UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS
MSc «INFORMATION SYSTEMS & SERVICES»
«BIG DATA AND ANALYTICS»

MASTER THESIS

«ARTIFICIAL NEURAL NETWORKS: AN OVERVIEW AND APPLICATIONS»

THALASSINOU IOANNA, A.M: ME1603
SUPERVISOR: ASSOCIATE PROFESSOR, FILIPPAKIS MICHAEL

CONTENTS:

Abstract

This dissertation describes the concept of Artificial Neural Networks for Cancer Prognosis and Handwritten Digits Recognition. Artificial Neural Networks inspired by biological neural networks, are efficiently used to model complex relationships between input signals and outputs. When analyzing the detection of cancer cells the model chosen is a Multi-Layer Perceptron, consisted of a variable number of hidden layers, trained using a back-propagation algorithm. Taking into consideration the characteristics of cell nucleus, provided by Breast Cancer Wisconsin dataset, the model can accurately classify the incident between benign and malignant. Using various different settings of parameters and activation functions our model achieved 97.35% accuracy, showing that it can provide an equivalent or even better alternative to human diagnosis. For the handwritten digits recognition task, the MLP trained using back-propagation algorithm achieved 93.9% accuracy. The noticeable is that the model is having difficulty to distinguish digits that often confused by naked eye.

Περίληψη

Η παρούσα εργασία μελετά την έννοια των Τεχνητών Νευρωνικών Δικτύων στον τομέα της Προβλεπτικής Αναλυτικής στην Ιατρική και στην Αναγνώριση Χειρόγραφων Ψηφίων. Το Τεχνητό Νευρικό Δίκτυο είναι ένα μαθηματικό μοντέλο, εμπνευσμένο από βιολογικά νευρωνικά δίκτυα, χρησιμοποιείται αποτελεσματικά για να μοντελοποιήσει πολύπλοκες σχέσεις μεταξύ σημάτων εισόδου και εξόδων. Για το μέρος που αφορά τη διάγνωση καρκίνου, το μοντέλο που επιλέχθηκε είναι ένα Perceptron πολλαπλών επιπέδων, αποτελούμενο από ένα κρυφό επίπεδο, εκπαιδευμένο με αλγόριθμο οπίσθιας διάδοσης. Τροφοδοτώντας το νευρωνικό με μετρήσεις χαρακτηριστικών από τον πυρήνα κυττάρων, που παρέχεται από το σύνολο δεδομένων για τον καρκίνο του μαστού του Wisconsin, το εκπαιδευμένο μοντέλο μπορεί να ταξινομήσει με ακρίβεια το περιστατικό μεταξύ καλοήθους και κακοήθους. Χρησιμοποιώντας διαφορετικούς συνδυασμούς των παραμέτρων και αλλάζοντας τις συναρτήσεις ενεργοποίησης, το μοντέλο πέτυχε 97,35% ακρίβεια, δείχνοντας ότι μπορεί να προσφέρει μια ισοδύναμη ή ακόμα καλύτερη εναλλακτική λύση συγκριτικά με την ανθρώπινη διάγνωση. Για το μέρος της εργασίας που αφορά στην αναγνώριση χειρόγραφων ψηφίων, το MLP που εκπαιδεύτηκε με αλγόριθμο οπίσθιας διάδοσης πέτυχε 93,9% ακρίβεια. Το αξιοσημείωτο είναι ότι το δίκτυο δυσκολεύτηκε να διακρίνει ψηφία που συγχέονται συχνά και από γυμνό μάτι.

ACKNOWLEDGEMENTS

I would like to thank the Associate Professor, Michael Filippakis, as well as the Assistant Professor Orestis Telelis, for their guidance through this process and their contribution to this work.

I would especially like to thank my family for the support and the encouragement I have gotten over these years. Finally, I would like to thank my dear friend and colleague Emmanouil Kavakakis who made the hard times funny and turned coffee into code.

1. Introduction in Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by biological neural networks and are used to approximate functions that are generally unknown. In detail, they are inspired by the behaviour of brain neurons and the electrical signals they transfer between input (like the eyes or nerve endings in the hand), processing, and output from the brain (such as the reaction to light, touch, or cold). The way neurons semantically communicate is an area of ongoing research. Most Artificial Neural Networks are the biologically inspired simulations performed on the computer to execute certain specific tasks like clustering, classification, pattern recognition and many more.

Nowadays, with the explosion of interest in Machine Learning, Neural Networks are mainly used for prediction and classification problems. Due to this, neural networks are successfully applied across an expansible range of domains, in miscellaneous areas like finance, physics, medicine, molecular biology and engineering.

1.1 History

It all began in 1943 when McCulloch and Pitts published an article [1], which showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function. Between 1940's and early 1950's, there were many people examining the subject of neurocomputing, whose job set the stage for later developments rather than causing them.

Mark I perceptron, the first neuro-computer, was created in 1958 by Rosenblatt at Cornell University. The Mark I Perceptron was a linear system (two-layer Perceptron) and was useful for solving problems where the input classes were linearly separable in the input space. In 1990, Hecht-Nielsen showed that a three-layer machine (multi-layer Perceptron, or MLP) was capable of solving nonlinear separation problems.

After many "quiet" years, the Perceptron and ANNs returned to the foreground in 1986 with the rediscovery of a backpropagation algorithm (as proposed by Rumelhart, Hinton, Williams in [2]).

1.2 Feedforward Neural Networks

The Feedforward neural network was the first and simplest type. In this network the information moves only in one direction, from the input layer directly through any hidden layers to the output layer without cycles/loops as depicted in figure 1. Feedforward networks can be constructed with various types of units, the simplest of which is the Perceptron, presented in detail in 1.2.1.

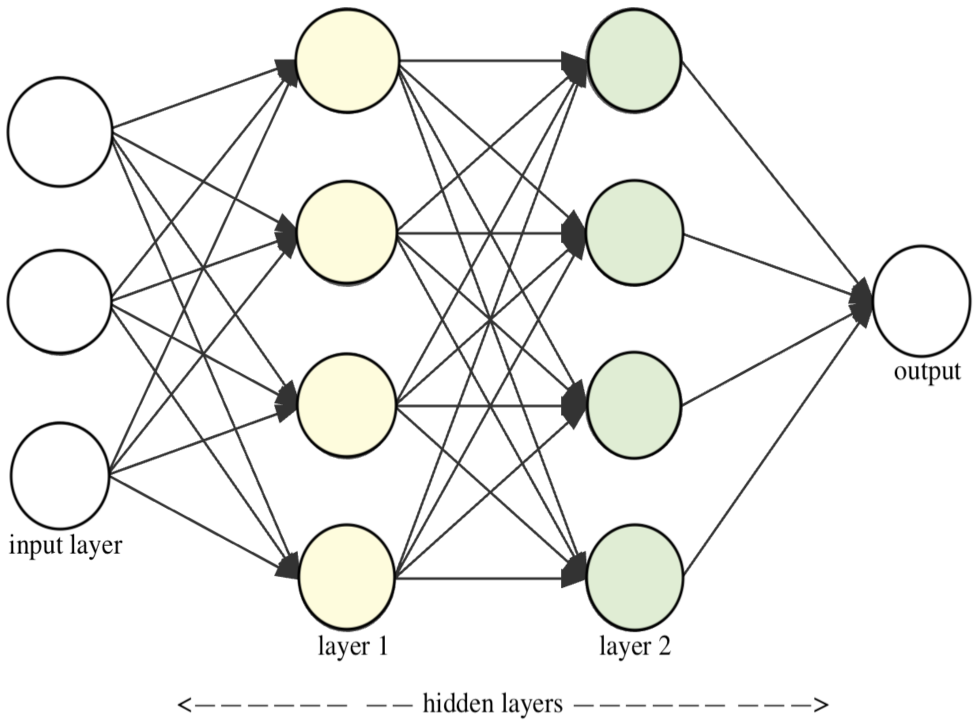


Figure 1. Feedforward ANN architecture [7]

[1] https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

A more detailed analysis based on the above figure about ANN architecture and its layers:

- **Input Layer:** The nodes in this layer provide information from the outside world to the network. No computation is performed in any of these Input nodes. Their role is just to pass on the information to the hidden nodes.

- **Hidden Layer:** The Hidden nodes have no direct connection with the outside world (therefore they are called "hidden"). In this layer, computations are performed, and information is being transferred from the input nodes to the output nodes. A Feedforward network despite having only a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

- **Output Nodes:** The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network (hidden layer) to the outside world.

1.2.1 Perceptron

The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes.
The Perceptron is a linear (binary) classifier and consists of 4 parts the input values, the weight and bias, the net sum and the activation function.
The inputs are fed directly to the outputs via a series of weights. All the inputs x are multiplied with their weights (Figure 2).

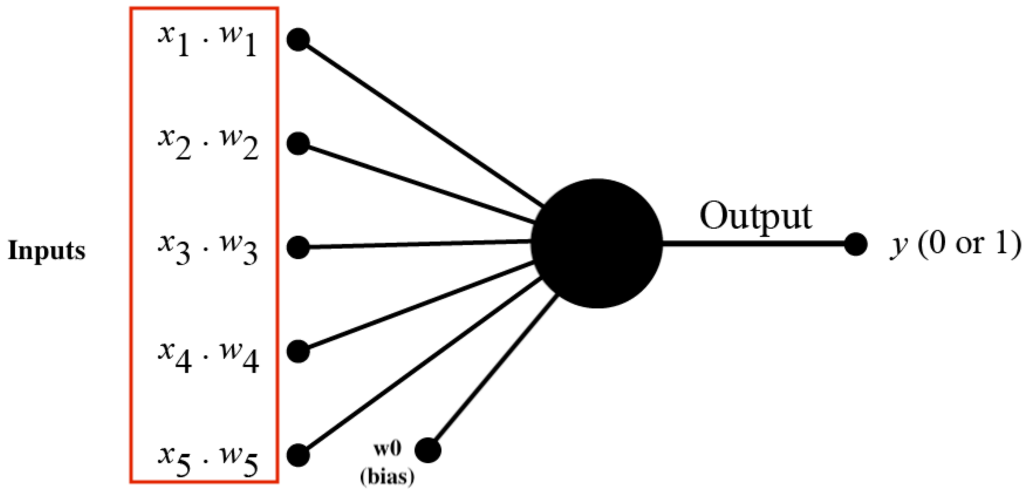


Figure 2. Input process in Perceptron.

All the multiplied values (between inputs and weights) are then added into weighted sum:

$$\Sigma = W_0 + W_1X_1 + W_2X_2 + \ldots.. + W_nX_n$$

Bias node and its importance can be as well described with the constant 'b' of a linear function: y = ax + b.
It allows to move the line up and down to fit the prediction with the data better.

An activation function is then applied to the weighted sum. The activation function applies a step rule to check if the output of the weighting function is greater than zero or not. Activation functions are described in detail in chapter 1.3.

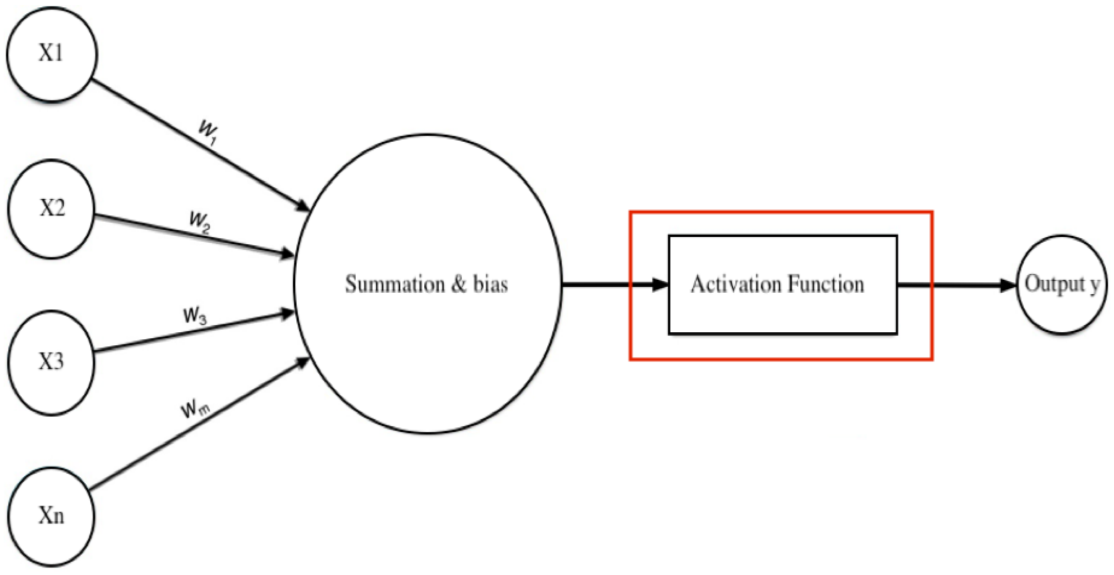


Figure 3. Activation Functions in a Single-layer Perceptron

The sum of the products of the weights and the inputs is calculated in each node, and if the value is above the threshold, that has been set, (typically 0) the neuron fires and takes the activated value (typically 1), otherwise it takes the deactivated value (typically -1).

1.3 Activation Functions and ANN’s

The most significant perspective of ANN ranges in its Activation Function, which introduces non-linearity into the network. A neuron, as depicted in figure 3, calculates a “weighted sum” of its inputs, adds the bias and decides whether it should be fired. Weights and the bias transform the input linearly. The purpose of activation functions is to transform the input non-linearly. This non-linearity gives the ability to the ANN to learn complex transformations between input and output.

The activation functions are basically divided into 2 categories:

- Linear Activation Function
- Non-Linear Activation Functions

Activation functions are usually required to be non-linear, considering their role as mentioned above, to make neural networks non-linear.

1.3.1 Step Function

Step function is one of the most common activation functions in neural networks. It produces binary output that’s why it also called binary step function. The function produces 1 (or true) when input passes threshold limit whereas it produces 0 (or false) when input does not pass the given threshold.

Step function is not suitable for training in the backpropagation algorithm because its derivative is zero, except the zero point, which is infinite. This leads to no change for any value other than zero and no progress can be made. At the point zero, the derivate is infinite, so the step is not manageable either.

In addition to that, even the smallest change in the value of weight can suddenly change the neuron’s output from 0 to 1 and conversely from 1 to 0 (Figure 4).

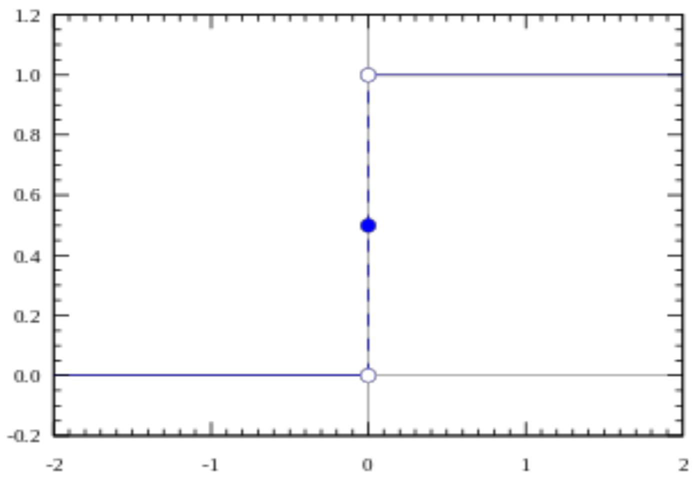


Figure 4. Step Function

With step function as ANN's activation function, some tiny changes in any weight w could lead to the perceptron output to jump suddenly from 0 to 1 (In figure 14: $w+\Delta w = y+\Delta y$). The aim, however, is to have the weights change gradually to produce better results in the output.

1.3.2 Sigmoid Function

Sigmoid functions are assumed to be real-valued and differentiable, and their derivatives exhibit a "bell-shaped" curve over the interval of interest. Sigmoid function has some advantages over the step one:

- It is nonlinear in nature, so its combinations are also nonlinear. This that implies with the fact that we can stack layers in a network.
- It works very well also in non-binary activations. It gives an analogue activation unlike step function.
- It has a smooth gradient (which is used in back-prop algorithm).
- It represents all output values between 0 and 1.

$$A = \frac{1}{1+e^{-x}}$$

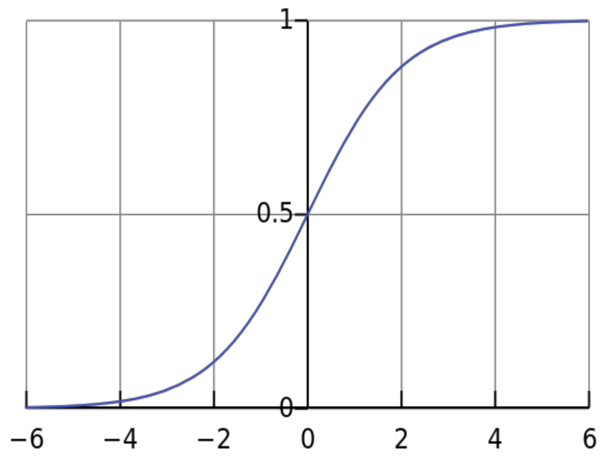


Figure 5. Sigmoid Function

Sigmoid function is widely used because it introduces non-linearity in a model. Without these activation functions, a neural network will be very similar to that of a linear model (with no use of layers). When used by each neuron in a multi-layer neural network, produces a new representation of the original data, and allows for non-linear decision boundary, such as XOR problems.

The problem with sigmoid is when the activations reach near the "near-horizontal" part of the curve on either side. Gradient is small or has vanished and cannot make significant change because of the extremely small value. In this case the network refuses to learn any further or is dramatically slow. This is called the vanishing gradient problem.

Vanishing gradient problem depends on the choice of the activation function. In particularly, sigmoid function crushes its input into a very small output range in a very non-linear fashion. For example, sigmoid maps the real number line onto a strait range of [0, 1], as mentioned above. This causes, large regions of the input space to be mapped to an extremely small range. In these regions of the input space, even a large change in the input will produce a small change in the output - hence the gradient is small.

In the backpropagation algorithm the gradient of sigmoid is being used:

Consider σ(x) = $\frac{1}{1+e^{-x}}$ and let's compute the gradient of σ(x), $\frac{d\,\sigma(x)}{dy}$:

$$\frac{d}{dx}\sigma(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right]$$

$$= \frac{d}{dx}(1+e^{-x})^{-1}$$

$$= -(1+e^{-x})^{-2} \cdot (-e^{-x})$$

$$= \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)$$

$$= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x) \cdot (1 - \sigma(x))$$

1.3.3 Hyperbolic tangent Function (Tanh)

Tanh Function is a scaled sigmoid Function, that's why it looks a lot like it:

- It's also nonlinear in nature, which means the multiple layers can be stacked.
- It is bound to range (-1, 1).
- The gradient is stronger for tanh than sigmoid, its derivatives are steeper.

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 = 2\text{sigmoid}2(x) - 1$$

Deciding between the sigmoid or tanh will depend on the requirement of gradient strength. Like sigmoid, tanh also has the vanishing gradient problem.

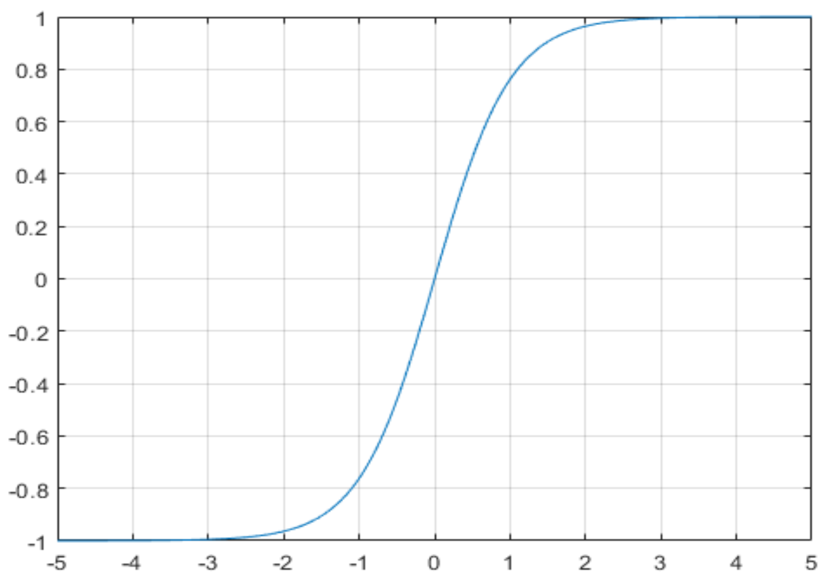


Figure 6. Tanh(x) Function

1.3.4 ReLu Function

ReLu (Rectified Linear Unit) Function is nonlinear in its nature. It gives an output x, if x is positive, and 0 if x is negative. Its range is [0, inf].

$$A(x) = \max(0, x)$$

ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

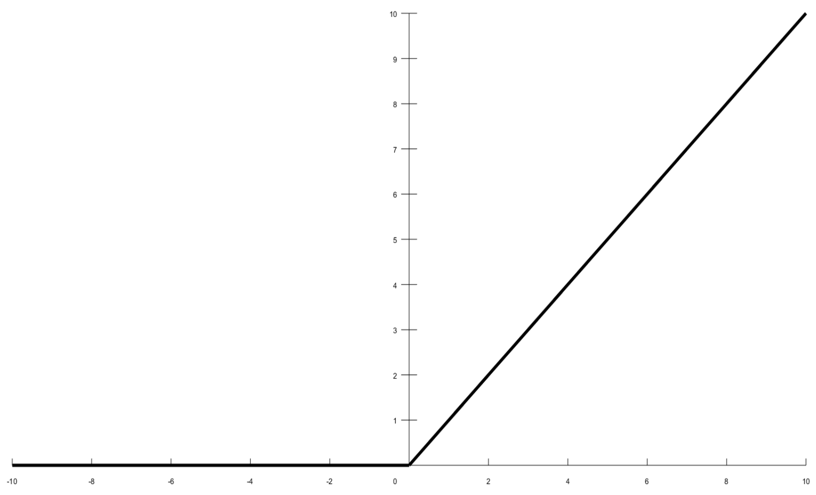


Figure 7. ReLu Function

1.3.5. Softmax Function

Softmax function calculates the probabilities distribution of the event over ‘n’ different events. This function will calculate the probabilities of each target class over all possible target classes. The calculated probabilities will be helpful for determining the target class for the given inputs. Softmax properties:

- The calculated probabilities will be in the range of 0 to 1.
- The sum of all the probabilities is equal to one.

$$\text{softmax}(y)_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

The main advantage of using Softmax is the output probabilities range. If softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.
The formula computes the exponential (e-power) of the given input value and the sum of exponential values of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function.

1.4 Other Feed Forward Networks

1.4.1 Multi-layer perceptrons

A Multi-Layer Perceptron (MLP) contains one or more hidden layers, in addition to one input and one output layer). Unlike, a single layer perceptron which can only learn linear functions, a multi-layer perceptron can also learn non – linear functions.

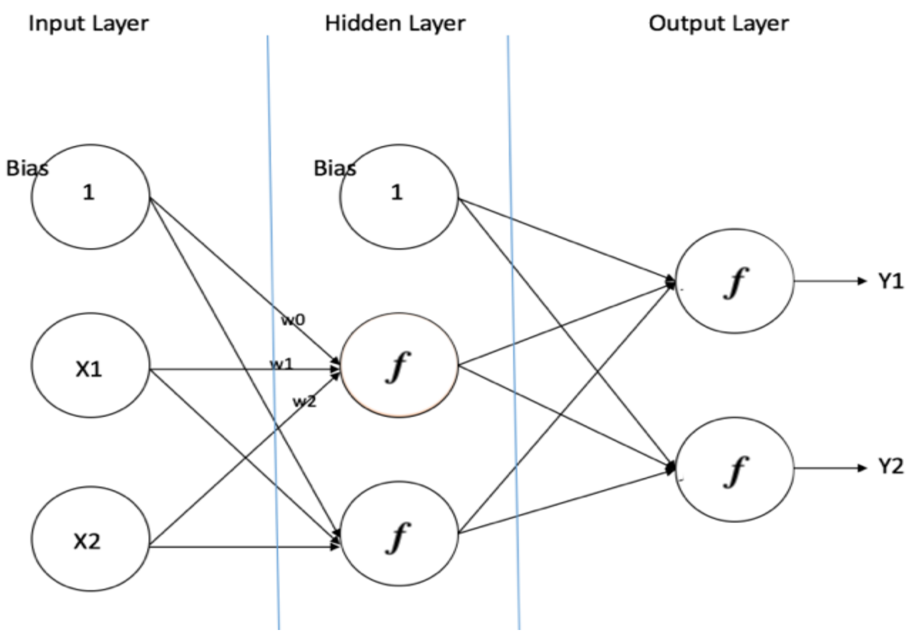


Figure 8. A multi-layer perceptron with one hidden layer

- **Bias node:** Allows to move the line up and down to fit the prediction with the data better. Just like the linear function y = ax + b, without b the line always goes through the origin (0,0) causing a poorer fit.
- **Input Layer:** The Input layer of figure 2, has three nodes. The Bias node has a value of 1. The other two nodes take X1 and X2 as external inputs, depending upon the input dataset. The outputs from nodes in the Input layer are 1, X1 and X2 because no computations take place in this layer. These values feed the Hidden Layer.
- **Hidden Layer:** In this layer we also have three nodes with the Bias node having an output of 1. The output of the other two nodes in the Hidden layer depends on the outputs from the Input layer (1, X1, X2) as well as the weights associated with the connections (or edges). The ***"f"*** refers to the activation function. These outputs are then fed to the nodes in the Output layer.
- **Output Layer:** The Output layer has two nodes which take inputs from the Hidden layer. The values calculated (Y1 and Y2) as a result of these computations are the outputs of the Multi-Layer Perceptron.

1.4.2 Adaptive Linear Neuron (Adaline)

The Adaline is a network developed by d Widrow and Hoff at Stanford University in 1960, having one single linear unit. Its architecture is similar to perceptron, except having one extra feedback loop which is used to compare the actual output with the desired output. It uses a bipolar activation function, weights, bias which are adjustable and delta rule in training phase for minimizing the MSE (Mean-Squared Error) between the actual and the target output, as described in [14].

The difference between the Adaline and the Perceptron is in the learning phase. The Perceptron uses the class labels to learn model coefficients. The Adaline, in the other hand, uses continuous predicted values to learn model coefficients, which is more "powerful", since it witnesses how much wrong or right the predicted result is. In figure 8 this difference is depicted in the two networks' architecture.

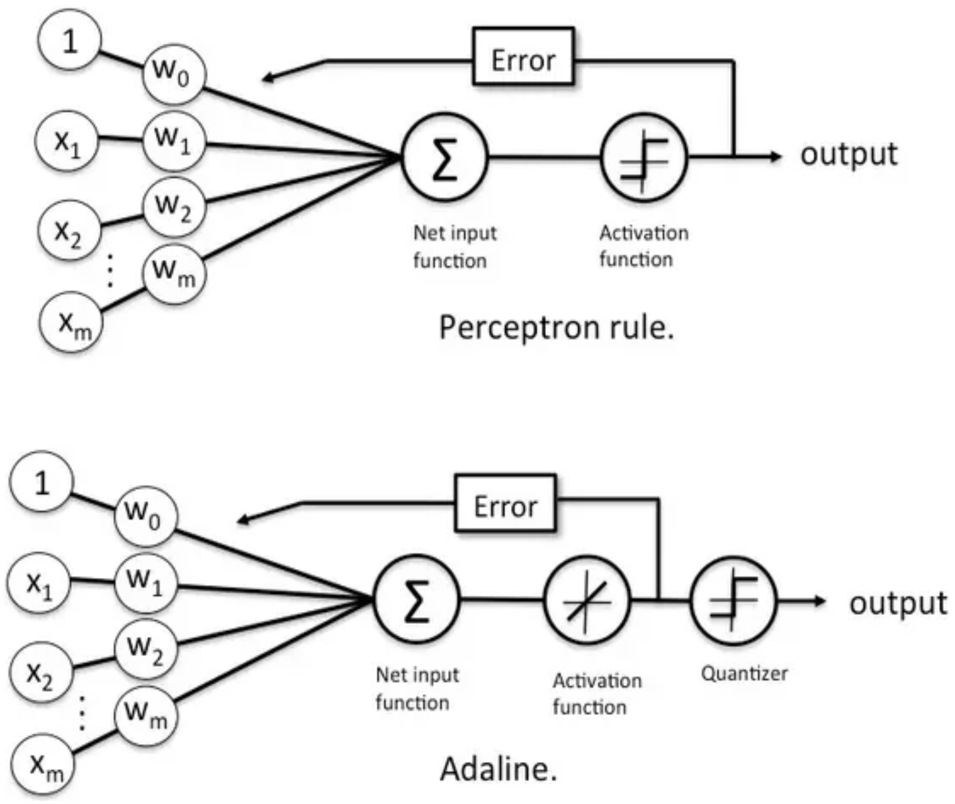


Figure 9. Difference in architecture between Perceptron and Adaline (Source: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html)

1.4.3 Multiple Adaptive Linear Neuron (Madaline)

Madaline is a network which consists of many Adaline networks in parallel. It is like an MLP, where Adaline acts as the hidden layer between the input and the Madaline layer. The weights and bias between the input and Adaline layers remain adjustable like in standard Adaline's architecture.

Learning algorithm for Madaline is known as MRII, for Madaline Rule II. In 1988, Winter and Widrow in [10] showed that MRII has the ability to derive useful generalizations when training adaptive nets on even 1% of the data given (input).

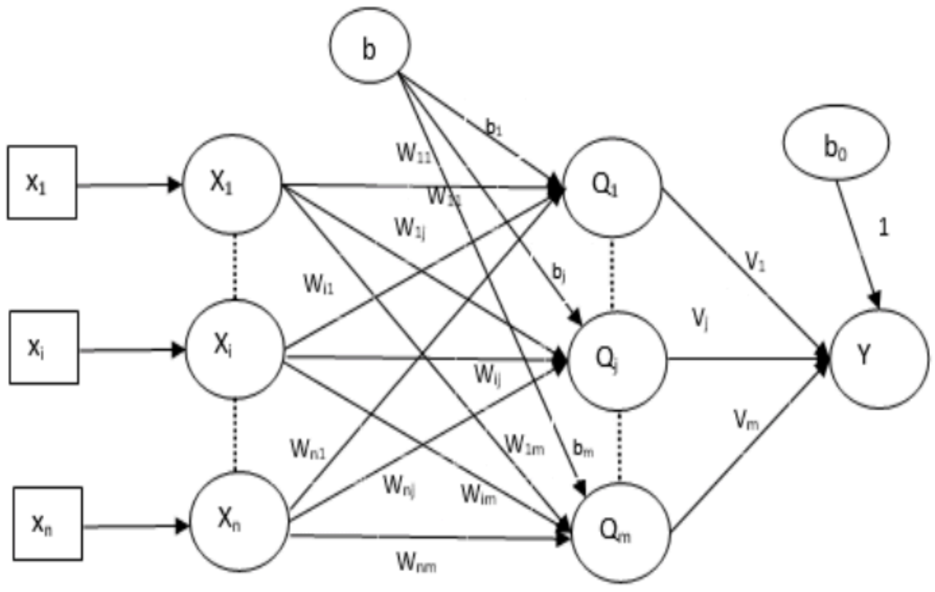


Figure 10. Architecture of Madaline Network
(Source:
https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_supervised_learning.htm)

1.5 Training algorithms

1.5.1 Back-propagation algorithm

The Backpropagation algorithm as aforementioned in 1.1 section, originally introduce in 1970's, but it's importance fully appreciated in paper written by Rumelhart, Hinton, and Williams in 1986 [20].

The Backpropagation algorithm belongs to supervised learning techniques, which means that the network learns from labelled training data. The output values are compared with the correct answer to compute the value of some predefined error-function (in most cases with sigmoid function). The error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. This process is repeated until the output error is below a predetermined threshold. Once the above algorithm terminates, the ANN has been trained and its able to work with new inputs.

The Backpropagation mode is preferred versus the forward one, because it's computationally cheaper. Forward-wise algorithm leads to a multiplication of large matrices for each network's layer till the output layer which would result as a multiplication of a large matrix by a vector. Unlike, the backwards approach which starts with a multiplication of a matrix by a vector, which leads to another vector and so on. So, the efficiency of Backprop algorithms is that matrix-vector multiplications instead of matrix-matrix multiplications are taken place.

1.5.2 Backpropagation Basics

One of the most important elements of Back-prop algorithm is the gradient computation, which leads to the calculation of the new synaptic weights. As depicted in the picture below, the re-calculation of the new weights begins from the output and gradually moving backwards to the input layer. The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

The Backpropagation algorithm uses the chain rule, to calculate the derivatives from one layer to another. The Chain Rule is defined as follows:

$$\frac{df}{dx} = \frac{df}{du} \cdot \frac{du}{dx}$$

The figure below showcases how the chain rule works in computational graphs, backward pass. For example:

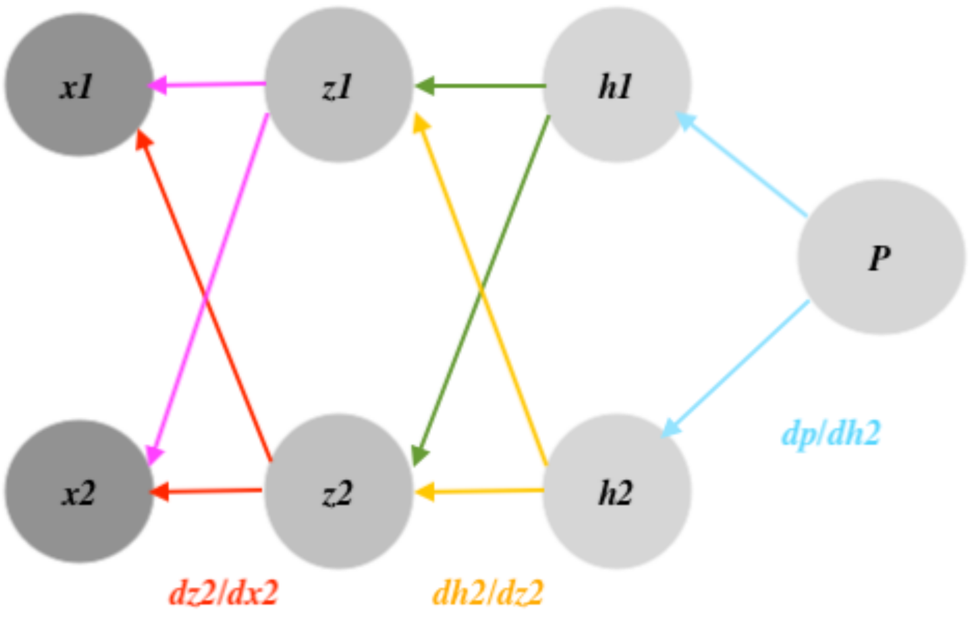


Figure 11. Chain Rule in applied Backward pass

$z_1 = z_1\,(x_1, x_2)$ $\quad$ $h_1 = h_1\,(z_1, z_2)$ $\quad$ $P = P\,(h_1, h_2)$

$z_2 = z_2\,(x_1, x_2)$ $\quad$ $h_2 = h_2\,(z_1, z_2)$

The computation of partial derivative of P with respect to x_1, by applying the chain rule:

$$\frac{\partial p}{\partial x1} = \frac{\partial p}{\partial h1} \cdot \frac{\partial h1}{\partial x1} + \frac{\partial p}{\partial h2} \cdot \frac{\partial h2}{\partial x1}$$

$$\frac{\partial h1}{\partial x1} = \frac{\partial h1}{\partial z1} \cdot \frac{\partial z1}{\partial x1} + \frac{\partial h1}{\partial z2} \cdot \frac{\partial z2}{\partial x1} \quad , \quad \frac{\partial h2}{\partial x1} = \frac{\partial h2}{\partial z1} \cdot \frac{\partial z1}{\partial x1} + \frac{\partial h2}{\partial z2} \cdot \frac{\partial z2}{\partial x1}$$

$$\frac{\partial p}{\partial x1} = \frac{\partial p}{\partial h1} \cdot \left(\frac{\partial h1}{\partial z1} \cdot \frac{\partial z1}{\partial x1} + \frac{\partial h1}{\partial z2} \cdot \frac{\partial z2}{\partial x1}\right) + \frac{\partial p}{\partial h2} \cdot \left(\frac{\partial h2}{\partial z1} \cdot \frac{\partial z1}{\partial x1} + \frac{\partial h2}{\partial z2} \cdot \frac{\partial z2}{\partial x1}\right)$$

The figure below depicts the process of back-propagation algorithm applied in an ANN, in order to redefine the values of the weights.

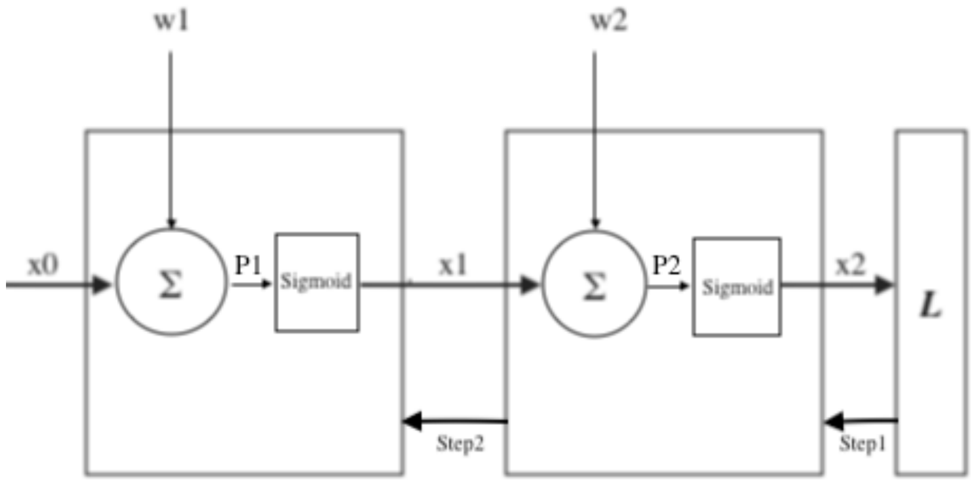


Figure 12. Back-propagation process in Artificial Neural Network.

In the forward pass the error is calculated relative to the desired output. The objective is to minimize the error across all training samples. This process requires the recalculation of weights w1 & w2 by applying the back-propagation algorithm. Assuming that the activation function, in the example of figure 11, is sigmoid function and the loss function is E $= \frac{1}{2}(Y - X2)^2$.

Moving backwards, from step1 in order to recalculate w2 by applying the chain rule:

$$\frac{\delta E}{\delta w2} = \frac{\delta E}{\delta X2}\frac{\delta X2}{\delta P2}\frac{\delta P2}{\delta w2} \quad (1)$$

- $\frac{\delta E}{\delta X2}$ = (Y- X2)
- $\frac{\delta X2}{\delta P2}$ is the derivative of sigmoid activation, $\frac{\delta \sigma(x)}{\delta(x)} = \sigma(x)\,(1 - \sigma(x))$,

so $\frac{\delta X2}{\delta P2}$ = X2 · (1 – X2)

- $\frac{\delta P2}{\delta w2}$ = X1

Combining them in $^{(1)}$ → $\frac{\delta E}{\delta w2}$ = (Y- X2) · X2 · (1 – X2) · X1

The new value of w2 is : w2 new = w2- α · $\frac{\delta E}{\delta w2}$

Moving backwards, from step2 in order to recalculate w1 by applying the chain rule:

$$\frac{\delta E}{\delta w1} = \frac{\delta E}{\delta X1}\frac{\delta X1}{\delta P1}\frac{\delta P1}{\delta w1} \quad (2)$$

- $\frac{\delta X1}{\delta P1}$ is the derivative of sigmoid activation, $\frac{\delta \sigma(x)}{\delta (x)}$ = σ(x) (1 – σ(x)) ,

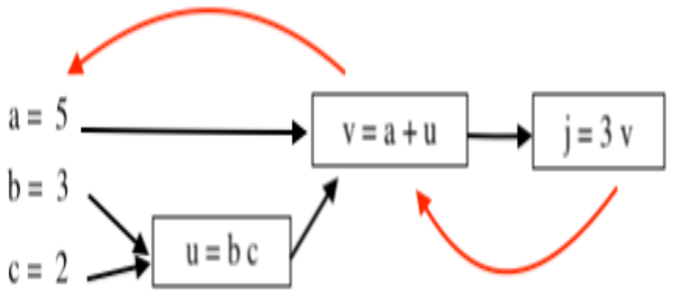
so $\frac{\delta X1}{\delta P1}$ = X1 · (1 – X1)

- $\frac{\delta P1}{\delta w1}$ = X0

- $\frac{\delta E}{\delta X1}$, in order to compute this derivative, the chain rule must be applied again:

so $\frac{\delta E}{\delta X1} = (\frac{\delta E}{\delta X2} \frac{\delta X2}{\delta P2} \frac{\delta P2}{\delta X1})$ = (Y- X2) $^{(3)}$ · X2 · (1 – X2) $^{(4)}$ · w2 , [3, 4 have been computed from previous step.

A very simple example with equations, about how the chain rule works in the scheme below:



$$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = 3 \cdot 1$$

1.6 Convolutional Neural Networks

Convolutional Neural Networks (ConvNets or CNNs) are a category of ANNs, successfully used in the field of pattern recognition and classification within images. A simple Convolutional Network is a sequence of layers that produce by transforming the input using some activation function. The layers used are the Convolutional layer, the RELU layer, the Pooling layer, the Dropout Layer and finally the Fully Connected Layer.
While some layers (RELU/Pooling) perform a fixed function on their input, others (Convolutional/Fully Connected layer) take into consideration additional parameters like the biases and weights of the Neurons.

Here’s a brief description of each layer and its role in the network:

Convolutional Layer: Produces the Activation or Feature map by applying various filters (feature identifiers) to the input image.

RELU Layer: Because throughout the system, the operations used are linear, the gradient decreases exponentially. This results in the lower layers of the system to train slowly. In order to help with the problem of the vanishing gradient, the RELU layer is used to inject nonlinearity to the system. The function used is the $f(x) = \max(0, x)$ to all the input values. Other nonlinear functions like the sigmoid can also be used.

Pooling layer: It is often applied after the RELU layers and it outputs the maximum number in every filter sub region. This helps in decreasing the volume of calculations in subsequent layers and more importantly controls the chance of the model over-fitting the training data.

Dropout Layer: Without the use of the Pooling and Dropout layer the weights of the network are so tuned to the training samples that don't perform well on new samples. The Dropout layer randomly removes a set of activations by setting their values to zero.

Fully Connected layer: It's the final layer of the network. It receives the output of the previous layers and outputs an N dimensional vector where N is the number of classes that the network has to identify. Each member of this array represents the probability of a certain class.

1.6.1 Inception V3

In 2014 Google proposed a deep convolutional neural network codenamed Inception that achieved a new state of the art score for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014. The first incarnation of this architecture, Google Net, a 22-layer deep network has been subjected to further improvement with its latest incarnation being Inception V3.

The real power of the Inception network is its ability to use its prior learned knowledge in order to be able to solve more specific classification problems. This is achieved through transfer learning and the ability it gives to users to retrain its final layer (fully connected layer) depending on their specific needs.

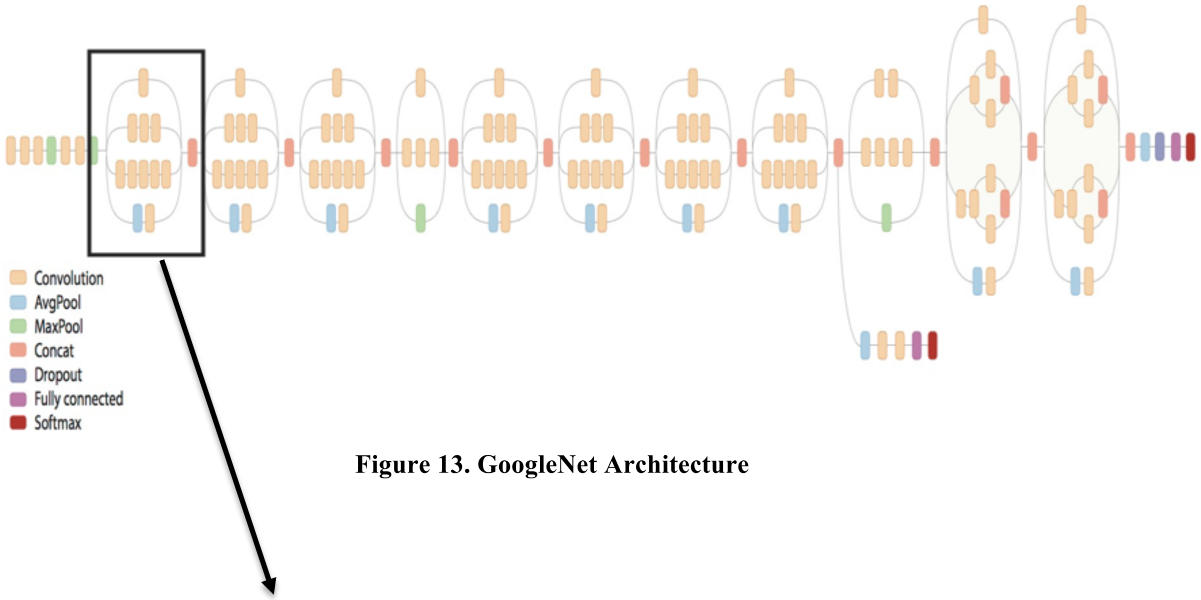


Figure 13. GoogleNet Architecture

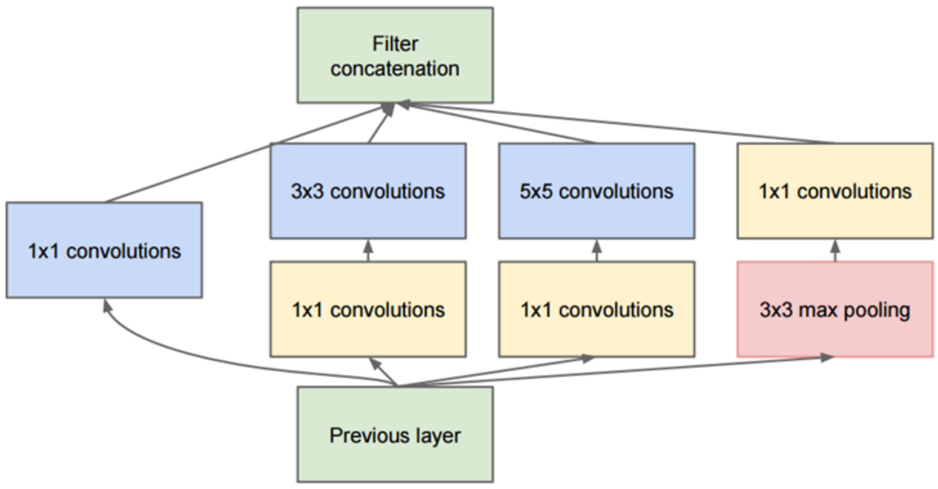


Figure 14. Inception V3 Node

2. Neural Networks and Artificial Intelligence

An Artificial Neural Network consists of an interconnected group of artificial neurons, which use mathematical or computational models, to process given information. They are widely used to model intricate relationships between inputs and outputs, as well as to find hidden patterns in data. As aforementioned, application areas of ANN's include medical diagnosis, finance applications, data mining, etc.

Neural Networks have been successfully used for Bankruptcy predictions. The prediction of bankruptcies is an extensively studied subject since it is inextricably tied with a banks' lending decisions and therefore its profitability. Specifically, in the work of Atiya [3], achieves 85.5% accuracy (from 81.46%) on the Bankruptcy classification problem. This was achieved by introducing a number of new indicators (book value/total assets BV/TA;, cash flow/total assets CF/TA;, rate of change of cash flow per share ROC(CF); etc) to Merton's already existing prediction model.

Medical diagnosis via image recognition is also an area, where ANN's contribution has proven to be more than useful. In 2017, Esteva et al. [4], trained a deep convolutional network with a 192,450 clinical images of skin disease in order to classify skin lesions. The outcome was an algorithm that could classify lesions from photographic images similar to those taken with a mobile phone. Their model's performance in detecting malignant melanomas and carcinomas, was tested against certified and trained dermatologists. The system was used in two different use cases, keratinocyte carcinomas versus benign seborrheic keratoses; and malignant melanomas versus benign nevi [5], [6]. The authors suggest that this technique could be used outside the clinic as a visual screen for cancer. More specifically, mobile devices supplied with these deep neural networks (like smartphones) can possibly provide low cost access to vital diagnostic care.

Another work that proves how useful can ANNs be in the medical sector is by Snow, et al [7]. Through their work they pursued to determine, whether ANNs would be helpful to predict biopsy results in men with uneven screening test(s) and to predict treatment effect after radical prostatectomy. Their proposed neural network, predicted the biopsy result with 87% accuracy and the probability of tumor recurrence with 90% accuracy. They conclude, that trained neural networks can be very helpful in decision making for prostate tumor patients.

2.1 Model Selection of ANN

As mentioned above (Chapter 1.2), an Artificial Neural Network consists of 3 layers. The input, the hidden and the output layer. The most challenging and difficult part when designing an ANN, is to decide the number of hidden layers, if needed, as well as the number of nodes in hidden layers.

2.1.1 Input & Output Layer Neurons

Being able to identify the number of input and output layers and number of their neurons is the easiest part of the design procedure. Every network has a single input and output layer. The number of neurons in the input layer is equal to the number of input variables in the dataset being processed. The number of neurons in the output layer is equal to the number of outputs associated with each input.

For example, having to design an ANN for the iris dataset (for dataset information & description, https://archive.ics.uci.edu/ml/datasets/iris). The number of nodes constituting the input layer would be four which equals to the number of attributes (sepal length in cm, sepal width in cm, petal length in cm & petal width in cm). The number of nodes constituting the output layer equals to three, which are the three different classes/species (Iris Setosa, Iris Versicolour & Iris Virginica).

2.1.2 Hidden Layers and Neurons

The challenge is knowing the number of hidden layers and their neurons. The number of hidden layers depends on the complexity of the data. In case where the data is linearly separable, a hidden layer is not even necessary.

Deciding the number of neurons in the hidden layers is a decisive part of the neural network's overall architecture, since these layers have a tremendous influence on the final output. Both the number of hidden layers and the number of neurons in each of them must be carefully examined.

Using too few neurons in the hidden layers will result in something called under fitting. Under fitting is a state when there are too few neurons in the hidden layers to sufficiently detect the signals in a complicated data set. In other words, the neural network created is not complex enough to map with precision the relationship between a dataset's attributes and a target output.

On the other hand, it's important to keep in mind that using too many neurons in the hidden layers comes at a cost. The most common problem is over fitting. Over fitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all the neurons in the hidden layers. This means that the network has simply memorized the training data but has not learned to generalize to new examples.

Over fitting can be a serious problem when training a neural network, especially in cases where the dataset consists of many parameters. Srivastava, et al in their work [15], introduce the Dropout method, proving with experiments that is a way to prevent neural networks from over fitting.

Too many neurons in a neural network, apart from over fitting, can also have another cost. An overly large number of neurons in the hidden layers can increase the learning time to the point that is impossible to sufficiently train it.

In conclusion, an underfed model will have high training and testing error and an overfit model will have extremely low training error but high testing error. Obviously, there must be a balance between over fitting and under fitting in order for a network to be accurate and have good fit. Van der Wmp Wil, et al in [16] proposed a two-step approach to control in some way the balance between over fitting and under fitting.

2.2 Man versus Machine

2.2.1 Anatomy of a Biological Neural Network

Biological neurons, as depicted in figure 15, consist of a cell nucleus, which receives input from other neurons through a web of input terminals, or branches, called dendrites. The combination of dendrites is often referred to as a "dendritic tree", which receives stimulus or inhibitory signals from other neurons via an electrochemical exchange of neurotransmitters.

The extent of the input signals, that reach the cell nucleus depends both on the width of the action potentials propagating from the previous neuron and on the conductivity of the ion channels feeding into the dendrites. The ion channels are responsible for the flow of electrical signals passing through the neuron's membrane.

More frequent or larger magnitude input signals generally result in better conductivity ion channels, or easier signal propagation. Depending on this signal aggregated from all synapses from the dendritic tree, the neuron is either "activated" or "inhibited", or in other words, switched "on" or switched "off", after a process called neural summation. The neutron has an electrochemical threshold, analogous to an activation function in artificial neural networks, which governs whether the accumulated information is enough to "activate" the neuron. The result is then fed into other neurons and the process begins again.

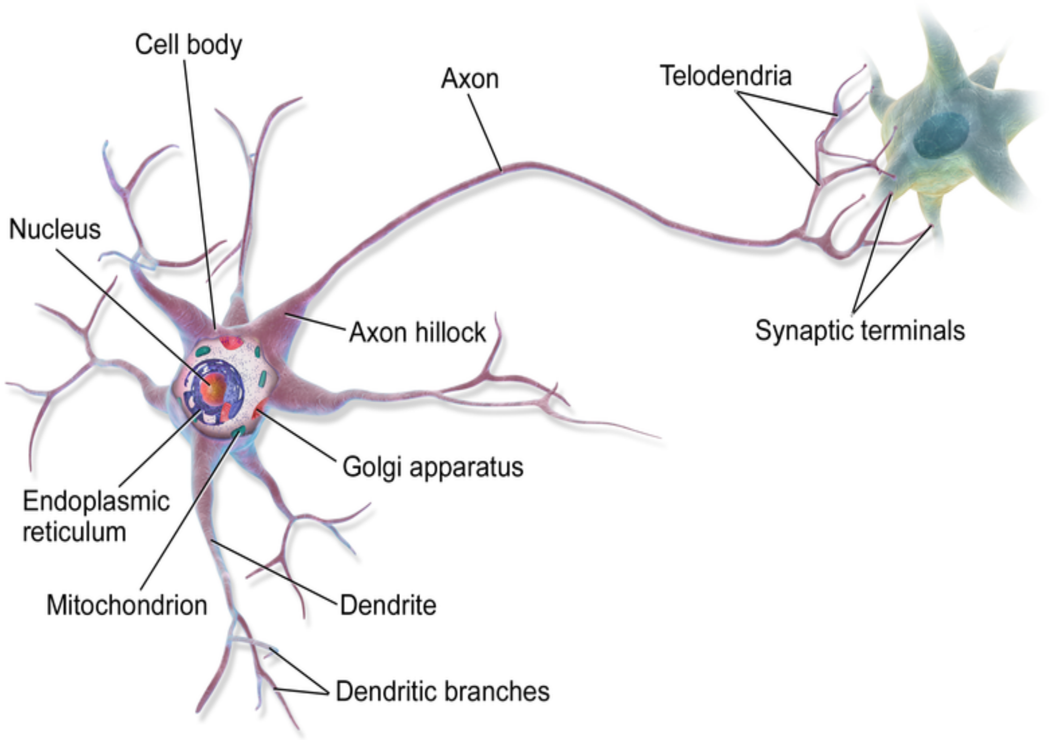


Figure 15. Anatomy of Biological Neural Network (Source: Wikipedia)

2.2.2 Learning in Biological Neural Networks

In biological neural networks, like these in mammal brain, learning is achieved by making small pinches to an existing representation, its configuration contains significant information before any learning is conducted. The strengths of connections between neurons, or weights, do not start as random, are genetically derived as a product of human evolution.

Over time, the network learns how to perform new functions by adjusting both topology and weights. The fact that there is an initial representation that works well for many tasks is supported by research, which suggests that as young as one-month old new-borns are able to recognize faces demonstrated by their learning to differentiate between strangers and their parents. In other words, the concept of a human face has largely been passed down genetically from parent to child.

The same phenomenon applies to other tasks as well. In the sense of "tasks" both passives tasks, such as recognizing generic objects to process sound as speech patterns, and active tasks, like movement and speech, are encompassed. These skills are learned gradually, and progressively smaller tweaks are used to refine them. The precise topologies are a function of the types of stimuli upon which these biological neural networks are trained. A prominent example is the monocular deprivation studies led by Hubel and Wiesel [19]. The study involved forcing an animal's eye shut for two months during development and observing the changes to their primary visual cortex.

2.2.3 Learning in Artificial Neural Networks

Contrariwise to Biological Neural Networks, Artificial Neural Networks (ANNs), are commonly trained from scratch, using a fixed topology chosen for the problem. In the present, their topologies do not change over time and weights are randomly initialized and adjusted via an optimization algorithm to map the input data to the desired output.

Nevertheless, ANNs can also learn based on a pre-existing representation. This procedure is known as fine-tuning. The process of fine-tuning consists of conforming the weights from a pre-trained network topology at a relatively slow learning rate in order to perform well on newly supplied input training data.

Whether training from scratch or fine-tuning, the weight update process begins by passing data through the neural network, measuring the outcome, and modifying the weights based on the deviation between the target and the desired output. This whole process represents how an artificial neural network "learns". Weights are gradually pushed in the directions that most increase performance of the desired task like maximizing recognition accuracy. This concept of learning can be compared to a child learning how to recognize, for example, animals. After failed attempts and feedback on the accuracy of the answer, the child tries until achieving the correct response.

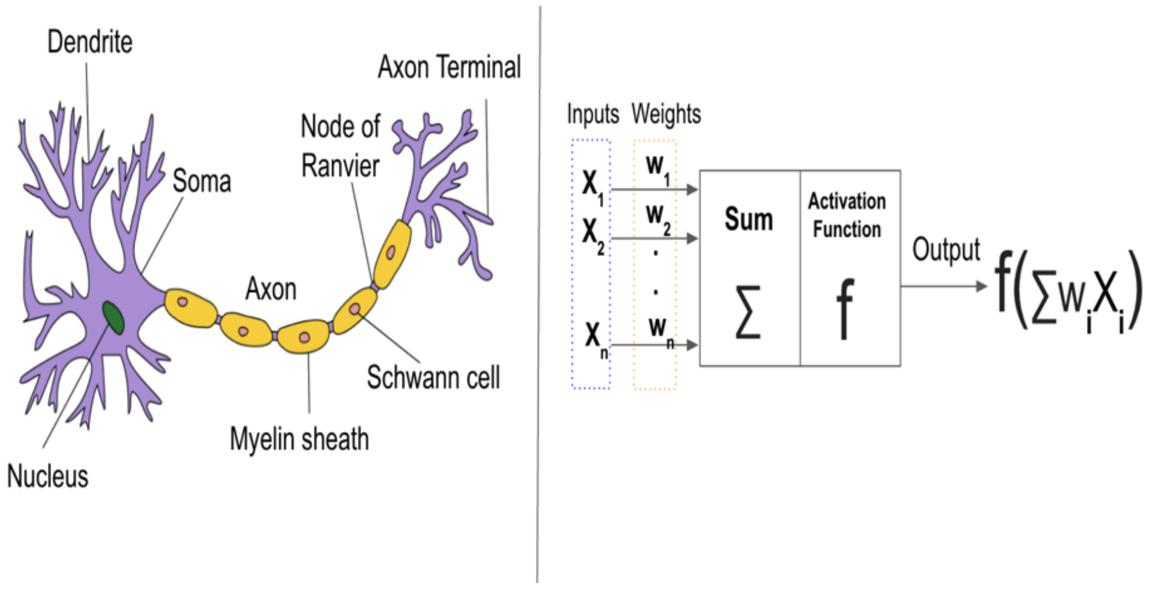
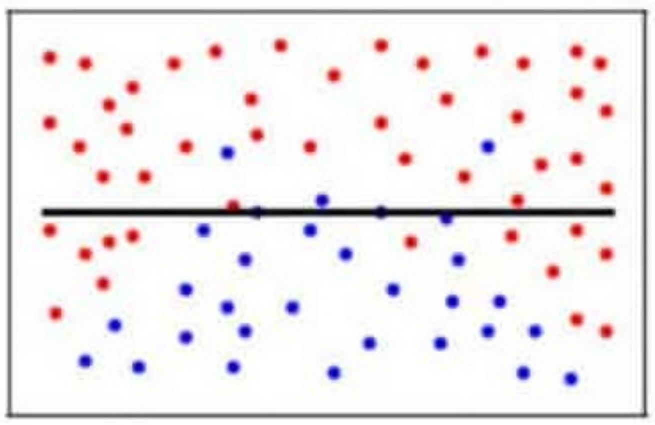


Figure 16. Structure of a typical neuron (left) VS artificial neuron (right) (Source: Wikipedia)

2.2.4 Learning Methods and ANN's

The concept of learning is to gain knowledge of by study, experience or by being taught by connoisseur. There are two types of learning, either talking about people learning or machine learning.

Supervised Learning is the most common for pattern, speech and text recognition for artificial neural networks. With this procedure, the training data the network is being fed with, includes both the input and the desired results. Via this procedure, there is an input x, an output Y and an algorithm which is used in order, the ANN, to learn the mapping function between x and Y (Y=f(x)). These methods are fast and accurate, as long as a proper training, validation and test set has been constructed. The most common, disadvantage that these methods can result to is the phenomenon of "over fitting" (Figure 17). This happens when the network, fails to learn the underlying function and just despots the data. This process results having a poorly trained network that has great performance on the data used in training phase, but really bad when it comes to new data.



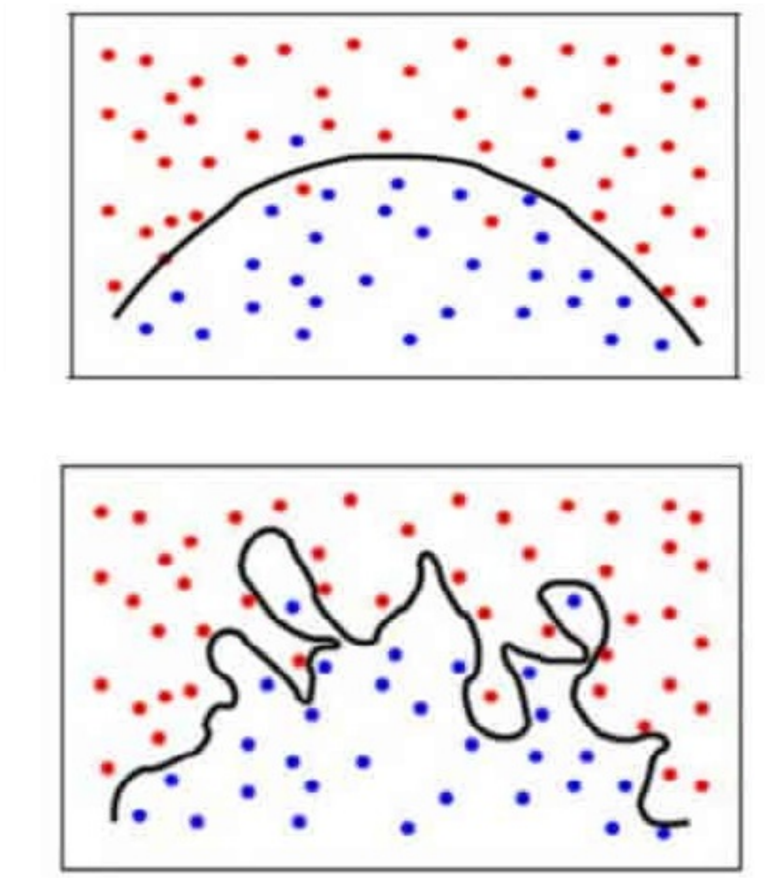


Figure 17. The three possible states of supervised learning (Under fitting, Good Fit & Over fitting)

Unsupervised Learning, as its name reveals is an independent method because there is no supervision of a "teacher". During a model's training phase, using unsupervised learning, the input vectors of similar type are joined to form clusters.
When a new input pattern is applied, then the neural network gives an output response indicating the class to which input pattern belongs. In this method, there is no feedback from the environment like for example what should be the desired output and whether it is correct or not. Therefore, in this type of learning the network itself must discover the patterns, features from the input data and the relation for the input data over the output.

Self Organized Maps (SOM) belong to this category [29]. A SOM is a ANN that has a set of neurons connected to form a topological grid (usually rectangular). When some pattern is presented to an SOM, the neuron with closest weight vector is considered a winner and its weights are adapted to the pattern, as well as the weights of its neighbourhood. In this way an SOM naturally finds data clusters.

Clustering is the most typical example of unsupervised learning strategy. Clustering is the organization of unlabeled data into similarity groups called clusters (Figure 18). A cluster is a collection of data items which are "similar" between them, and "dissimilar" to data items in other clusters.

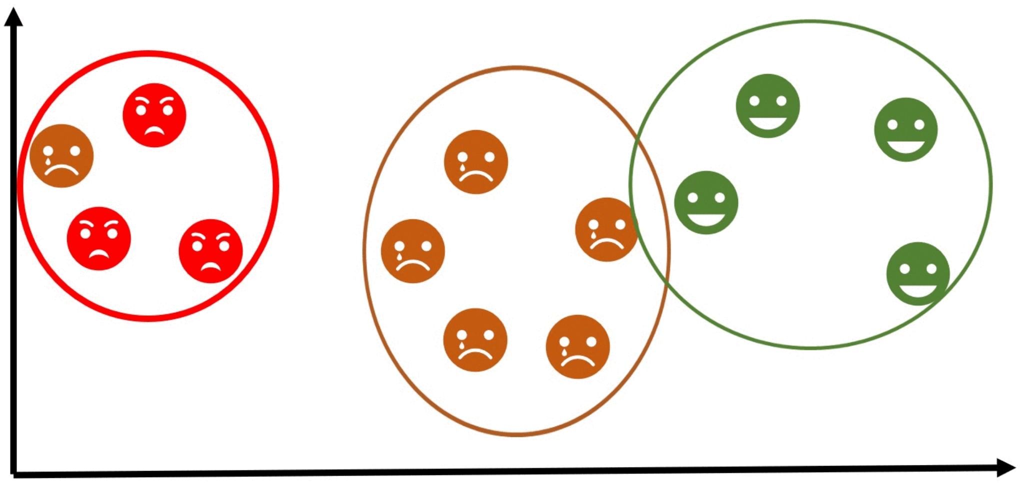


Figure 18. Clustering: The process of grouping similar entities together (Source: https://towardsdatascience.com/clustering-unsupervised-learning-788b215b074b)

Cluster analysis is a main task of data mining, and a common technique for data analysis, used in many fields, like machine learning and image recognition. Detailed analysis about clustering and its algorithms can be found in [29].

3. Preliminary Experiments

In this section, preliminary experiments done in MATLAB, are presented and basics steps of MATLAB's provided toolbox for ANN are introduced.

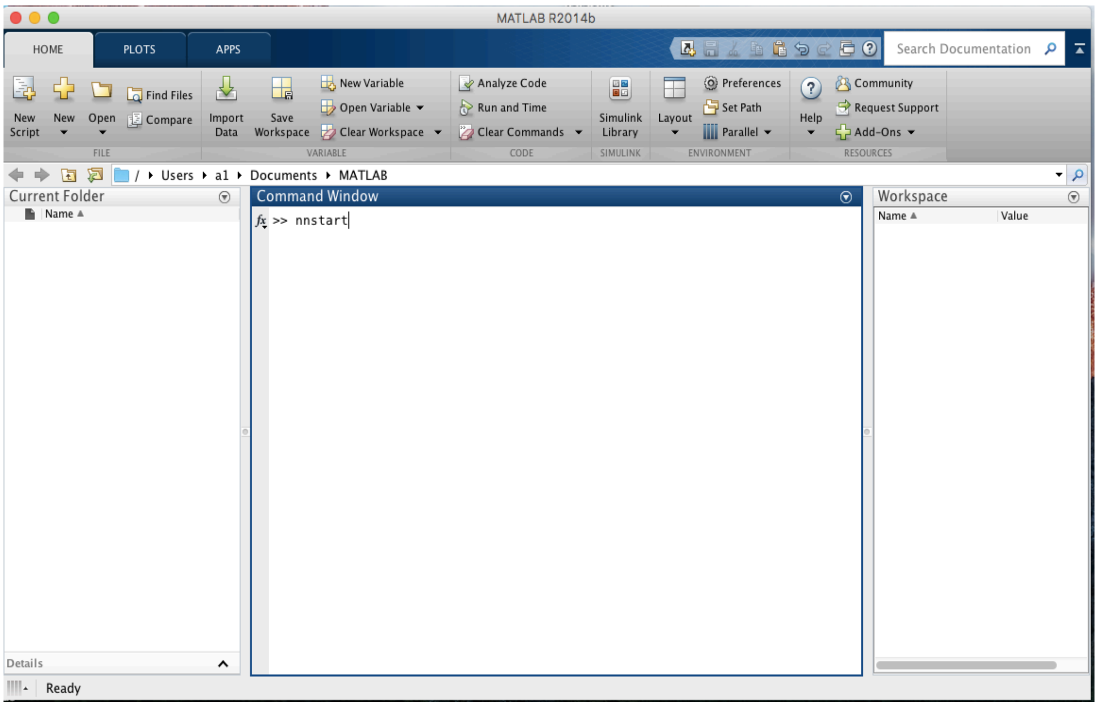
MATLAB's toolbox is ideal for experimenting in building networks. It contains all basic functionalities, graphs (ROC Area, Confusion Matrix) and datasets to play with.

3.1 MATLAB & Neural Networks

Engineering and IT teams are using MATLAB to build today's advanced Big Data Analytic systems ranging from predictive maintenance and telematics to advanced driver assistance systems and sensor analytics. MATLAB is considered to be one of the best tools, because it offers essential capabilities not found in business intelligence systems or other open source languages.

MATLAB provides to its users a full set of statistics and machine learning functionality, plus advanced methods such as nonlinear optimization, system identification, and thousands of prebuilt algorithms for image and video processing, financial modelling, control system design.

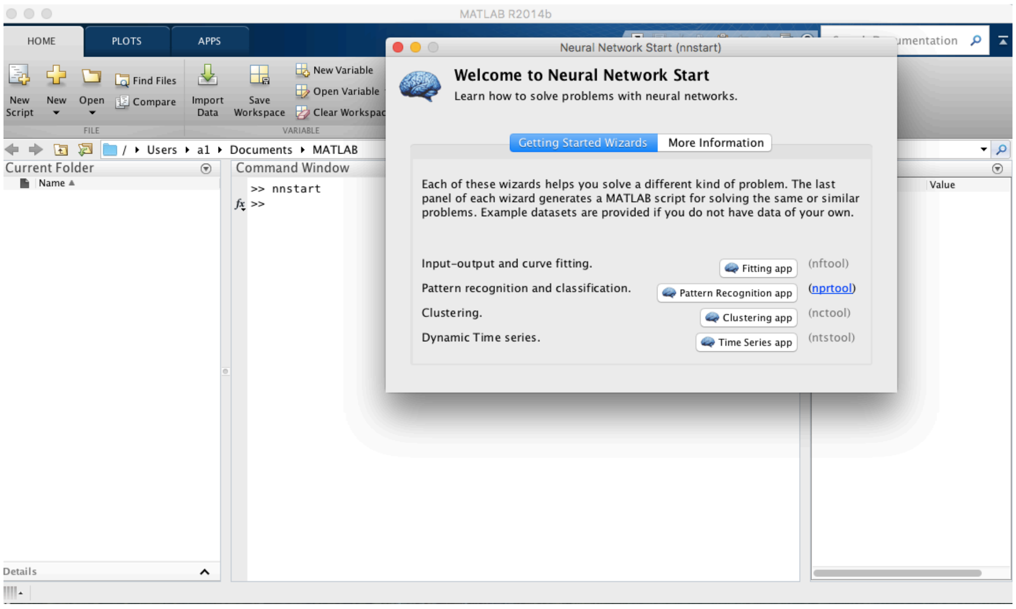
In order to work with neural networks and train them, we use MATLAB's Toolbox for ANN. Once, the software is successfully installed, the toolbox and its functionalities can be launched simply by writing, in MATLAB's environment, 'nnstart' and pressing Enter (Figure 19).



```
>> nnstart
```

Figure 19. Command 'nnstart'

This leads to a pop-up window, 'Neural Network Start', which showcases the available options, depending on what each user wants to apply on the data (Figure 20).



```
>> nnstart
fx >>
```

Welcome to Neural Network Start

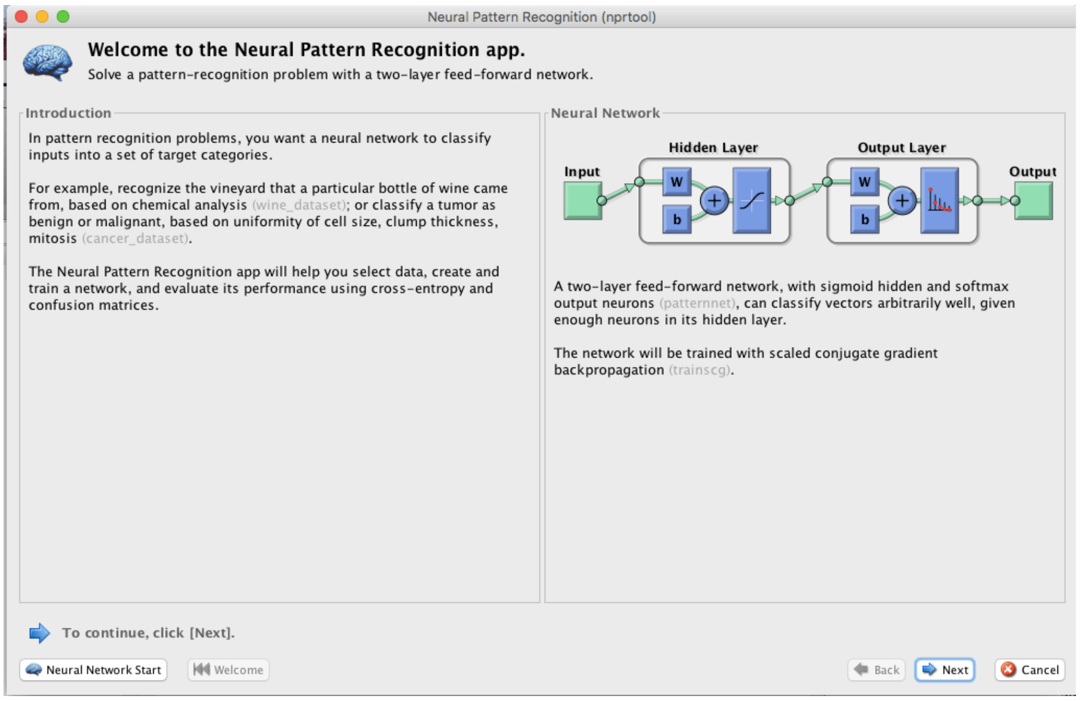
Learn how to solve problems with neural networks.

Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

Input-output and curve fitting.	(nftool)
Pattern recognition and classification.	(nprtool)
Clustering.	(nctool)
Dynamic Time series.	(ntstool)

Figure 20. 'Neural Network Start' pop-up window.

In order to train a ANN, to recognize and classify properly the data given, the 'Pattern recognition and classification' tool (nprtool) is the most suitable choice. It can be launched either by clicking on it or by typing the command 'nprtool' in the command window.
Once, the command is executed the user is being lead to a new window, which explains the use of the selected tool as well as the architecture of the Neural Network used (Figure 21).



Welcome to the Neural Pattern Recognition app.

Solve a pattern-recognition problem with a two-layer feed-forward network.

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories.

For example, recognize the vineyard that a particular bottle of wine came from, based on chemical analysis (wine_dataset); or classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis (cancer_dataset).

The Neural Pattern Recognition app will help you select data, create and train a network, and evaluate its performance using cross-entropy and confusion matrices.

A two-layer feed-forward network, with sigmoid hidden and softmax output neurons (patternnet), can classify vectors arbitrarily well, given enough neurons in its hidden layer.

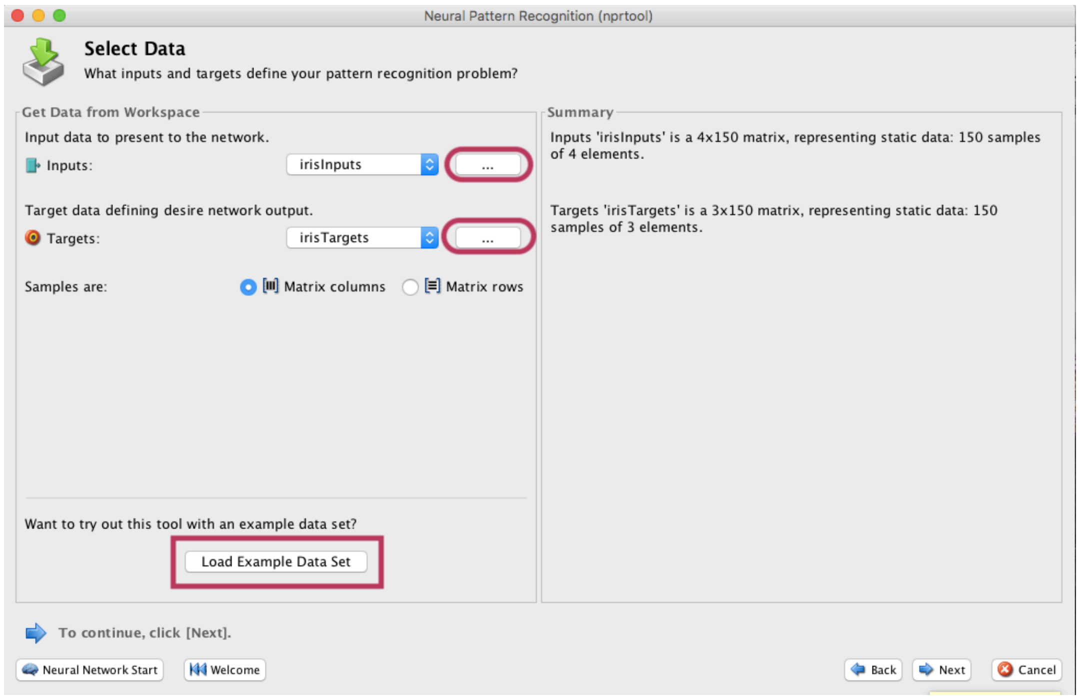
The network will be trained with scaled conjugate gradient backpropagation (trainscg).

To continue, click [Next].

Figure 21. Neural Network Recognition Tab

The neural network provided, is a two-layer feed-forward network, with sigmoid in hidden layer (1.5.2) and a softmax in the output. The network is trained with scaled conjugate gradient backpropagation (SCG), which is quoted in detail in the next paragraphs.

With ‘Next’ button the user can select the dataset to work with. The dataset can either be from the sample that MATLAB provides, or uploaded from user’s files (Figure 22).


Neural Pattern Recognition (nprtool)

Select Data

What inputs and targets define your pattern recognition problem?

Get Data from Workspace

Input data to present to the network.

Inputs: irisInputs

Target data defining desire network output.

Targets: irisTargets

Samples are: [✓ Matrix columns] Matrix rows

Want to try out this tool with an example data set?

Load Example Data Set

Summary

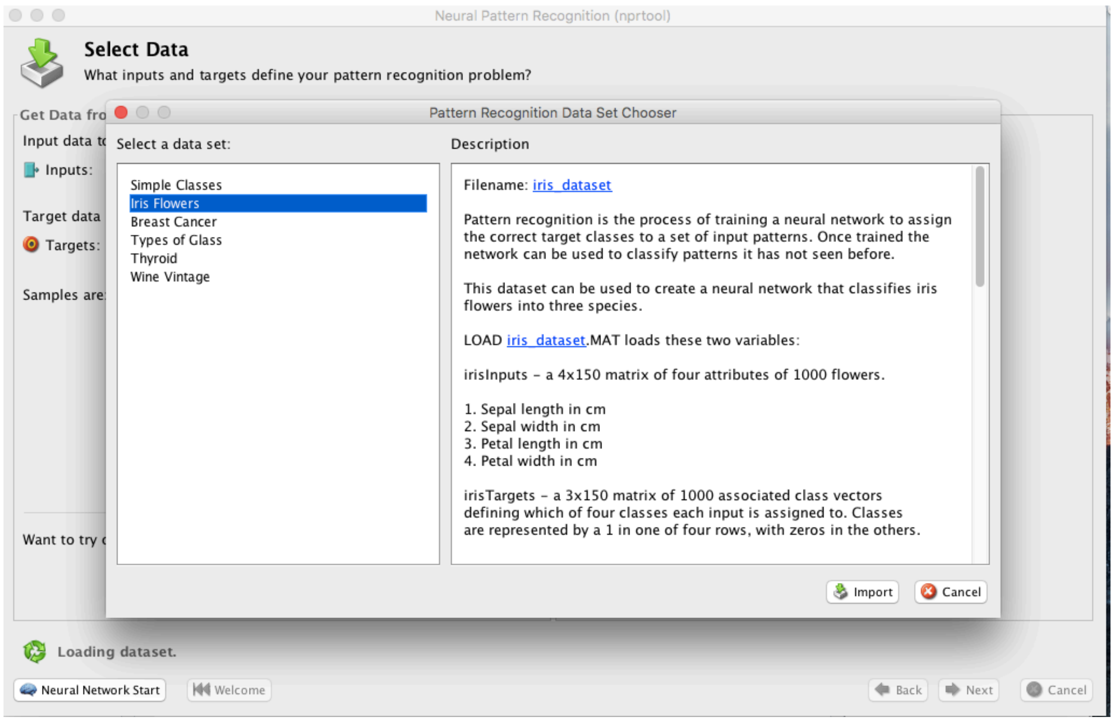
Inputs 'irisInputs' is a 4x150 matrix, representing static data: 150 samples of 4 elements.

Targets 'irisTargets' is a 3x150 matrix, representing static data: 150 samples of 3 elements.

To continue, click [Next].

Figure 22. Select Data Tab

To show a quick example, Iris dataset has been selected for the results seen below. With ‘Load Example Data Set’ button, the datasets provided by MATLAB are being shown. Its dataset comes with a description of the number of classes, attributes etc. as Figure 23 showcases. ‘Import’ button to load the dataset & ‘Next’.


Neural Pattern Recognition (nprtool)

Select Data

What inputs and targets define your pattern recognition problem?

Pattern Recognition Data Set Chooser

Select a data set:

- Simple Classes
- Iris Flowers
- Breast Cancer
- Types of Glass
- Thyroid
- Wine Vintage

Description

Filename: iris_dataset

Pattern recognition is the process of training a neural network to assign the correct target classes to a set of input patterns. Once trained the network can be used to classify patterns it has not seen before.

This dataset can be used to create a neural network that classifies iris flowers into three species.

LOAD iris_dataset.MAT loads these two variables:

irisInputs - a 4x150 matrix of four attributes of 1000 flowers.

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm

irisTargets - a 3x150 matrix of 1000 associated class vectors defining which of four classes each input is assigned to. Classes are represented by a 1 in one of four rows, with zeros in the others.

Loading dataset.

Figure 23. Example Data Sets provided by MATLAB

Iris flower data set is one of the most popular dataset, for clustering and classification problems. The dataset contains a set of 150 records under 4 attributes:

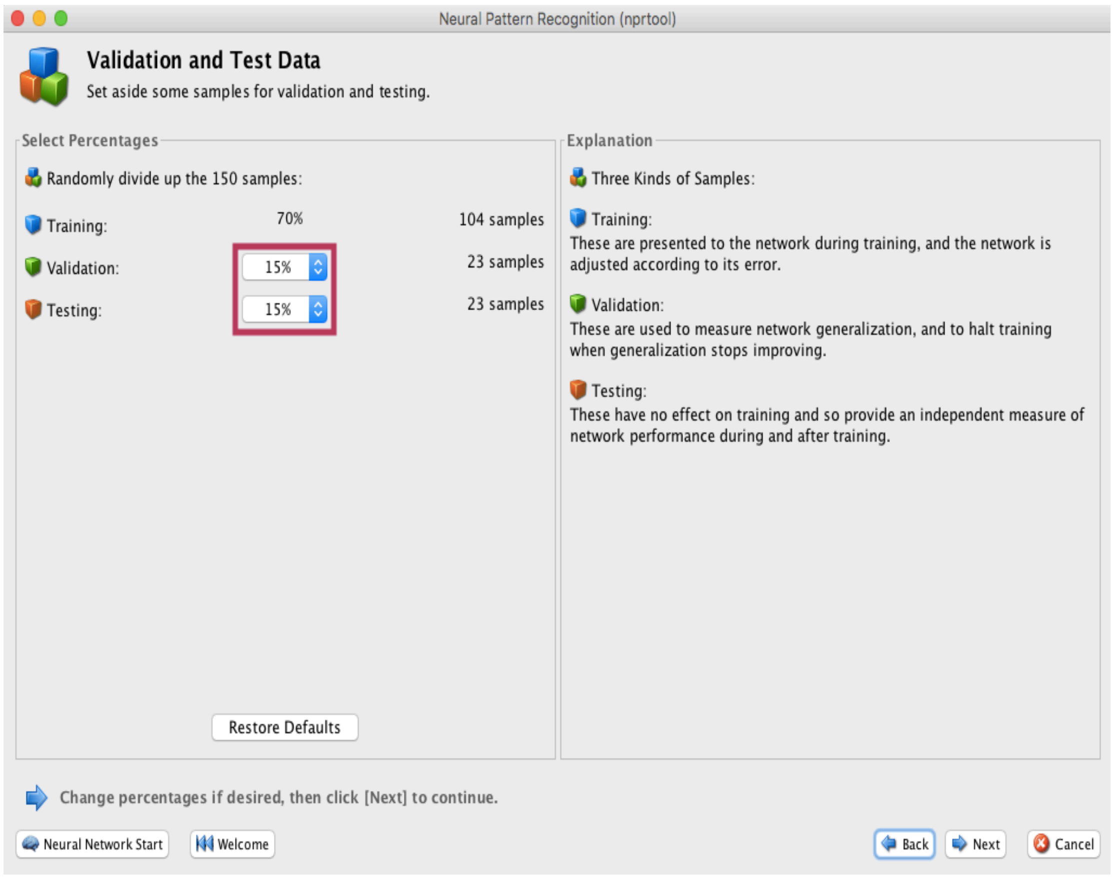
- Petal Length in cm
- Petal Width in cm
- Sepal Length in cm
- Sepal width in cm, which is described as Inputs.

The Targets a 3*150 matrix of class vectors, which showcases the class of the flower, ex. [0,1,0] the flower belongs to 2nd class.

After loading the dataset, its divided randomly in three individual datasets:

- Training Data
- Validation Data
- Testing Data

Validation and Testing data are by default equally divided (15% each), but the user can change it, in order to experiment and increase network's performance (Figure 24).


Neural Pattern Recognition (nprtool)

Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 150 samples:

Training:	70%	104 samples
Validation:	15%	23 samples
Testing:	15%	23 samples

Explanation

Three Kinds of Samples:

Training:
These are presented to the network during training, and the network is adjusted according to its error.

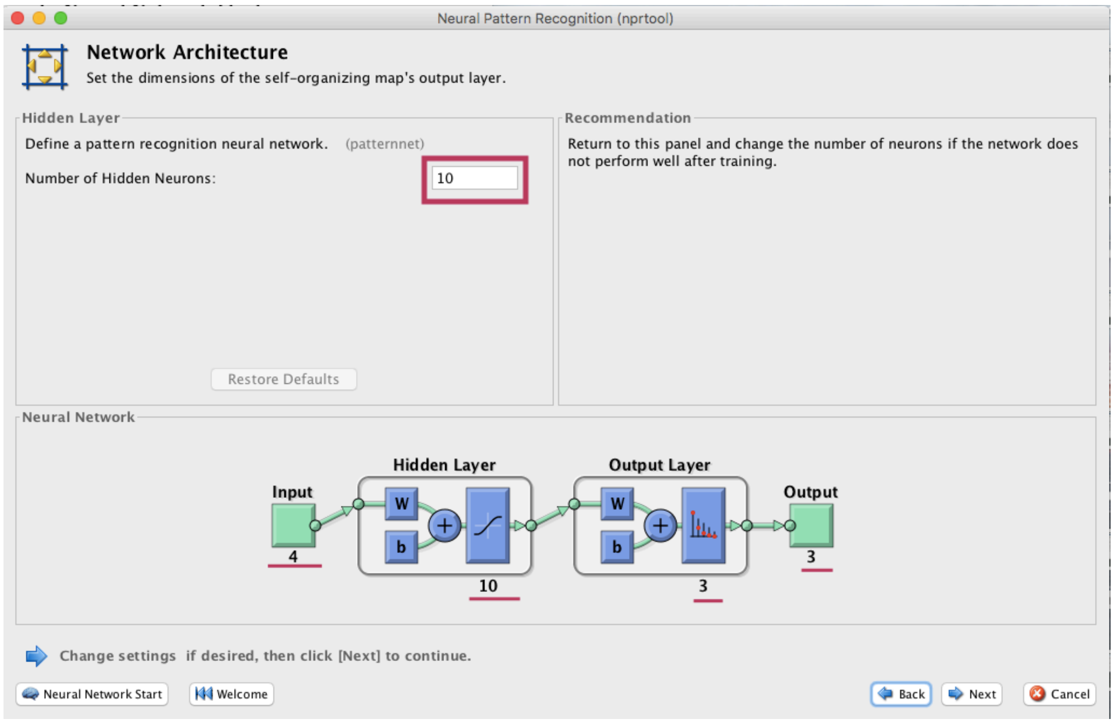
Validation:
These are used to measure network generalization, and to halt training when generalization stops improving.

Testing:
These have no effect on training and so provide an independent measure of network performance during and after training.

Change percentages if desired, then click [Next] to continue.

Figure 24. Validation and Test Data tab

By clicking 'Restore Defaults' the Validation & Testing data percentage are restored to 15% each. By clicking 'Next' button, the user can regulate the number of neurons the hidden layer will have. The default number of hidden neurons is 10 (Figure 25).



Network Architecture

Set the dimensions of the self-organizing map's output layer.

Hidden Layer

Define a pattern recognition neural network. (patternnet)

Number of Hidden Neurons: 10

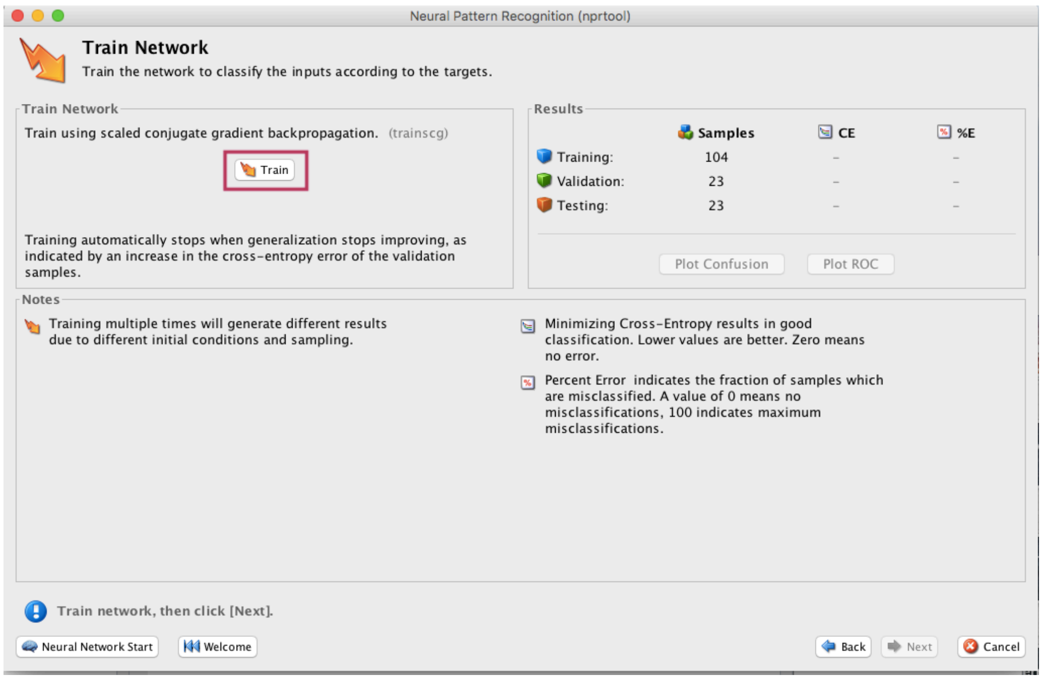
Recommendation

Return to this panel and change the number of neurons if the network does not perform well after training.

Change settings if desired, then click [Next] to continue.

Figure 25. Network Architecture tab

By clicking ‘Next’ the network is created, and it is ready for training (Figure 26). Depending on the results of the first experiment the user, with ‘Back’ button, can change both the allocation between Test & Validation data and the number of hidden neurons in order to retrain the network. These changes often lead to increased network’s performance & accuracy.



Train Network

Train the network to classify the inputs according to the targets.

Train Network

Train using scaled conjugate gradient backpropagation. (trainscg)

Training automatically stops when generalization stops improving, as indicated by an increase in the cross-entropy error of the validation samples.

Results

	Samples	CE	%E
Training:	104	-	-
Validation:	23	-	-
Testing:	23	-	-

Notes

Training multiple times will generate different results due to different initial conditions and sampling.

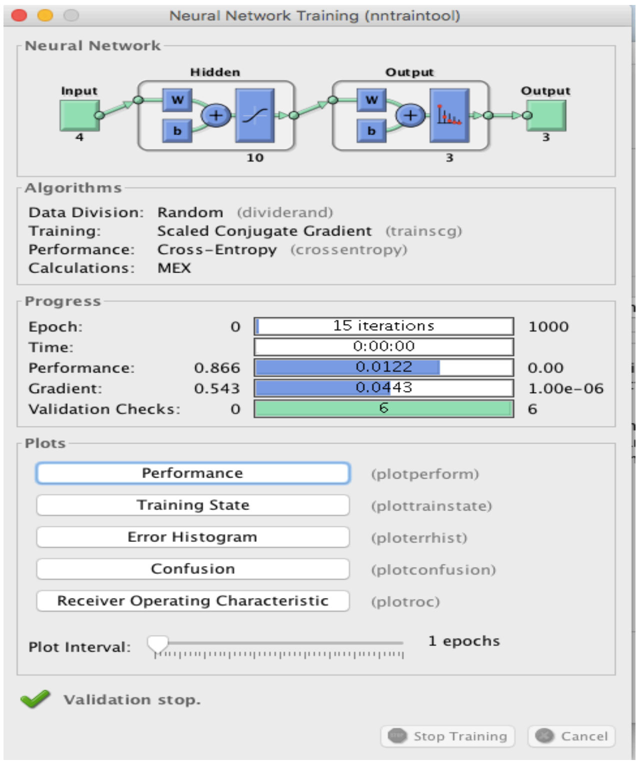
Minimizing Cross-Entropy results in good classification. Lower values are better. Zero means no error.

Percent Error indicates the fraction of samples which are misclassified. A value of 0 means no misclassifications, 100 indicates maximum misclassifications.

Train network, then click [Next].

Figure 26. Train Network tab

After the training is finished a pop-up window, shows a summary of the trained model. Depending on these results the user can decide what actions should be taken to achieve better performance (which factors should be changed, as mentioned above).



Algorithms

Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Cross-Entropy (crossentropy)
Calculations: MEX

Progress

Epoch:	0	15 iterations	1000
Time:		0:00:00	
Performance:	0.866	0.0122	0.00
Gradient:	0.543	0.0443	1.00e-06
Validation Checks:	0	6	6

(plotperform)
(plottrainstate)
(ploterrhist)
(plotconfusion)
(plotroc)

Plot Interval: 1 epochs

Validation stop.

Figure 27. Neural Network's training results

By choosing, each one of the provided plot tabs, a new pop up window shows up with the corresponding plot:

Performance (plotperfom)

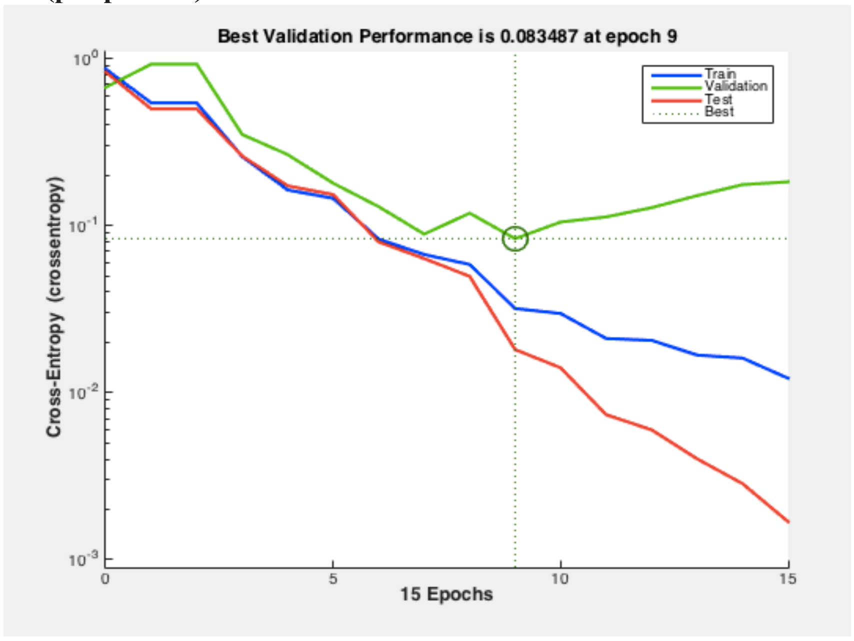


Figure 28. Performance plot

Training State (plottrainstate)

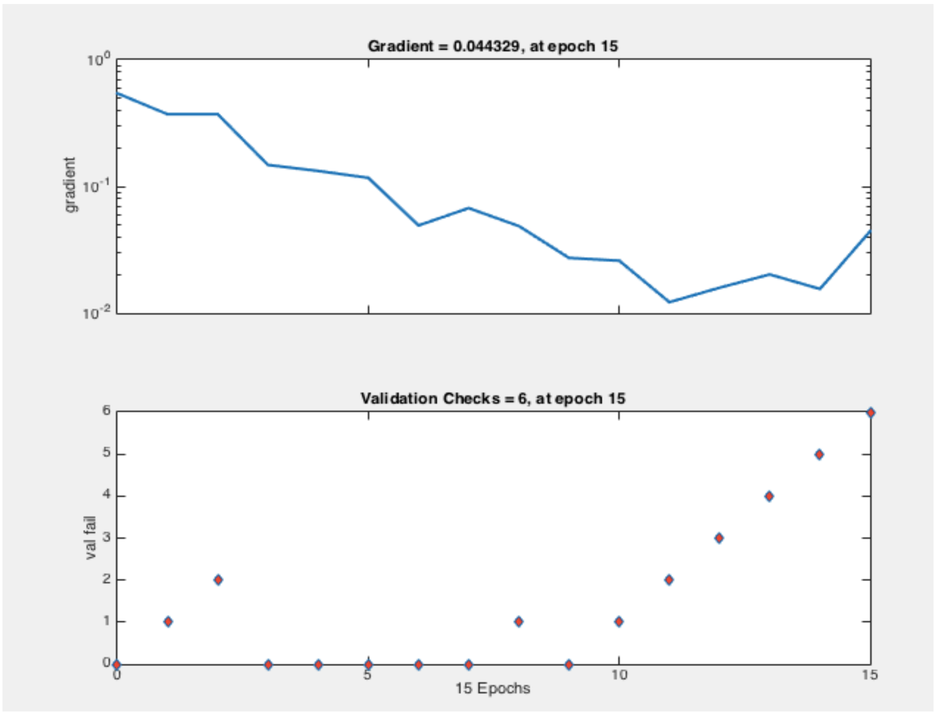


Figure 29. Training State plot

The plot shows variation in gradient coefficient with respect to number of epochs. The final value of gradient coefficient at epoch number 15 is 0.044329 (Figure 29).

Error Histogram (plotterhist)

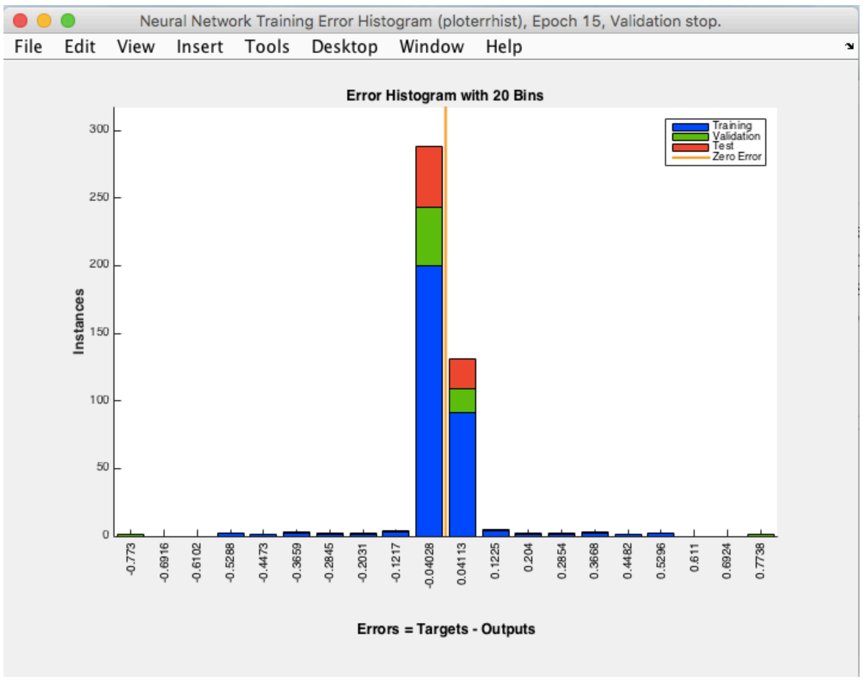
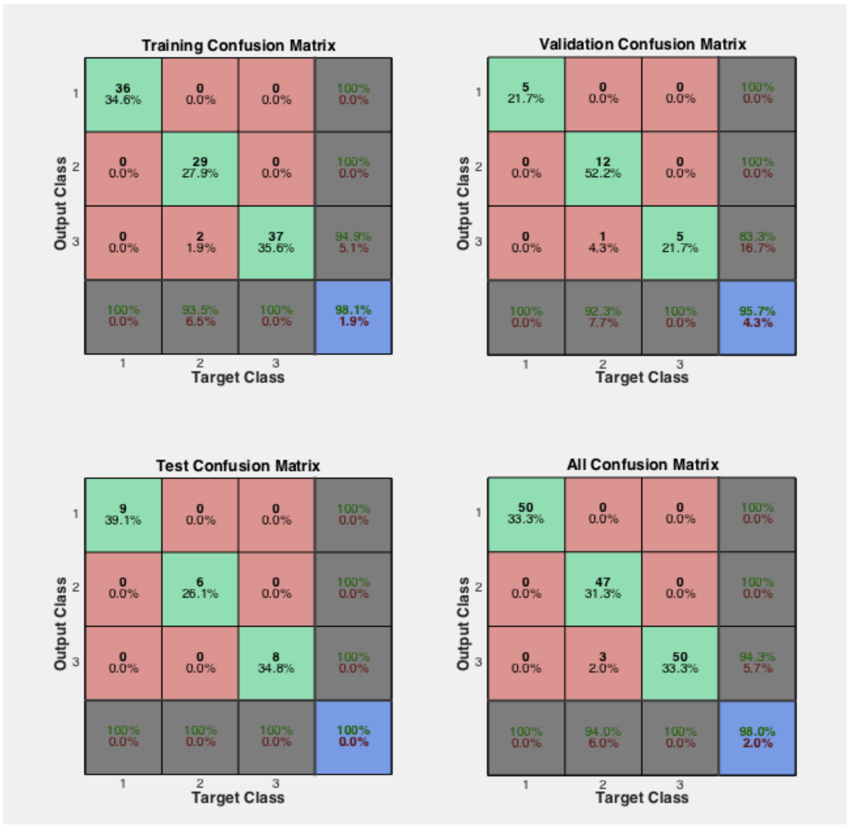


Figure 30. Error Histogram plot

x-axis: Error value (Targets – Outputs)
y-axis: Number of Instances

Confusion (plotconfusion)



Training Confusion Matrix

Output Class \ Target Class	1	2	3	
1	36 34.6%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	29 27.9%	0 0.0%	100% 0.0%
3	0 0.0%	2 1.9%	37 35.6%	94.9% 5.1%
	100% 0.0%	93.5% 6.5%	100% 0.0%	98.1% 1.9%

Validation Confusion Matrix

Output Class \ Target Class	1	2	3	
1	5 21.7%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	12 52.2%	0 0.0%	100% 0.0%
3	0 0.0%	1 4.3%	5 21.7%	83.3% 16.7%
	100% 0.0%	92.3% 7.7%	100% 0.0%	95.7% 4.3%

Test Confusion Matrix

Output Class \ Target Class	1	2	3	
1	9 39.1%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	6 26.1%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	8 34.8%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%

All Confusion Matrix

Output Class \ Target Class	1	2	3	
1	50 33.3%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	47 31.3%	0 0.0%	100% 0.0%
3	0 0.0%	3 2.0%	50 33.3%	94.3% 5.7%
	100% 0.0%	94.0% 6.0%	100% 0.0%	98.0% 2.0%

Figure 31. Confusion Matrix plot

The Confusion Matrix is a N x N table which showcases the number of false positives, false negatives, true positives, and true negatives. In this case, the "plotconfusion" command presents 4 confusion matrices, one for each set (training, test & validation) and one for the overall set.

False Negatives (FN): These are cases in which the algorithm has predicted 'No', and the actual result is 'Yes'.

False Positives (FP): These are cases in which the algorithm has predicted 'Yes', and the actual result is 'No'.

True Positives (TP): These are cases in which the algorithm has predicted 'Yes', and the actual result is also 'Yes'.

True Negatives (TN): These are cases in which the algorithm has predicted 'No', and the actual result is also 'No'.

Receiver Operating Characteristic (plotroc)

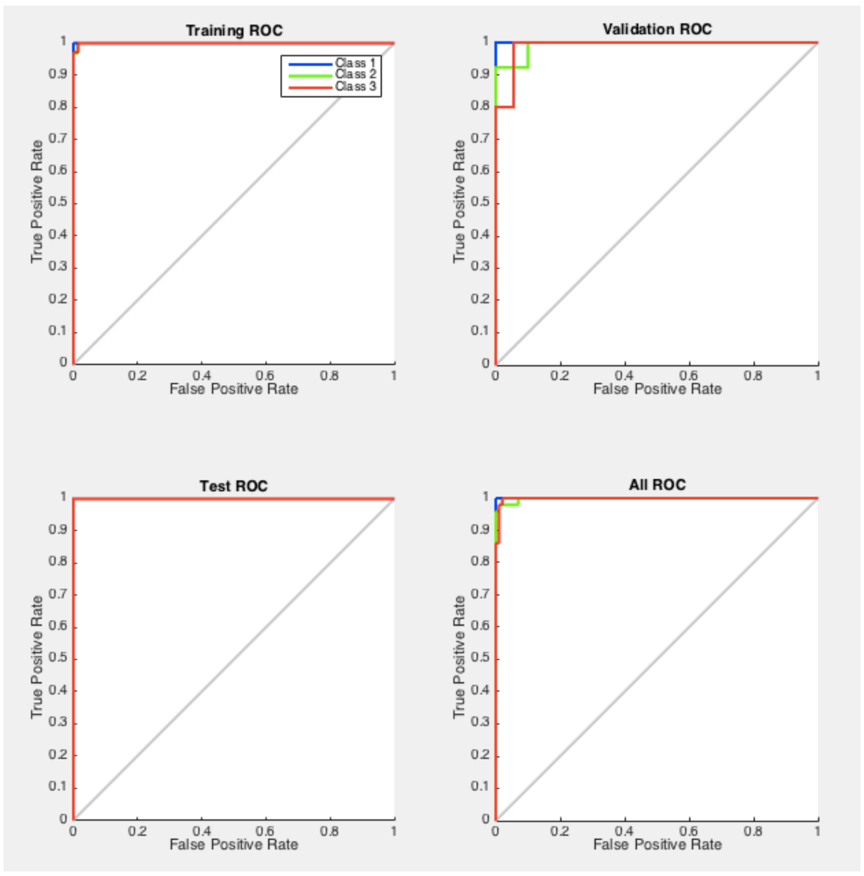


Figure 32. ROC plot

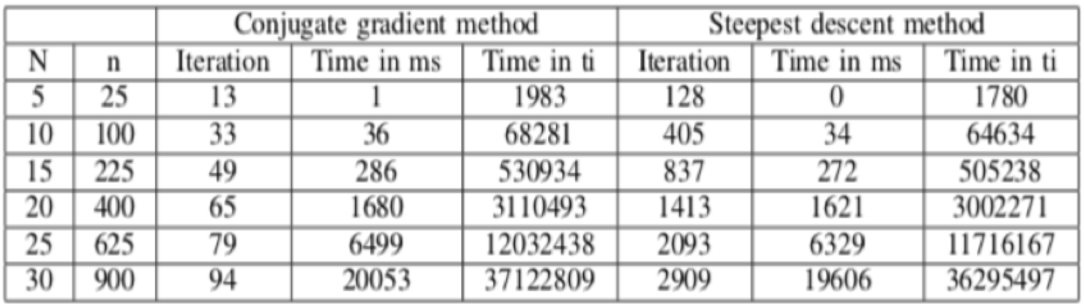
Another metric, which is commonly used in order to observe how well the neural network has fit data is the receiver operating characteristic plot. This shows how the false positive and true positive rates relate for all the possible values of threshold varied from 0 to 1.

The further left and up the line is, the fewer false positives need to be accepted in order to get a high true positive rate. The best classifiers will have a line going from the bottom left corner, to the top left corner, to the top right corner, or close to that (Figure 32).

3.1.1 Scaled Conjugate Gradient Method

Matlab’s library for neural networks uses the Scaled Conjugate Gradient method (SCG). This method is a variety of classic Gradient Descent Methods. The algorithm is based upon a class of optimization techniques well known in numerical analysis as the Conjugate Gradient Methods. SCG uses second order information from the neural network but requires only O(N) memory usage, where N is the number of weights in the network. SCG does not contain any user-dependent parameters whose values are crucial for the success of SCG. By using a step size scaling mechanism, SCG avoids a time-consuming line search per learning iteration, which makes the algorithm faster than other second-order algorithms (see [13] for deeper mathematical analysis).

The supremacy of the Conjugate Gradient Methods over the Steepest descent ones has been studied by Osadcha & Marszaek in [13] and is depicted in Figure 33 below (also included in the paper aforementioned):



		Conjugate gradient method			Steepest descent method		
N	n	Iteration	Time in ms	Time in ti	Iteration	Time in ms	Time in ti
5	25	13	1	1983	128	0	1780
10	100	33	36	68281	405	34	64634
15	225	49	286	530934	837	272	505238
20	400	65	1680	3110493	1413	1621	3002271
25	625	79	6499	12032438	2093	6329	11716167
30	900	94	20053	37122809	2909	19606	36295497

Figure 33. Results after comparison between Conjugate Gradient and Steepest Descent methods. (Source: http://ceur-ws.org/Vol-1853/p01.pdf)

4. Machine Learning and Applications

4.1 Machine Learning and Healthcare

As already mentioned, applications of Machine Learning can become more than useful in any malformation early detection. Doctors can benefit from advanced analytics, provided by these applications, having a comprehensive picture about patients' health care and treatment. Healthcare informatics combined with Machine Learning algorithms can upgrade level of quality and efficiency of healthcare industry [18]. Integrating these applications in smart devices (like smartphones), can reduce the cost of specialized medical examinations, making them affordable to a wide range of patients. In this way, prevention and early detection of cancer or other diseases can be accessible by everybody. Considering the importance of developing healthcare applications, the neural networks designed to predict breast cancer.

The Neural Networks are implemented with Python's library Scikit-learn. Scikit-learn has a class called MLPClassifier, which has been used to create and train the networks presented in the next sections. MLP uses backpropagation method to train the network. The hyper parameters that were changed have been recorded in result tables (Tables 1-3 & Tables 5-8). For the Confusion Matrices (Figure 37 & 38) library Matplotlib has been used.

4.1.1 Risk Factors and Breast Cancer

Breast Cancer is the deadliest cancer among women. It's the most commonly occurring cancer in women and the second most common cancer overall. In 2018, there were over 2 million new cases, based on the statistics publish by World Cancer Research Fund, WCRF [1].

By the risk factor, doctors define anything that increases the risk of developing cancer. Risk factors for breast cancer are grouped into two basic categories: risk factors you can control and risk factors you can't control [22].

Among risk factor you can control is exercise, alcohol, smoking and diet. Studies have shown that the lack of exercise and smoking are associated with a small increase in breast cancer risk. On the other hand, age, race and family history of cancer are included in risk factor you cannot control (Figure 34). Studies have shown that white women are slightly more likely to develop breast cancer than African American women [23].

[1] (https://www.wcrf.org/dietandcancer/cancer-trends/breast-cancer-statistics)

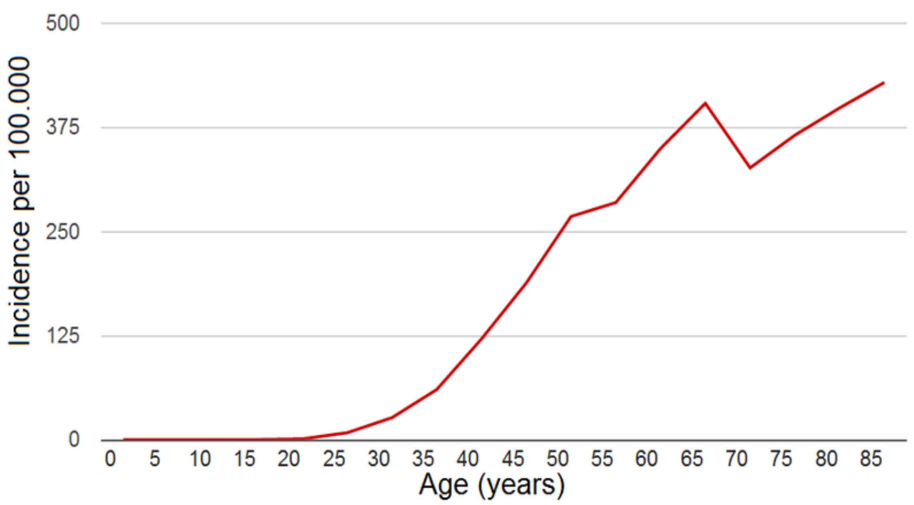


Figure 34. Number of Breast Cancer Incidents in relation to risk factor "Age" (Source: Wikipedia)

4.1.2 Breast Cancer Diagnosis

Early detection of breast cancer can increase the survival rate. The 5-year relative survival rate for women with stage 0 or stage I is close to 100%. The corresponding rate for women with stage II, drops to 93%, with information provided in American Cancer Society's website [1].

Diagnosis can be done with a variety of medical tests and with the supervision of an expert [26]. More specifically:

- **Diagnostic Mammogram**: It is a more detailed X-ray of the breast.
- **Breast ultrasound**: Gives detailed pictures of areas inside the breast, produced by sound waves.
- **Magnetic Resonance Imaging (MRI)**: The MRI scan will make detailed pictures of areas inside the breast.
- **Biopsy:** It's a test that tissue or fluid is removed from the breast to be looked at under a microscope.

STAGE OF BREAST CANCER	5 YEAR SURVIVAL RATE
STAGE 0	100 %
STAGE I	100 %
STAGE II	Around 93 %
STAGE III	Around 72 %
STAGE IV	28 %

Table above showcases the survival rate depending on cancer stage [2].

4.1.3 Dataset Description

The dataset used to feed the neural network is "Wisconsin Diagnostic Breast Cancer (WDBC)" and can be found in The UC Irvine Machine Learning Repository website [3]. The dataset consists of 569 instances and 32 attributes.

Attribute Information:

1) ID Number
2) Diagnosis (B = Benign, M = Malignant)
3) The other 30 columns provide information about ten-real valued features for each cell nucleus:
    a. radius (mean of distances from center to points on the perimeter)
    b. texture (standard deviation of gray-scale values)
    c. perimeter
    d. area
    e. smoothness (local variation in radius length)
    f. compactness (perimeter2 / area – 1.0)
    g. concavity (severity of concave portions of the contour)
    h. concave points (number of concave portions of the contour)
    i. symmetry
    j. fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
Separating plane described above was obtained using Multisurface Method-Tree (MSM-T), a classification method constructing decision tree using linear programming [24]. The actual linear program used to obtain the separating plane in the 3-dimensional space is the one described by Bennett and Mangasarian in [25].

The dataset was created by:
Wolberg, General Surgery Dept., University of Wisconsin, Clinical Sciences Center.
Street, Computer Sciences Dept., University of Wisconsin.
Mangasarian, Computer Sciences Dept., University of Wisconsin.

[1] https://www.cancer.org/cancer/breast-cancer/understanding-a-breast-cancer-diagnosis/breast-cancer-survival-rates.html

[2] Based on data & statistics provided by: https://seer.cancer.gov/data/

[3] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

4.1.4 Experimental Evaluation

The model was created and trained with the back-propagation method. The tables below present the results and the parameters used for each experiment:

Hidden layer size	100	100	100	100	100	100	100	100	100
Activation	Logistic	Logistic	Logistic	Logistic	Logistic	ReLu	ReLu	ReLu	ReLu
Solver	Adam	Adam	Adam	Sgd	Sgd	Sgd	Sgd	Adam	Adam
Alpha	0.001	0.001	0.002	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	2500	2500	2000	2500	2000	2500	2000	2500
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**93.8%**	**94.69%**	**97.35%**	**90.26%**	**85.84%**	**85.84%**	**76.99%**	**95.57%**	**97.34%**

Table 1. Accuracy with experiments based on architecture with 1 hidden layer and 100 hidden neurons.

Hidden layer size	35	35	35	35	35	35	35	35
Activation	ReLu	ReLu	Logistic	Logistic	ReLu	ReLu	Logistic	Logistic
Solver	Adam	Adam	Adam	Adam	Sgd	Sgd	Sgd	Sgd
Alpha	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	12000	2000	12000	2000	12000	2000	12000
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**96.46%**	**92.92%**	**95.57%**	**97.34%**	**76.99%**	**76.80%**	**88.49%**	**92.03%**

Table 2. Accuracy with experiments based on architecture with 1 hidden layer and 35 hidden neurons.

Hidden layer size	10	10	10	10	10	10	10	10
Activation	ReLu	ReLu	Logistic	Logistic	ReLu	ReLu	Logistic	Logistic
Solver	Adam	Adam	Adam	Adam	Sgd	Sgd	Sgd	Sgd
Alpha	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	4500	2000	4500	2000	4500	2000	4500
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**94.69%**	**93.81%**	**95.57%**	**97.35%**	**89.38%**	**79.64%**	**91.15%**	**92.92%**

Table 3. Accuracy with experiments based on architecture with 1 hidden layer and 10 hidden neurons.

The train and the test set were not randomly split up, in order to apply parameter changes on the exact same dataset and compare the results. The experiments processed on different MLP models, changing parameters like numbers of hidden neurons and activation functions. The activation functions used in the experimental process were sigmoid and ReLu.

Based on the above results, the conclusion drawn is that ‘sgd’ (Stochastic Gradient Descent) solver gives bigger accuracy when having as activation function “Logistic” (Sigmoid), that the ReLu. The ANN having 35 hidden neurons achieved 97.35% accuracy after 12000 epochs comparatively with the one having 100 hidden neurons which reached the same accuracy at 2500 epochs.

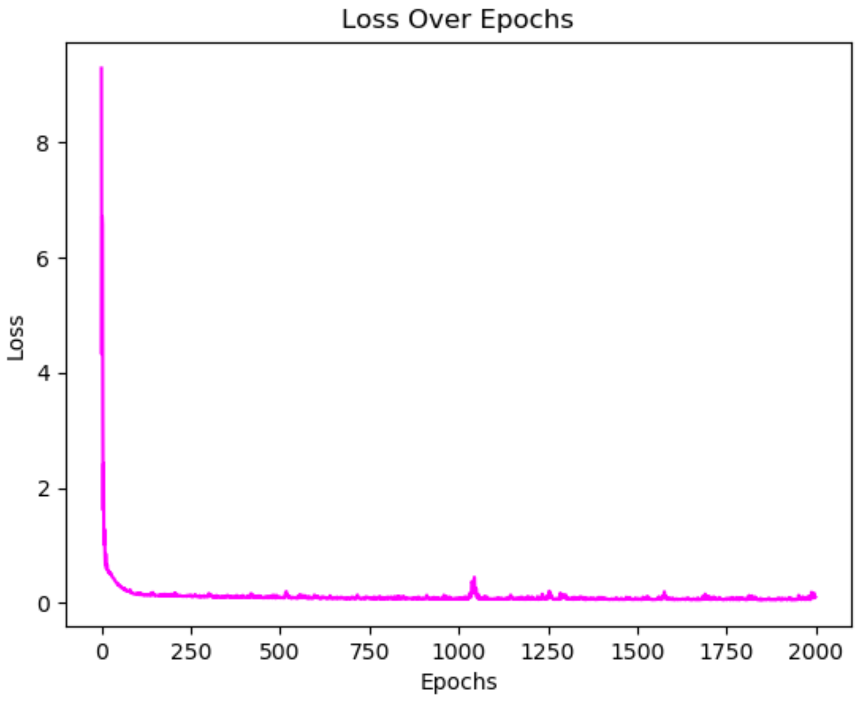


Figure 35. Graph of Loss Over Epochs

The graph in figure 36 depicts the results of accuracy between an ANN and an SVM. The ANN used for plotting, has 100 hidden neurons & ‘relu’ as activation function. The SVM used with linear kernel.

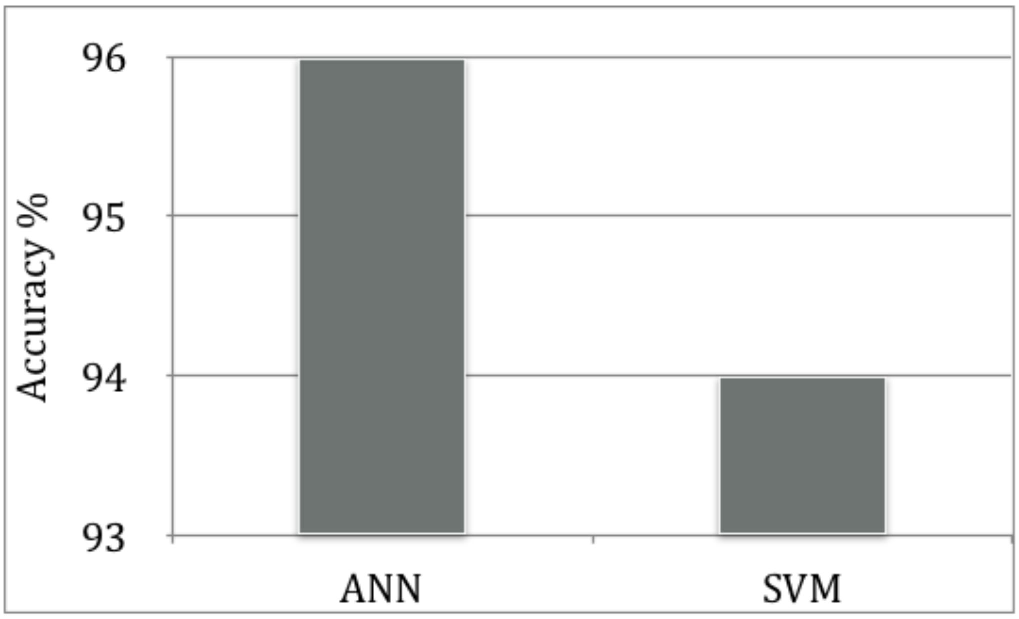


Figure 36. Accuracy Comparison ANN vs SVM

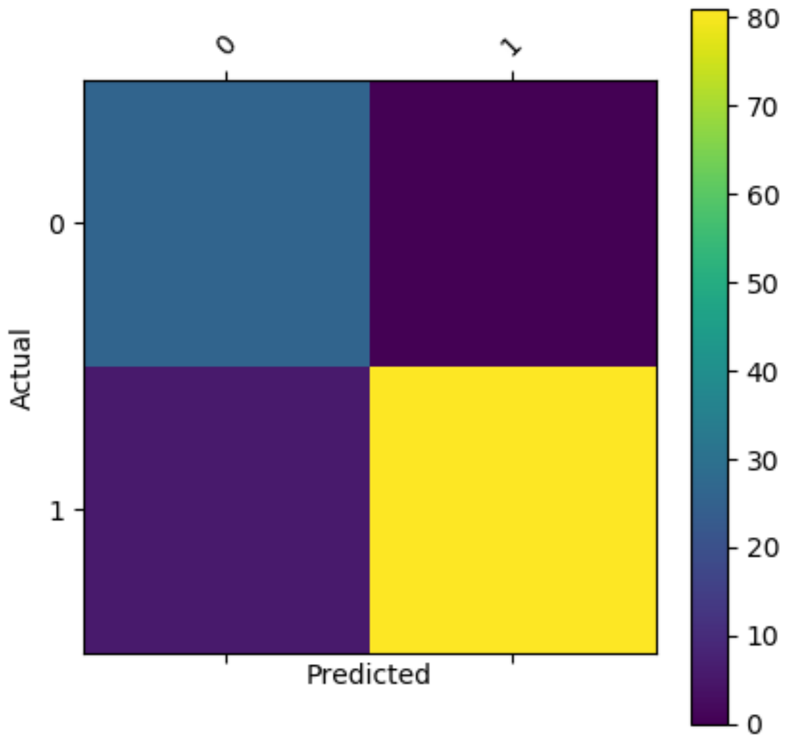


Figure 37. Confusion Matrix of ANN (Graph)

n = 113	**Predicted 0**	**Predicted 1**
Actual 0	26	0
Actual 1	6	81

Table 4. Confusion Matrix of ANN.

The confusion matrix is used to visualize the performance of an algorithm (Figure 37). In this confusion matrix, depicted above (Table 4), the ANN predicted correct 107 instances out of 113. All correct predictions are located in the diagonal of this matrix.

Sensitivity refers to the model's ability to correctly detect ill patients who really have the condition. In the example of a medical test used to identify a disease, the sensitivity of the test is the proportion of people who test positive for the disease among those who have the disease. Specificity, on the other hand, relates to the model's ability to correctly classify non-malignant instances as negative to cancer.

4.2 Machine Learning and Optical Character Recognition

Machine learning is the process of recognizing patterns by using algorithms. One of the important aspects of machine learning is its application potential such as speech recognition, speaker identification, multimedia document recognition (MDR), automatic medical diagnosis.

Handwritten digits recognition is a very intricate field of research in Optical Character Recognition (OCR), as each personal possess unique handwriting style [26]. Neural Networks have been used to recognize and predict the handwritten digits and are recommended as the best approach so far. In [27] Al-Mansoori achieved an overall accuracy of 99.32 % using an MLP Neural Network to recognize digits from 0 to 9.

4.2.1 Dataset Description

The dataset used to feed the neural network is "Optical Recognition of Handwritten Digits" and can be found in The UC Irvine Machine Learning Repository website [1]. The dataset consists of 1767 instances and 64 attributes. Preprocessing programs available by MNIST were used to extract normalized bitmaps of handwritten digits from a preprinted form. 32x32 bitmaps are divided into non overlapping blocks of 4x4. This generates an input matrix of 8x8 where each element is an integer in the range 0..16.

- Number of attributes 64 + 1 class attribute
- Input attributes are integers in the range 0…16.
- The last attribute define the class code is 0…9.

[1] https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits

4.2.2 Experimental Evaluation

The model was created and trained with the back-propagation method. The tables below present the results and the parameters used for each experiment.

Hidden layer size	80	80	80	80	80	80	80	80
Activation	Logistic	Logistic	Logistic	Logistic	ReLu	ReLu	ReLu	ReLu
Solver	Adam	Adam	Sgd	Sgd	Sgd	Sgd	Adam	Adam
Alpha	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	1500	2000	1500	2000	1500	2000	1500
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**92.1%**	**92.2%**	**92.2%**	**91.4%**	**93.4%**	**92.8%**	**93.03%**	**93.9%**

Table 5. Accuracy with experiments based on architecture with 1 hidden layer and 80 hidden neurons.

Hidden layer size	35	35	35	35	35	35	35	35
Activation	ReLu	ReLu	Logistic	Logistic	ReLu	ReLu	Logistic	Logistic
Solver	Adam	Adam	Adam	Adam	Sgd	Sgd	Sgd	Sgd
Alpha	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	1500	2000	1500	2000	1500	2000	1500
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**90.5%**	**91.64%**	**92.2%**	**93%**	**90.8%**	**89.7%**	**91.4%**	**90.8%**

Table 6. Accuracy with experiments based on architecture with 1 hidden layer and 35 hidden neurons.

Hidden layer size	21	21	21	21	21	21	21	21
Activation	ReLu	ReLu	Logistic	Logistic	ReLu	ReLu	Logistic	Logistic
Solver	Adam	Adam	Adam	Adam	Sgd	Sgd	Sgd	Sgd
Alpha	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	1500	2000	1500	2000	1500	2000	1500
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**91.8%**	**93.6%**	**92.2%**	**90.57%**	**93.31%**	**89.69%**	**90.5%**	**90.3%**

Table 7. Accuracy with experiments based on architecture with 1 hidden layer and 21 hidden neurons.

Hidden layer size	15	15	15	15	15	15	15	15
Activation	Logistic	Logistic	Logistic	Logistic	ReLu	ReLu	ReLu	ReLu
Solver	Adam	Adam	Sgd	Sgd	Sgd	Sgd	Adam	Adam
Alpha	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Max_iter	2000	1500	2000	1500	2000	1500	2000	1500
Train	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
Test	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Accuracy	**89.1%**	**89.4%**	**90.3%**	**89.13%**	**90.5%**	**88.7%**	**91.4%**	**90.2%**

Table 8. Accuracy with experiments based on architecture with 1 hidden layer and 15 hidden neurons.

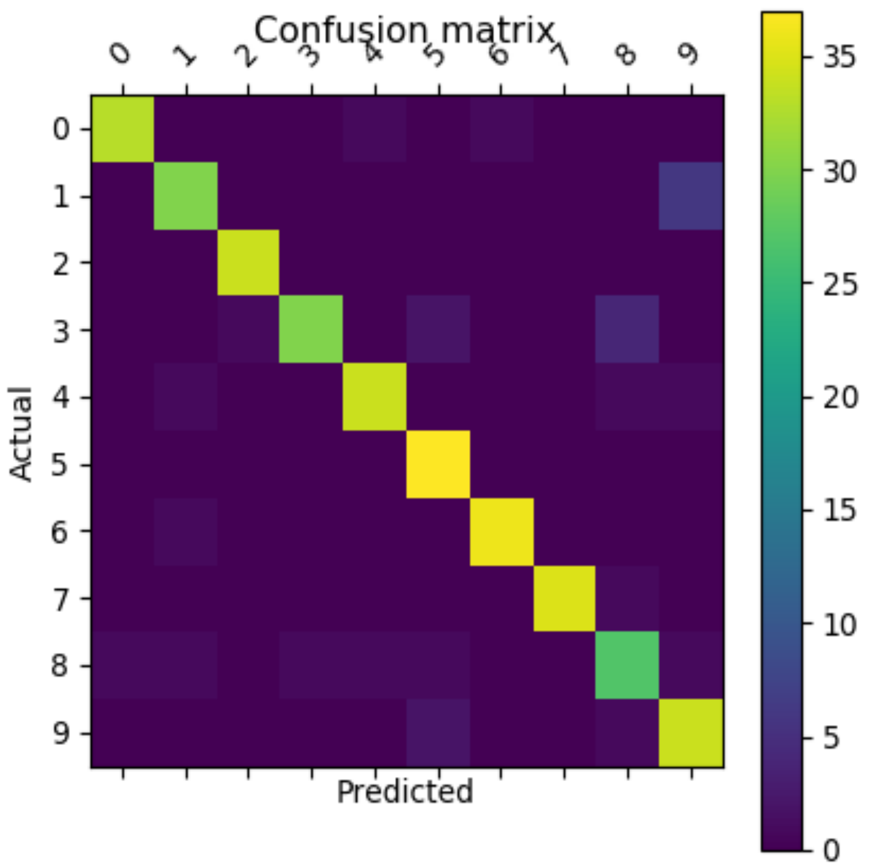


Figure 38. Confusion Matrix of ANN (Graph)

n=359	Pred.0	Pred. 1	Pred. 2	Pred. 3	Pred. 4	Pred. 5	Pred. 6	Pred. 7	Pred.8	Pred. 9
Actual 0	**33**	0	0	0	1	0	1	0	0	0
Actual 1	0	**30**	0	0	0	0	0	0	0	6
Actual 2	0	0	**34**	0	0	0	0	0	0	0
Actual 3	0	0	1	**30**	0	2	0	0	4	0
Actual 4	0	1	0	0	**34**	0	0	0	1	1
Actual 5	0	0	0	0	0	**37**	0	0	0	0
Actual 6	0	1	0	0	0	0	**36**	0	0	0
Actual 7	0	0	0	0	0	0	0	**35**	1	0
Actual 8	1	1	0	1	1	1	0	0	**27**	1
Actual 9	0	0	0	0	0	2	0	0	1	**34**

Table 9. Multiclass Confusion Matrix of ANN.

From the results depicted in the tables above, it seems that the number of hidden neurons can induce the network's accuracy. By comparing the accuracy results between the networks with 21 and 15 hidden neurons, the diminution of accuracy rates in the second one is remarkable.

The confusion matrix is used to visualize the performance of an algorithm (Figure 38). As depicted in table 8 the ANN predicted correct 330 (sum the diagonal elements of the matrix) instances out of 359. The network seems to have problem to distinguish '1', which is confused with '9' and '3' often predicted as '8', an incident which regularly takes place in real life as well.

5. Conclusions

Artificial neural networks are efficiently applied in classification tasks, in this dissertation, in medical diagnosis and optical character recognition. In this work, we examined the performance of two different types of Artificial Neural Networks each one on different purpose. Both models have reached a noteworthy rate of accuracy which can be compared to the one produced by human experts.

The general conclusions drawn from the above experiments are the following:
a. The model's performance can be heavily effected by parameters like the number of hidden neurons, epochs and the combination of solvers and the activation function. Thus, researchers must cope with the cumbersome task of fine tuning these hyper parameters in order to achieve the highest possible scores.

b. No matter how carefully the aforementioned parameters are chosen, the resulting model's performance is additionally affected by the distribution of classes inside the training dataset. For example, in the case of cancer diagnosis, since cancer is a rare case, training datasets are biased in favour of the benign class. The lack of malignant cases in public medical datasets is decisive for the model's learning process and subsequent performance. This fact explains the low sensitivity rates.

The results from the experiments conducted above, show that the proposed diagnosis neural network could, with further training and fine tuning, be useful for identifying possible infected persons. Undoubtedly, ANNs are one of the most effective AI tools, in medical diagnosis and treatment and suggest that there is an expanding role for machine learning in the future of medicine. Nevertheless, medical diagnosis applications must be developed and applied with care as well as the supervision of an expert.

Future work could also include a more careful evaluation of the network design, where different combination of parameters and pre-processing of the data is used. It could also be very interesting if we could see how the weights of neurons are shaped during the experiments. Additionally, it could be very intriguing to see how the accuracy of the network is formed when it's trained with a larger dataset.

REFERENCES

1. W. McCulloch, W. Pitts, 1943, “A logical calculus of the ideas immanent in nervous activity”.
2. Rumelhart, David E., et al. “Learning Internal Representations by Error Propagation.” Neurocomputing: Foundations of Research, 1988, pp. 673–695.
3. Atiya, Amir F. “Bankruptcy Prediction for Credit Risk Using Neural Networks: A Survey and New Results.” IEEE Transactions on Neural Networks, vol. 12, no. 4, 2001, pp. 929–935.
4. Esteva, Andre, et al. “Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks.” Nature, vol. 542, no. 7639, 2017, pp. 115–118.
5. Rogers, H. W. et al. Incidence estimate of nonmelanoma skin cancer (keratinocyte carcinomas) in the US population, 2012. JAMA Dermatology 151.10, 1081–1086 (2015).
6. Stern, R. S. Prevalence of a history of skin cancer in 2007: results of an incidence-based model. Arch. Dermatol. 146, 279–282 (2010).
7. Snow, Peter B., et al. “Artificial Neural Networks in the Diagnosis and Prognosis of Prostate Cancer: A Pilot Study.” The Journal of Urology, vol. 152, no. 5, 1994, pp. 1923–1926.
8. https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6
9. B. Widrow, R. Winter, 1960, “Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition”.
10. Winter, and Widrow. “MADALINE RULE II: A Training Algorithm for Neural Networks.” IEEE 1988 International Conference on Neural Networks, 1988, pp. 401–408.
11. Møller, Martin Fodslette. “Original Contribution: A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning.” Neural Networks, vol. 6, no. 4, 1993, pp. 525–533.
12. O. Osadcha, Z.Marszaek, Conference: Symposium for Young Scientists in Technology, Engineering and Mathematics SYSTEM, 2017, “Comparison of steepest descent method and conjugate gradient method”.
13. https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77

14. Haykin, Simon S., and Bernard Widrow. Least-Mean-Square Adaptive Filters. 2003.

15. Srivastava, Nitish, et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” Journal of Machine Learning Research, vol. 15, no. 1, 2014, pp. 1929–1958.

16. Aalst, van der Wmp Wil, et al. “Process Mining: A Two-Step Approach to Balance between Underfitting and Overfitting.” Software and Systems Modeling, vol. 9, no. 1, 2010, pp. 87–111.

17. Holzinger, Andreas, and Andreas Holzinger. “Machine Learning for Health Informatics.” Machine Learning for Health Informatics, 2016, pp. 1–24.

18. Hubel, David, and Torsten Wiesel. “David Hubel and Torsten Wiesel.” Neuron, vol. 75, no. 2, 2012, pp. 182–184.

19. Rumelhart, David E., et al. “Learning Internal Representations by Error Propagation.” Neurocomputing: Foundations of Research, 1988, pp. 673–695.

20. https://news.sophos.com/en-us/2017/09/21/man-vs-machine-comparing-artificial-and-biological-neural-networks/

21. https://www.breastcancer.org/symptoms/understand_bc/risk/factors

22. https://www.cancer.org/content/dam/cancer-org/research/cancer-facts-and-statistics/breast-cancer-facts-and-figures/breast-cancer-facts-and-figures-2017-2018.pdf

23. Bennett, Kristin. Decision Tree Construction via Linear Programming. 1992.

24. Bennett, Kristin P., and O. L. Mangasarian. “Robust Linear Programming Discrimination of Two Linearly Inseparable Sets.” Optimization Methods & Software, vol. 1, no. 1, 1992, pp. 23–34.

25. https://www.nationalbreastcancer.org/breast-cancer-diagnosis

26. https://en.wikipedia.org/wiki/Optical_character_recognition

27. Islam, Kh Tohidul, et al. “Handwritten Digits Recognition with Artificial Neural Network.” 2017 International Conference on Engineering Technology and Technopreneurship (ICE2T), 2017, pp. 1–4.

28. https://towardsdatascience.com/self-organizing-maps-ff5853a118d4

29. http://www.mit.edu/~9.54/fall14/slides/Class13.pdf

30. https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/