Artificial Neural Networks

Milan Ender

1. Abstract

As of now, Neural Networks are a well-established software tool in a diverse number of different fields in research and industry. They are mimicking the workings of a biological brain on a computer system by reducing it to a simpler model of neural interaction. Similar to their biological origin, they have to be trained towards intelligent behavior. Different approaches to this training based on their architecture have emerged over the last decades, ranging between unsupervised and supervised learning. The former relies on a simulation on which experience can be gathered while the latter needs properly processed datasets. The extracted knowledge alters the strength of the connection between the modeled neurons (or perceptrons) leading to a different response of the system. The arrangement of those connections varies greatly depending on the architecture used. This also determines how the system learns. Once they received proper training, they can become a powerful software tool able to respond to situations beyond their training. Their main capabilities are lying in the detection of patterns and computation of complex systems. Thus they are finding application in a vast number of different fields in science and industry, e.g. in medical records for a certain disease or in the development of stock-market prices.

2. The Perceptron

A perceptron is a model of a biological neuron and thus are the building blocks of every artificial neural network.

2.1 The Biological Origin

A neuron is a cell found in almost all higher organism able to transmit and process information through electrical and chemical signals. They consist of a cell body with a cell core, tree-like dendrites and a axon. The axon can connect to the dendrites of other neurons hence forming networks.
The number of neurons in the brain are estimated to be 10^{11} with 100 000 synapses per neuron[7].

2.2 The model itself

The perceptron is composed of input links from other perceptrons, a processing function and output links.

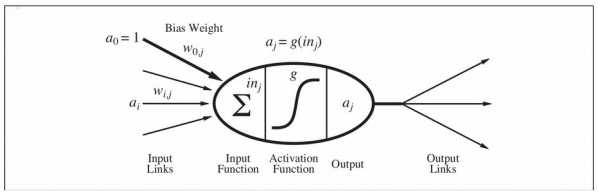

Fig. 1| The perceptron structure

Incoming neural activity is summed up based on the weight and signal strength of the input to in_j. To filter low activity/noise a bias is subtracted from in_j.
The activation function $g(in_j)$ processes in_j into an output signal.
The output/input value range and activation function can vary over the learning algorithm used and influences the behavior of the network in big magnitude.
With a simple step function which turns to 1 when the net-sum is greater than the bias simple logical units can already be realized.
Annotation: neurons and perceptrons will now be used as synonym, understanding them in the artificial context.

3. Methods

3.1 Feed-Forward Networks

The most common architecture for a perceptron network consists of one input, one output and several hidden layers of perceptrons connected respectively to the next layer. However they are not connected back to the previous layer, hence the name.

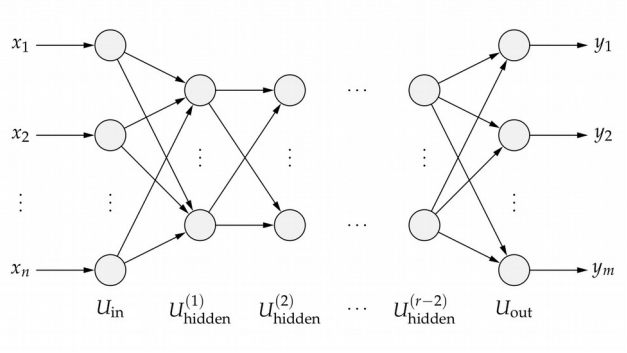

Fig. 2| Feed-Forward network with hidden layers

Once a data set is applied to the first layer, the perceptrons will fire, based on their bias and weights w_{ij}, a signal to the next layer. This will propagate until the signal reaches the output layer. The errors Δ_i will then be calculated based on the training datasets. With this information the weights of the connections will be adjusted based on the error value and activity o_i, e.g. a connection with a large influence/weight will be adjusted to a greater amount than one with a low one. However to accomplish such an adjustment on more than one layer a back propagation of errors is needed, as the error values of all perceptrons is needed in order to adjust every weight.
Hence the necessity of a derivable activation function $g(in_j)$ arises. In this work the most commonly used sigmoid function is used.

$$g(x)=\frac{1}{1+e^{-x}} \qquad \text{(x)}$$

Once this criteria is met, one can utilize the back-propagation algorithm:
Calculate error-signal for every neuron:

$$\Delta_j=\begin{cases} g'(\text{in}_j)*(t_j-o_j) & \textit{outputneuron} \\ g'(\text{in}_j)*\sum_k \Delta_k * w_{jk} & \textit{else} \end{cases} \qquad \text{(x)}$$

Update weights for each neuron according to

$$w_j'=w_j+\lambda*o_j*\Delta_j \qquad \text{(x)}$$

The bias is affected by this algorithm too by simulating it as a perceptron connected respectively to its corresponding neuron with a constant activation value of 1. The weight of this connection becomes the bias value with a negative sign. Hence the bias can be adjusted like any other weight.
The system gained, with the method described above, the ability to learn in a supervised manner based on datasets with known results. The usefulness of such a network is rated by its ability to cope with new datasets, not included in the training set.
The main task of engineering is to find the proper amount of layers and neurons. As of yet, there are little known rules of which configuration applies best to a certain task, leaving it to trial and error or computational optimization, e.g. with genetic algorithms.
Additionally, the learning rate plays a certain role however with much the same problems as the configuration of layers.
The main problem of this design is that the system can be easily tangled up in a local minima.
Many modifications of the back-propagation algorithm have emerged but with none as the universal remedy against the local minima problem. Much more so, each of them have their own strengths and weaknesses, based on the task given, leaving even more room for configuration and optimization.
Another problem arises from over-fitting. When coping exceptionally good with the training sets, the system can be crippled in its ability to respond adequately to sets beyond known ones.

Therefore the performance of a network can vary greatly, with the possibility of little changes leading to dramatical different results.
The main drawback of this approach however is the need for a vast amount of resources and computation time for a reasonable complex task.

3.2 Self-Organizing Map

The approach of the Self-Organizing Map, also known as Kohonen Map (name after its creator) introduces competitive, unsupervised learning to the realm of neural networks. They are able to cluster high-dimensional data to regions of similarity in a 2-dimensional space.

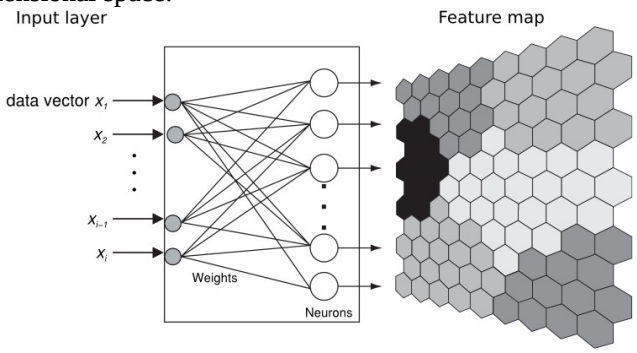

Fig 3. | Structure of the Kohonen map

It is composed of an input layer connected fully to a map layer. The neurons of the map layer are also connected fully together but without any direct feedback. All the weights are initialized at random.
When a data set is applied to the input layer, a center of excitation neuron Z is identified on the map layer.
Here, the neuron with the most similar weight-vector $W_i=(w_1,...,w_2)$ to the input-vector $O=(o_1,\ldots,o_n)$ was chosen as excitation center:

$$|O-W_Z|=min(|O-W_1|,\ldots,|O-W_n|) \quad \text{(x)}$$

A commonly used method is finding the maximum dot-product of the input-values $O=(o_1,\ldots,o_n)$ of each input-neuron and its corresponding weight $W_i=(w_1,...,w_2)$ to a individual neurons.

$$O*W_z=\sum_i O*W_i \quad \text{(x)}$$

However, this showed inferior results compared to the euclidean distance.

Once the neuron Z is known, its weight will be change as well as the weights of the neurons in its neighborhood within a distance r controlled by a learning rate λ. However those farther away will be changed less based on a Gaussian distribution function or something similar. In mathematical terms, this change can be expressed as

$$w_{ij}^{'}=\begin{cases} w_{ij}+\lambda*g_r(d_{jz})*(o_i-w_{ij}) \\ w_{ij} \end{cases} \begin{matrix} d_{jz}<r \\ else \end{matrix} \quad \text{(x)}$$

with the Gaussian distribution function as

$$g_r(x)=\exp\left(\frac{-x^2}{2*r^2}\right) \quad \text{(x)}$$

As the learning-phase progresses, the learning rate λ and radius r should be decreased with each iteration to achieve a better precision and distinguishable borders between clusters.
In this work the method of Ritter et al. (1991) was utilized for the decay of radius and learning rate:

$$r_t=r_{t-1}*\left(\frac{r_{start}}{r_{end}}\right)^{\frac{t}{t_{end}}} \quad \text{(x)}$$

$$\lambda_t=\lambda_{t-1}*\left(\frac{\lambda}{\lambda_{end}}\right)^{\frac{t}{t_{end}}} \quad \text{(x)}$$

The trained map can be visualized by converting the response values of each neuron for each input value to a color gradient.

3.3 K-Means Clustering

After a SOM is sufficiently trained, it is common practice to cluster the map in different regions, which can later be associated with a specific class e.g..
In this work, the trained map is clustered with the K-Means method (Lloyd's Algorithm). The algorithm is applied by iterating step two and three:

1. Initiate vector for the centers of clusters at random.
2. Assign each node to the cluster of which its center has the nearest distance to the node.
3. Recalculate the center of the cluster j with the formula:

$$\vec{c}_j = \frac{1}{\sum_i |\vec{w}_i|} \sum_i \vec{w}_i \qquad \text{(x)}$$

whereas $\vec{w}_i$ is a weight-vector of the node i in the center j.

If no change occurs anymore, the clustering is complete.
Subsequently, a class can be assigned to each cluster and thus unknown data can be classified by identifying the cluster in which the center of excitation lies.

4. Examples

4.1 Classifying with Feed-Forward Networks

This classification task consists of identifying poker hands, e.g. a flush, straight and so on.
The variables were the symbol, ranging from 1 to 13, and color, ranging from 1 to 4, for 5 cards and the resulting hand, ranging from 1 to 9.
To gain better results, the data was translated into a binary system where one input neuron corresponded to only one value and was either 0 or 1.
Overall, there were 1000 sets for training and 1000 for testing.
The network was fully connected and trained with the backpropagation with momentum learning method.
The best results were achieved with one hidden layer with 18 neurons and a learning rate of 0.3.
After enough training the network could recognize most of the hands. Only 85 test examples were classified incorrectly.
Most probably, there were too few examples of these hands available in the training data.

4.2 Classifying with SOM Iris Subtypes

In this example data consisting of the five variables,
1. sepal length in cm,
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica

which describe some genus of the Iris plant, was used to test the network.
In total, 150 datasets were obtained from UCI machine learning repository. Of those, 90 sets were used to train the network and the remaining 60 sets were used to test the effectiveness of the network.

The trained map can be visualized by converting the response values of each neuron for each input value to a color gradient.
As this example uses only four variables, each variable is assigned to a CMYK color channel and then plotted on the neuron map.
In addition, the K-Means method was utilized through the Lloyd-Algorithm to distinguish three clusters, each for one genus.

To determine the training success, 60 samples of unused datasets, 20 sets per genus, were applied on the the net. It was rated as a success if all center of excitation for one genus are found in the same cluster and none of the genuses shared a cluster.

Result

The dataset is composed of 150 examples of measure values of Iris plants, whereof the 3 genuses,"Iris setosa", "Iris verticolor" and "Iris virginica" are distinguished, each with 50 examples. Of the whole dataset only 90 examples were used for training. The leftover examples were later used to examine the ability to classify new data into the right clusters.

It was possible to achieve a accuracy of 99% in the presented network with the dimension of 30x30 neurons. Overall the setosa genus was better distinguishable than the other two (Fig.1), as a higher weight difference corresponds to a higher color contrast, whereas

versicolor and virginica have more similar weights as both have dark shaded areas.

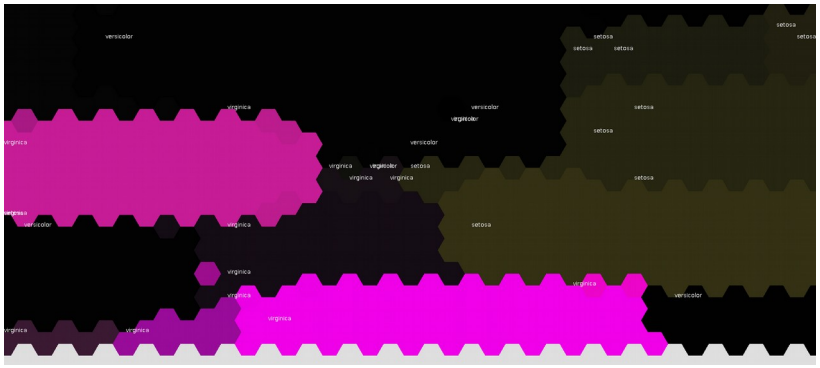

Fig. 4 | 30x30 CMYK Neuron Map after 300 iterations with start radius=5 and learning rate=1.
Dark green corresponds to setosa, pink, black and dark purple to virginica and black to versicolor.

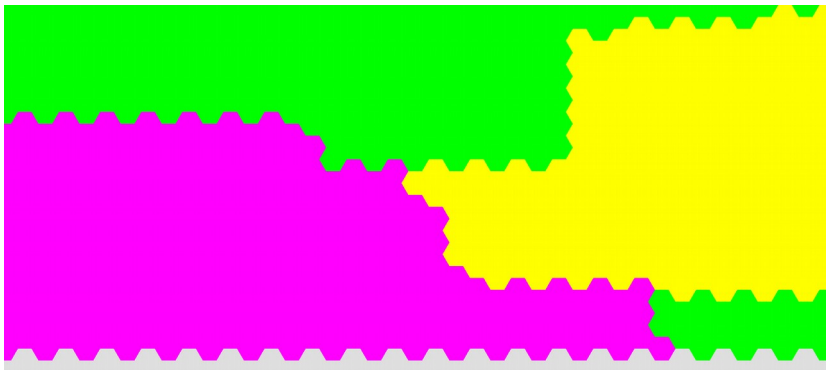

Fig. 5 | 30x30 Clustermap after K-Means algorithm is applied.
Colors: Yellow - setosa; Green - versicolor;
Magenta – virginica.

Discussion

The system was able to learn the difference between the 3 genus to a satisfying degree as only one set was wrongly attributed. A higher iteration count or different map dimensions might accomplish further accuracy, however random tests did not show much better results yet. For further analysis of the limitations of this approach, data with higher dimensionality should be tested. It has already accomplished solid results in facial recognition[1].
Once trained the system was able to classify input data rapidly as the 60 test samples were assigned in under one second, thus gaining significance in real-time software applications such as 3D object recognition for robots.

6.Summary and Outlook

The experiments yielded satisfactory results, as the FFNN could classify 915/1000 (only 8.5% were classified wrong) and the SOM 59/60 test samples successfully. The results were sufficiently clear and reproducible to say that the networks learned systematically and not by coincidence. Even though the networks and tasks were differently, both could mark their significance in data analysis.
Even though the code used was not optimized for efficiency and with using CUDA or similar GPU acceleration, the training for those examples was possible in under 10 minutes with an Intel I5 4200 CPU. This implicates scalability and that larger datasets could be analyzed by using CUDA. Also, optimized learning algorithms could be utilized to further improve accuracy and speed. For example, for FFNN the resilient back-propagation could be used to prevent oscillation of the weights during learning phase.
In the future, more developed derivatives of those algorithms could be assembled and used in conjunction to solve more complex tasks. This was already shown for stacking multiple fully connected FFNN with some modifications resulting in the now very successful Convolutional Neural Networks.
Those combinations could all become important steps on the road to a general AI and an increasingly important topic in current research. It is yet to show, whether some form of consciousness can be simulated on current technology or new advanced computing hardware is needed.

7. Sources

[1]: M. I. Chacon M. and P. Rivas-Perea, Performance Analysis of the Feedforward and SOM Neural Networks in the Face Recognition Problem. CIISP 2007
[2]: Uwe Lämmel and Jürgen Cleve, Künstliche Intelligenz 4th edition.
[3]: Russell S., Norvig P. Artificial intelligence - a modern approach 3rd edition.
[4]: Dataset: http://archive.ics.uci.edu/ml/datasets/Iris 13.01.2016
[5]: Authors code: https://github.com/CptCrispyCrunchy/SOM-Project
[6]: Gurney et al., An Introduction to Neural Networks
[7]: Prof. Dr. Holstein, University of Heidelberg, lecture slides “Entwicklungsbiologie” WS2015/16

8. Table of Figures

9. Index

FFNN	Feed Forward Neural Network
SOM	Self-Organizing Map
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture. A software system for parallel computing on Nvidia graphic cards.