training TLUs. To make the calculations easier, the input, target, and weight are presented in vector form, respectively x is $\{X_1, X_{2,} \dots, X_n\}$, t is $\{t_1, t_2, \dots, t_n\}$, and w is $\{w_1, w_2, \dots, w_n\}$. Therefore, the dot product $w \cdot x \ \geq \theta \ \Rightarrow w \cdot x - \theta \geq 0$ is the condition for output "1" or else "0". Note that the negative threshold in the condition equation is referred to as bias.

To also treat the threshold the same as the weights in the parameter tuning phase, an augmented weight vector will be used such that the threshold is the w_{n+1}, meaning it is $\{w_1, w_2, \dots, w_n, \theta\}$, and also vector input x modified to $\{X_1, X_{2,} \dots, X_n, -1\}$. To avoid ambiguity, from now on, w is the augmented weight vector $\{w_1, w_2, \dots, w_n, \theta\}$ and v is $\{X_1, X_{2,} \dots, X_n, -1\}$. The training set is presented as a set of pairs $\{\ v\ ,\ t\ \}$, where the input vector v corresponds to the output target t. Therefore, the condition equation for output y is

$$w \cdot v \geq 0 \quad \Rightarrow \quad \mathrm{y} = 1 \tag{2.1}$$

$$w \cdot v < 0 \quad \Rightarrow \quad y = 0$$

The motive is to make all of the weights and the threshold in one vector w so that it can be modified easily during the parameter tuning phase by subtracting or adding a certain amount to w. Recall that the learning rate η is the degree of change in the parameter, it affects the learning speed. Suppose the output y is 1 and the corresponding target t is 0, the goal is to make dot product $w \cdot v$ negative to make output y = 0. The w needed to be subtracted by ηv , then the new weight w' will be:

$$w' = w - \eta v \tag{2.2}$$

On the other hand, if output y is 0 and target *t* is 1, the new weight w' is:

$$w' = w + \eta v \tag{2.3}$$

Since the possible output value in TLU is either 0 or 1, combining (2.2) and (2.3), a general equation for getting w' is:

$$w' = w + \eta(t - \mathrm{y})v \tag{2.4}$$

Therefore, the change $\Delta w = w' - w$ can be written as

$$\Delta w = \eta(t - \mathrm{y})v \tag{2.5}$$

In terms of the component, where *i* = 1, 2, ..., n+1, and $w_{n+1} = \theta$ and $v_{n+1} = -1$

$$\Delta w_i = \eta(t - y)v_i \tag{2.6}$$

2.1.3 Training steps

According to Gurney, (2.4), (2.5), and (2.6) define the perceptron training rule, and the training steps are as follows:

```
for each training vector pair ($v$, $t$){
    evaluate the output $y$ when $v$ is input to the TLU
    if ($y \neq t$ ){
        form a new weight vector $w'$ according to (2.4)
    }
}
Repeat the for loop until $y$ = $t$ for all vectors
```

[Gurney, 1997]

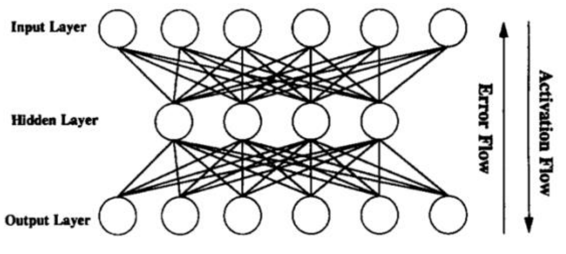
2.1.4 Limitation of perceptron learning rule

First proven by Rosenblatt, the perceptron convergence theorem states that given a linear separatable training set, the perceptron training algorithm will converge to a solution whose decision hyperplane separates and correctly classifies them [Rosenblatt, 1961; Gurney, 1997]. Convergence will not occur when the training set is not linear separatable, but Gurney suggested that dividing the non-linear separatable pattern space into regions that are linear separatable and search for the specific combinations of overlaps within these regions can address this problem [Gurney, 1997].

2.2 Backpropagation

According to Zhou, since perceptron only has one layer of functional neuron, that is, a layer containing an activation function, which is just the output layer, the learning ability of perceptron is weak against more complex problems. In today's world, multi-layer neural networks are much preferred because of their better learning ability and the ability to solve non-linear problems. For a multilayer artificial neural network, because of its complexity, a different learning algorithm needs to be used. Rather than using the perceptron learning rule from (2.1), many learning algorithms, one of which is the error Backpropagation (BP) algorithm [Zhou, 2021]. According to Zhou, the error backpropagation algorithm is by far the most successful learning algorithm [Zhou 2021]. Not only is it capable of training feedforward neural networks, but it also can train other neural networks, such as recurrent neural networks [Pineda, 1987; Zhou, 2021].

2.2.1 Structure



[Wythoff, 1993]

A "BP neural networks" usually referred to as a feedforward neural network trained with the BP algorithm [Zhou, 2021]. According to Wythoff, a BP neural network is a multi-layer neural network with high connectivity across each layer. It is not required to be fully connected between the adjacent layers, and the connection of the nodes can be across one or more layers. The restriction is that the activation is only able to go forward and not backward. The diagram above is a two-layered feedforward network with fully connected between adjacent layers. Without loss of generality, each node in the BP neural network is still the same as in the previous chapter. The input layer takes input from the training set and does no processing, but just distributes the input. Following one or more hidden layer, where the input and output are not from or to the outside world. For the final layer, it is the output layer. Each nodes receive one or more inputs, and produce one output [Wythoff, 1993].

Using the notation from Reed & MarksII, for simplicity, *i* > *j* implies that node *i* follows node *j* in terms of dependency, meaning node *j* is connected to node *j*. The activation/weighted sum α_i of node *i* is

$$\alpha_i = \sum_{j<i} w_{ij} y_j \tag{2.7}$$

where w_{ij} is the weight connection for node *j* to node *i*, and y_j is the output from node *j*, such that it is the result of the weighted sum α_i passed through nonlinear function *f*

$$y_i = f(\alpha_i) \tag{2.8}$$

2.2.2 Error calculation

An error in ANN is the deviation of output from its target, and the goal in error backpropagation, or generally other training algorithms is to reduce each output's error [Zhou, 2021]. There are many interpretations of the error, some use mean square error

(MSE) to calculate the error [Wythoff, 1993], and some use sum of square error (SSE) [Reed & MarksII, 1999]. The sum of square error will be used for the calculation of the error *E* in this paper, such that

$$E = \tfrac{1}{2}\textstyle\sum_p \sum_i \left(\hat{y}_{pi} - y_{pi}\right)^2 \tag{2.9}$$

where *p* indexes the pattern in the training set, *i* indexes the output nodes, $\hat{y}_{pi}$ and y_{pi} are the target and the network output for the *p*th pattern and the *i*th output node respectively [Reed & MarksII, 1999]. From Reed & MarksII, the SSE error function *E* is the sum of squared errors for each pattern. The reason why the MSE *E* is divided by 2 is for simplification of the derivative of it later. Simplifying the error *E* in (2.7) in terms of the pattern error E_p,

$$E = \sum_p E_p \tag{2.10}$$

$$E_p = \frac{1}{2}\sum_i \left(\hat{y}_{pi} - y_{pi}\right)^2 \tag{2.11}$$

The derivative of E respect to connection weight between the *i*th node of the previous layer and the *j*th node of the current layer w_{ij}, using the definition of *E* (2.8),

$$\frac{\partial E}{\partial w_{ij}} = \sum_p \frac{\partial E_p}{\partial w_{ij}}$$

With *k* indexes the output nodes, and α_i as the activation/weighted sum input for node *i*, the derivative can be written as,

$$\frac{\partial E}{\partial w_{ij}} = \sum_k \frac{\partial E_p}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial w_{ij}} \tag{2.12}$$

First, calculate the first term. Let $\delta_i \, be \, the \, \frac{\partial E_p}{\partial \alpha_i}$ for node *i*. The calculation for δ_i is different for output and hidden nodes.

For the output nodes, δ_i can be written as

$$\delta_i = \frac{\partial E_p}{\partial \alpha_i} = \frac{\partial E_p}{\partial y_i} \frac{\partial y_i}{\partial \alpha_i} \tag{2.13}$$

From (2.11),

$$\frac{\partial E_p}{\partial y_i} = -(\hat{y}_{pi} - y_{pi}) \tag{2.14}$$

and by definition of y_i in (2.8),

$$\frac{\partial y_i}{\partial \alpha_i} = f'(\alpha_i) \tag{2.15}$$

where f' is the derivative of the function f.

Then, from (2.13), (2.14), and (2.15), we get δ_i for output nodes

$$\delta_i = -(\hat{y}_{pi} - y_{pi}) f'(\alpha_i) \tag{2.16}$$

For the hidden nodes, let denote *s* as the nodes that node *i* is connected to, where *s*>*i*. Notice that node *i* is only able to affect the error through the output connection to the next *s* nodes, and so the derivative of δ_i can be expressed as

$$\delta_i = \frac{\partial E_p}{\partial \alpha_i} = \sum_s \frac{\partial E_p}{\partial \alpha_s} \frac{\partial \alpha_s}{\partial \alpha_i} \tag{2.17}$$

For the first term $\frac{\partial E_p}{\partial \alpha_s}$, it is the derivative of the pattern E_p with respect to α_s, which α_s is the activation value/weighted sum input of the previous layer *s*th node, and so it is just

$$\frac{\partial E_p}{\partial \alpha_s} = \delta_s \tag{2.18}$$

For the second term $\frac{\partial \alpha_s}{\partial a_i}$, note that *s*>*i*, meaning node *s* takes input from node *i*, and thus α_s is dependent on a_i. By the definition of activation (2.7),

$$\frac{\partial \alpha_s}{\partial a_i} = f'(a_i)w_{ki} \tag{2.19}$$

Then, from (2.17), (2.18), and (2.19), we get δ_i for the hidden nodes, where it depends on δ_s, which is from nodes that are connected to

$$\delta_i = f'(a_i)\sum_s \delta_s w_{ki} \tag{2.20}$$

Therefore, the δ_i is

$$\delta_i = \begin{cases} -\left(\hat{y}_{pi} - y_{pi}\right)f'(\alpha_i) & (for\ output\ nodes) \\ f'(a_i)\sum_s \delta_s w_{ki} & (for\ input\ nodes) \end{cases} \tag{2.21}$$

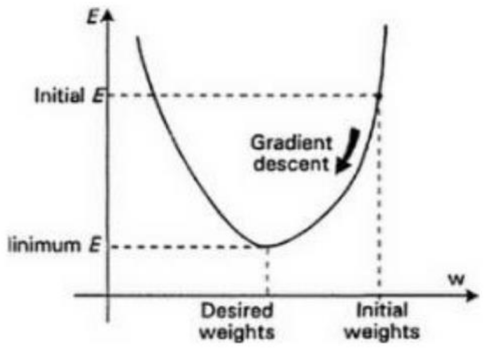
Note that the delta δ_s must be calculated before being able to get δ_i , which means the delta value must be calculated starting from the output nodes and working backward toward to the input nodes, thus the name “backward propagation”. Coming back to (2.12), the second term $\frac{\partial \alpha_k}{\partial w_{ij}}$ is y_j by (2.7), where y_j is the output of node *j*. Then,

$$\frac{\partial E}{\partial w_{ij}} = \delta_i y_j \tag{2.22}$$

The delta value δ_i will not be calculated for the input nodes because it is not involved in adjusting the weights [Reed & MarksII, 1999].

2.2.3 Backpropagation algorithm

To achieve a minimum of error E, change the weights bits by bits towards the direction of the negative error gradient on each step, as known as the gradient descent [Zhou, 2021].



[Gurney, 1997]

By Reed & MarksII, the weight update formula is

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{2.23}$$

The learning rate $\eta > 0$ controls how fast the error descent, and it is advised to use a large learning rate value to decrease the training time if the gradient is very small. The weights are normally initialized with small random values to break symmetry and prevent immediate saturation of the sigmoid nonlinearities. They also mentioned that there are two types of weight update methods.

One of which is batch learning. The tuning of weights is done after all of the pattern p is presented. The procedure is as follows. Each pattern p in the training set is applied and forward propagated. Then, get the pattern error E_p of each respecting pattern and calculate each single-pattern derivative $\frac{\partial E_p}{\partial w_{ij}}$ by backpropagation. Add up each single-pattern derivative, and get the total error derivative $\frac{\partial E}{\partial w_{ij}}$, and update the weights using (2.23). Repeat until satisfied.

Another method is on-line learning. Instead of updating the weights after all pattern *p* from the training set has been evaluated, update it based on a single pattern p. A random pattern *p* from the training set is chosen, then applied and forward propagated it, and get the pattern error E_p. Then, get the pattern derivative $\frac{\partial E_p}{\partial w_{ij}}$ by backpropagation and update the weights with this formula, $\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}$, similar to (2.23). Repeat until satisfied [Reed & MarksII, 1999].

Conclusion

There is a long history in the field of supervised machine learning. It is a vast topic with infinite possibilities and a broad range of opportunities for research, development, and application.

Reference

[Choi et al., 2020] Choi, R.Y., Coyner, A.S., Kalpathy-Cramer, J., Chiang, M.F., and Campbell, J.P., “Introduction to Machine Learning, Neural Networks, and Deep Learning”. *Translational Vision Science & Technology*, 9(2):14–14, 2020

[Gurney, 1997] Gurney, K., *An Introduction to Neural Networks*, CRC Press, London, 1997.

[McCulloch and Pitts, 1943] McCulloch, W., and Pitts, W., “A Logical Calculus of the Ideas Immanent in Nervous Activity”. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[Pineda, 1987] Pineda, F.J., “Generalization of Back-Propagation to Recurrent Neural Networks”. *Physical Review Letters*, 59(19):2229–2232, 1987.

[Reed and Marks II, 1999] Reed, R., and Marks II, R.J., *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, 1999.

[Rosenblatt, 1962] Rosenblatt, F., *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington DC, 1962.

[Wythoff, 1993] Wythoff, B.J., “Backpropagation Neural Networks: A Tutorial”. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993.

[Zhou, 2021] Zhou, Z.-H., “Neural Networks”. *Machine Learning*, 103–128, 2021.