Workbook

student name: Endri Kastrati
student id: 24064580
instructor: Dr ABM Russel

Week 1:

During week 1, we were introduced to computer science project and the topic of investigation that was selected by our instructor, Dr ABM Russel titled "The role of big data in health and natural disasters". As students we were given the option of either working on medical or natural disaster data-sets. Because I had experience working with machine learning and medical data from a previous project I choose the first option. Having spoken with my instructor, I decided that I would like to apply some form of machine learning to medical data-sets that have been collected from Australia. Initially, I had decided that I would use some type of meta-algorithms that learn from the outputs of other trained models and that I would use python and the libraries tensorflow and scikit-learn for the implementation stage.

Week 2:

During week 2, I began researching for a medical data-set that was rich in content and features and such that I could apply machine learning techniques on it. I searched many online data repositories especially, the ones established by the Australian government, but I could not find a proper data-set that met my requirements.Fonrtunately, I came across the UCI machine learning repository and I found a thyroid disease data-set that was established by the James Cook University and donated to the repository in 1992. The data-set had been collected from Australia and it satisfied all the necessary conditions for the proper implementation of machine learning algorithms. During the beginning of week 2, but before the class meeting I wrote a prototype of an application that used different supervised machine learning algorithms to extract patterns from the data-set and then built another meta-algorithm that would learn from the outputs of those models. Using the python libraries, the implementation was very straight forward and not worthy of a computer science project. I decided that I would still work on the same data-set but I would built my own machine learning library, more precicely an artificial neural network library. During the class meeting I spoken with my instructor and I reccommneded that I would design an ANN library with the C programming language and the GNU scientific library. The instructor gave the green light and so during the weekend I spent most of my time, searching for peer-reviewed papers and books on the topic of artificial neural networks. All papers and book that were consulted for the project are included in the final project report.

Week 3:

During week 3, I had a clear understanding of my project requirements and objectives and so I started downloading and installing the necessary libraries and packages that would assist in the design of the project. As previously mentioned, I decide to implement the project using the C programming langauge and the GNU scientific library, version GSL-2.4. The reason that I choose GSL was because of it's excellent vector and matrix libraries. I installed the latest version by building the source code and then manually installed it into the system. To manage memory properly, I also installed valgrind, the memory checker tool. Having configured all the necessary tools and libraries I structured the project directories into src, include and datasets. The include directory would contain the header files and the src directory the corresponding source files. The dataset directory would contain all the data-sets that would be used during the project life cycle process and of course the final version of the project. During the class meeting, I finalised my computer science project requirements with the instructor. During the weekend I studied in detail the papers and books

that I had collected and made sure that I properly understood the mathematical concepts that underly the artificial neural networks. In order to gain a deep understanding, I proved by hand the mathematical properties of a perceptron, the geometric representation of the delta rule, the mathematical properties of the ADALINE network and lastly the properties of the multi-layer perceptron network. The hand written proofs can be found at the end of the paper.

Week 4:

At this stage the specifications and the functional and non-functional requirements had been finalised. It was time to start working on the physical implementation of the project. Having properly understood the abstract concept of artificial neural networks I broke it down into further smaller abstract concepts and those in turn into mathematical objects. From this point onwards I began working on the coding aspect of the project. I created the first UML diagram regarding the neural layer module of the project. This was the module that would represent the layer components of the neural network data structure. The implementation process was very straight forward and no bottlenecks were encountered during the coding part. The module was thoroughly tested as it represented the back-bone of the neural network data structure. During the meeting, I reported my progress to the instructor.

Week 5:

At this stage the first milestone had been completed. During week 5, I created the second and third UML diagrams regarding the neural network module. This module defined two data structures, the first one the neural configuration datatype and the second one was the neural network datatype. Once I completed the neural network library I spent most of the time working on the back-propagation algorithm, which turned out to be really hard to implement. During the class meeting, I spoken with my instructor about the progress of the project and presented the risks that had manifested and their corresponding mitigation techniques. After the meeting and during the entire weekend I focused on implementing correctly the back-propagation algorithm without any optimisation techniques at this point.

Week 6:

During week 6, I had completed an unoptimised version of the back-propagation algorithm and the dataset module of the project. The first prototype of the ANN library and the command line program was completed. I demonstrated the first complete iteration of the project during the class meeting and I received useful feedback from my peers and instructor regarding the correctness and usability aspects of the application. After the meeting and during the weekend, I fixed a few minor bugs and created a couple of trained models based on some datasets. I began working on an optimised version of the back-propagation algorithm using the momentum parameter method. Because of the highly abstract design of the ANN library based on the seperation of concerns principle, modifications took place only in the neural layer data structure and the nerual configuration data structure to enable the storage of the local gradient matrix and the synaptic weights matrix from the previous training epochs. By the end of the week, the first two project deliverables, namely the ANN library and the command line program, had been completed and thoroughly documented.

Week 7 and 8:

At this stage, having completed the major aspects of the computer science project, it was time to prove the correctness of the ANN library and the command line program. Based on the universal approximation theorem (look at final project report), ANN are able to approximate all smooth and continous functions. In order to prove this property I created three octave scripts called sinf.m, circlef.m and qudratic.m located in the datasets folder of the project. These files would run the command line program in curve-fitting mode and train models that approximate the qudratic function, the sin function and the semicircle function. After the models were trained the scripts would plot the results of the real functions and the results of the approximators to demonstrate the correctness of the ANN library. This was an important milestone, as it

validated the proper functionality of the library and the command line program. During the class meeting, I demonstrated the correctness of the first two deliverables to my peers and instructors and received useful feedback in return. After the meeting and during the weekend I performed several experimentations to gather empirical evidence on what kind of configuration to use to optain an optimal thyroid disease classifier. During the next meeting, I provided a detailed walk-through of the code and the modules to the class and instructor and received again useful feedback regarding the structure of the code and the design. At this point the only remaining deliverable was a minimalist web application interface that would interact with the ANN library via the command line program.

Week 9:

During week 9, I decided that I would use the nodejs environment and web sockets to built a simple web interface that would abstract away all the technical details regarding the classification of thyroid diseases. I spend most of the week building a simple web page using the bootstrap studio program and then converted it into the corresponding pug file so that it could be rendered by nodejs using the pug engine. I initialised an npm project inside the directory webapp and downloaded all the necessary node libraries. During the class meeting I had a completed prototype of the web interface and I demonstrated it to my peers and instructors and received constructive feedback regarding it's UI and usability aspects.

Week 10:

During week 10, I had taken into consideration the feedback from my peers and instructor and I became fixing and improving the corresponding user interface and making the web application more usable and comprehesive. At this point, all project deliverables had been completed, all functional and non-functional requirements had been met and the final version of the project was ready for demonstration and submission.

Week 11 and 12:

Week 11 and 12 were dedicated to the construction of the final project report, the test report, the workbook report and the presentation slides. At this stage all necessary requirements have been met and implemented successfully.

Research papers and books:

1. http://onlinelibrary.wiley.com/doi/10.1002/1097-0142(20010415)91:8%2B%3C1615::AID-CNCR1175%3E3.0.CO;2-L/full
2. http://www.springer.com/gp/book/9783319431611
3. https://pdfs.semanticscholar.org/f22f/6972e66bdd2e769fa64b0df0a13063c0c101.pdf
4. https://link.springer.com/chapter/10.1007/978-3-642-30223-7_87
5. http://www.springer.com/gp/book/9781447155706
6. http://ieeexplore.ieee.org/document/125864/?reload=true
7. http://www.cs.toronto.edu/~fritz/absps/momentum.pdf
8. https://tuppu.fi/wp-content/uploads/2017/05/The_C_Programming_Language_KandR.pdf
9. http://www.mosaic-industries.com/embedded-systems/_media/c-ide-software-development/learning-c-programming-language/understanding-and-using-c-pointers.pdf
10. https://dl.acm.org/citation.cfm?id=1538674
11. https://www.ncbi.nlm.nih.gov/pubmed/11309760
12. http://sipi.usc.edu/~kosko/FuzzyUniversalApprox.pdf
13. http://valgrind.org/docs/valgrind2007.pdf
14. https://www.ncbi.nlm.nih.gov/pubmed/6346016
15. http://www.tandfonline.com/doi/abs/10.1080/00401706.1987.10488301

Git log

==========

commit 5d7f47abff9194a6a19bdf12dfac724a43da8245
Author: Endri Kastrati <endriau@gmail.com>
Date: Mon Oct 16 21:25:10 2017 +1100

    Completed the final project report

commit 94ac88bb3fb244d526bda26e43ba2310f3c6c1c0
Author: Endri Kastrati <endriau@gmail.com>
Date: Mon Oct 9 18:24:56 2017 +1100

    completed a few changes to the user-interface

commit 950c396e07da3a0e2e3343927b2c454ff9586ced
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Oct 4 12:02:38 2017 +1100

    Completed a minimalistic documentation of the webapp code.

commit 42fd9797cdc9e1eef5e24621318f92c1d326e064
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Oct 4 11:36:44 2017 +1100

    Completed the user-interface for the online thyroidologist app.

commit 094a743aaa6eeae843a94dcd4f938307d87603e2
Author: Endri Kastrati <endriau@gmail.com>
Date: Thu Sep 14 15:31:03 2017 +1000

    Completed code documentation and implemented a couple of octave
    scripts that visualize the curve approximation for different
    functions.

commit 35b92dd7ec165683834a73c061fc808c951fdcfb
Author: Endri Kastrati <endriau@gmail.com>
Date: Sun Sep 10 12:42:58 2017 +1000

    Optimized the back-propagation algorithm using the momentum parameter.
    More specifically modified the neural layer data structure and the
    neural configuration data structure to enable the storage of the
    local gradient matrix and the synaptic weights matrix from the
    previous training epochs.

commit 79ec7035e46a92bd4c193dc34521842b285b18ff
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Sep 6 19:38:06 2017 +1000

    Fixed a few bugs and creating a couple of trained networks on some datasets.

commit 8825cf62518387062c08a10409d875d0fcf6206a
Author: Endri Kastrati <endriau@gmail.com>
Date: Tue Sep 5 14:56:28 2017 +1000

Completed the dumping of the trained neural network into binary
as well as the loading of the trained neural network from binary.

commit c91e252853b41c25e47371d402431ae1d1c20695
Author: Endri Kastrati <endriau@gmail.com>
Date: Mon Sep 4 20:32:52 2017 +1000

implemented the dataset data structure.

commit 601b37e2b58db1e8773304769452c0a460950d07
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Aug 30 22:57:58 2017 +1000

Structured the file hierarchy of the application and completed the
command line arguments interface for the training process of the
neural network.Next commit will include ability to dump model into
binary files and load them for predictions.

commit 123b2769b3c4a5e6cd023fa46bc52a2119ea2944
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Aug 30 15:38:43 2017 +1000

Changed optimization method

commit b1762b0be799be22b5ea052a4bcee8c0b76ff292
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Aug 30 13:55:39 2017 +1000

still debugging the resilient algorithm

commit 22d6674c1e255b86f70c93876842565a2c14e97b
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Aug 30 13:40:37 2017 +1000

debugging process

commit 2b18c56d64165cc4ef95b7d7bb4fe81ee31fc1e4
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Aug 30 12:56:57 2017 +1000

working on the resilient back-propagation algorithm

commit 61b6c04b4b82a1a529c91168a545dd4489f9da6b
Author: Endri Kastrati <endriau@gmail.com>
Date: Tue Aug 29 19:53:18 2017 +1000

Tested the neural network on both pattern classification problems
and curve fitting problems and it works correctly.

commit 2e5f3add01a4bc6b27a6d331aea5decdc6cd208e
Author: Endri Kastrati <endriau@gmail.com>
Date: Mon Aug 28 23:01:48 2017 +1000

completed the back propagation algorithm

commit 126380e18cab18d734553aae42315840a2b18409
Author: Endri Kastrati <endriau@gmail.com>

Date: Sat Aug 26 19:58:58 2017 +1000

    Started working on neural network data structure and the back-propagation algorithm

commit bb3103ed22e40b8be585b8c14b28a3b4545e1131
Author: Endri Kastrati <endriau@gmail.com>
Date: Wed Aug 23 16:03:12 2017 +1000

    finished the implementation of the neural layer data structure

==========

ENDRI KASTRATI

FIT3036 S2 2017 ☆ Team Visible

Pin Board

A Deep-learning approach to Thyroid disease classification

Build a multi-layer perceptron network that can classify thyroid gland disease with accuracy.

C programming language,GNU scientific library, GNU Make

http://archive.ics.uci.edu/ml/datasets/thyroid+disease

Synthesis and performance analysis of Multilayer neural network architectures,Optimization of the Back propagation algorithm for training Multilayer Perceptrons

1

project proposal/specifications

3 1

Course Progression

Assignment1

Assignment 2

Assignment 3

Assignment 4

To-Do

Doing

Done

week 1: Figure out what kind of a project you will work on.

week 2: Search for datasets, research papers and tools.

week 3: download the necessary packages and libraries.

week 4: Start implementing the artificial neural network.

1

Week 5: Completed the implementation of the neural layer data structure, currently working on the neural net data structure.

Week 6: Completed the ANN library. Also completed the command line interface for training a model.

1

Week 7: Computer Science project has been completed. The only remaining aspect is a user-interface for the digital thyroidologist

2

Week 8: Performed a set of tests to verify that the command line program is working as expected.

Week 9: Working on the user-interface

Week 10: Completed the user-interface for the digital thyroidologist.

1

Week 11: Completed the final project report.

3 1

Week 12: Completed the presentation

I'm stuck

Ideas

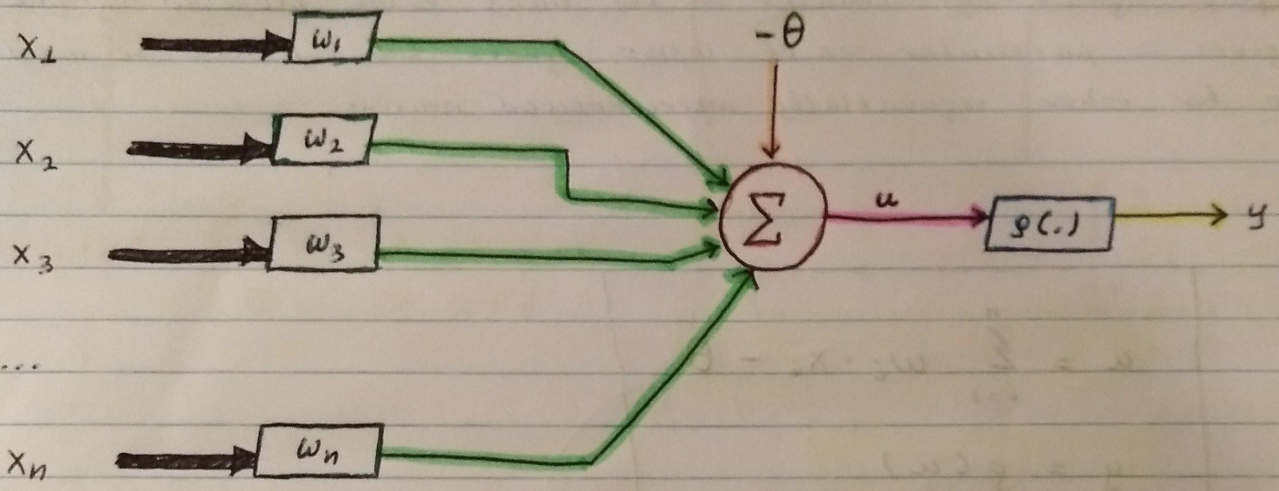
Wishlist

Graveyard

Trash

"Neural Networks"

"Artificial neuron - The perceptron network"



a) **Input signals** $(x_1, x_2, x_3, \ldots, x_n)$ are the signals or samples coming from the external environment and representing the values assumed by the variables of a particular application. The input signals are usually normalized in order to enhance the computational efficiency of learning algorithms.

b) **Synaptic weights** $(w_1, w_2, w_3, \ldots, w_n)$ are the values used to weight each one of the input variables, which enables the quantification of their relevance with respect to the functionality of the neuron.

c) **Linear aggregator** (Σ) gathers all input signals weighted by the synaptic weights to produce an activation voltage.

d) **Activation threshold** or **bias** (θ) is a variable used to specify the proper threshold that the result produced by the linear aggregator should have to generate a trigger value toward the neural output.

e) **Activation potential** (u) is the result produced by the difference between the linear aggregator and the activation threshold. if $u \geq \theta$ then the neuron produces an excitatory potential, otherwise it will be inhibitory.

f) **Activation function** (g) whose goal is limiting the neuron output within a reasonable range of values, assured by its own functional image.

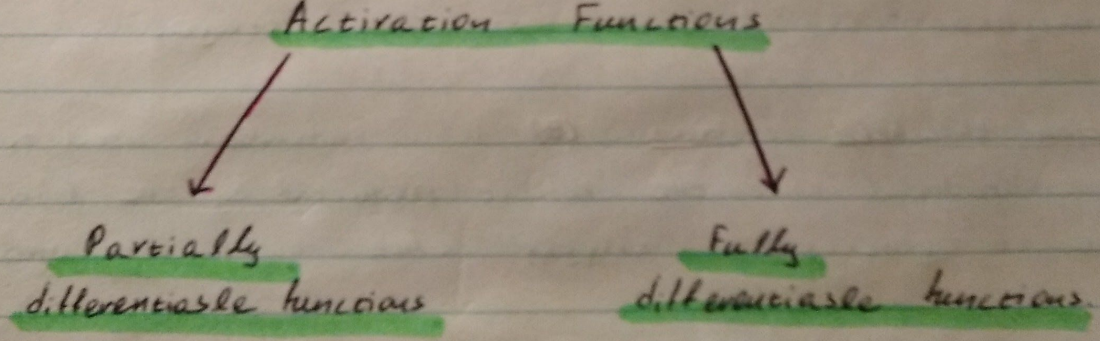
g) **Output signal** (y) consists on the final value produced by the neuron given a particular set of input signals, and can also be used as input for other sequentially interconnected neurons.

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \theta$$

$$y = g(u)$$

Thus the artificial neuron operation can be summarized as follows:

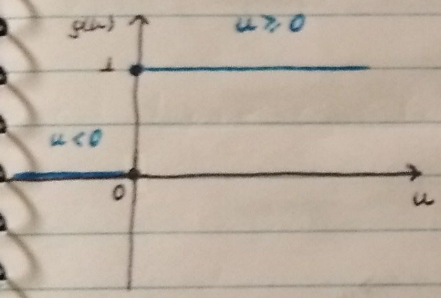
1. Present a set of values to the neuron, representing the input.
2. Multiply each input of the neuron to it's corresponding weight.
3. Obtain the activation potential produced by the weighted sum of the input signals and subtract the activation threshold.
4. Apply a proper activation function to limit the neuron output.
5. Compile the output by employing the neural activation function in the activation potential.



Partially differentiable Activation functions

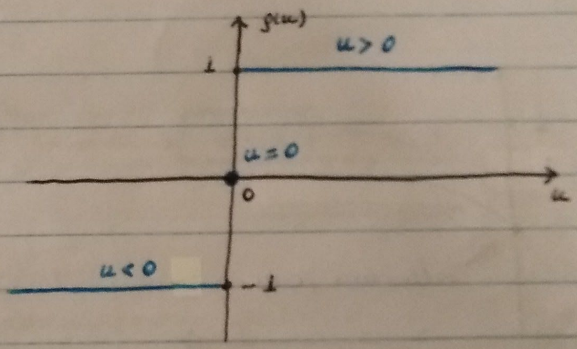
step function

$$g(u) = \begin{cases} 1, & \text{if } u \geq 0 \\ 0, & \text{if } u < 0 \end{cases}$$



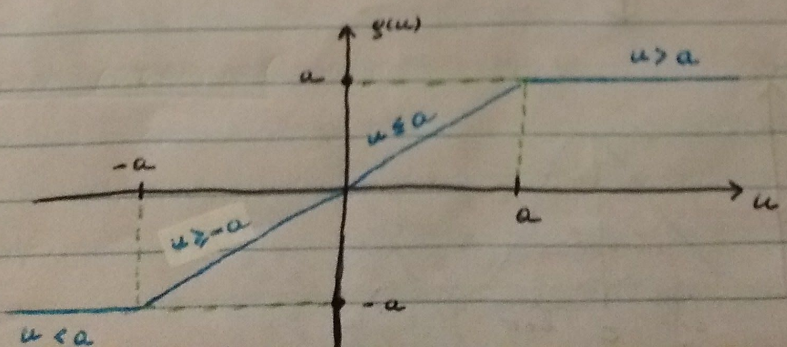
Bipolar step function

$$g(u) = \begin{cases} 1, & \text{if } u > 0 \\ 0, & \text{if } u = 0 \\ -1, & \text{if } u < 0 \end{cases}$$



Symmetric ramp function

$$g(u) = \begin{cases} a, & \text{if } u > a \\ u, & \text{if } -a \leq u \leq a \\ -a, & \text{if } u < a \end{cases}$$



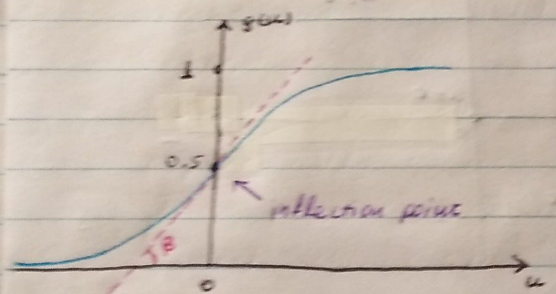
Fully differentiable activation functions

- Logistic function
- Hyperbolic tangent function
- Linear Function → $g(u) = u$

wormhole, entry

Logistic function

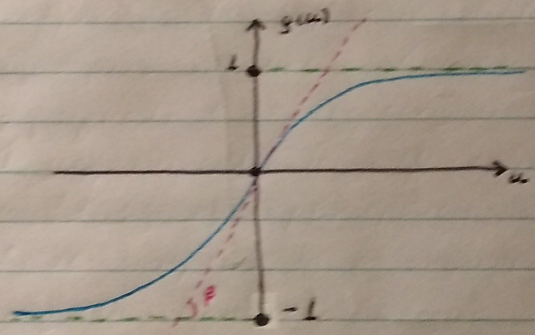
$$g(u) = \frac{1}{1 + e^{-\beta \cdot u}}$$



when $\beta \to +\infty$, logistic function approximates the step function.

Hyperbolic tangent function

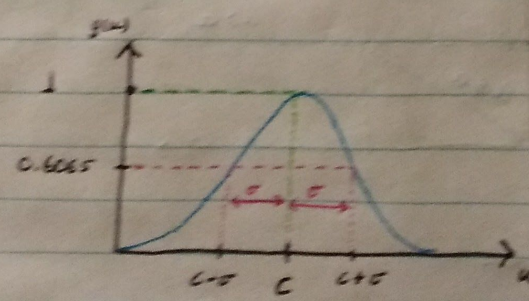
$$g(u) = \frac{1 - e^{-\beta \cdot u}}{1 + e^{-\beta u}}$$



when $\beta \to +\infty$, hyperbolic tangent function approximates the bipolar step function

wormhole, exit

Gaussian Function → $g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}}$

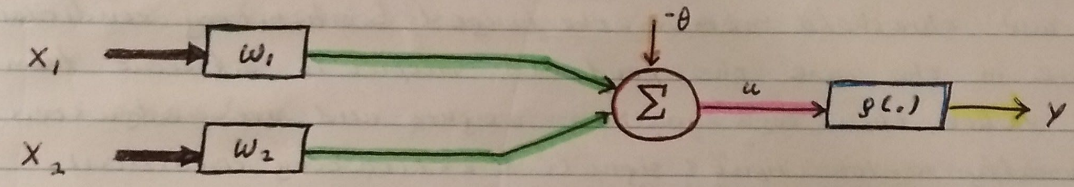


ExecutionHandler

perceptron classification: The perceptron network belongs to the class of single-layer feedforward architectures, because the information that flows in its structure is always from the input layer to the output layer, without any feedback from the output produced by its single output neuron.

"Mathematical Analysis of the Perceptron Network"

From the mathematical analysis of the Perceptron and by considering the bipolar step function, it becomes possible to verify that such network can be considered a typical case of a linear discriminator. To demonstrate this scenario, consider a Perceptron with only two inputs, as illustrated below:



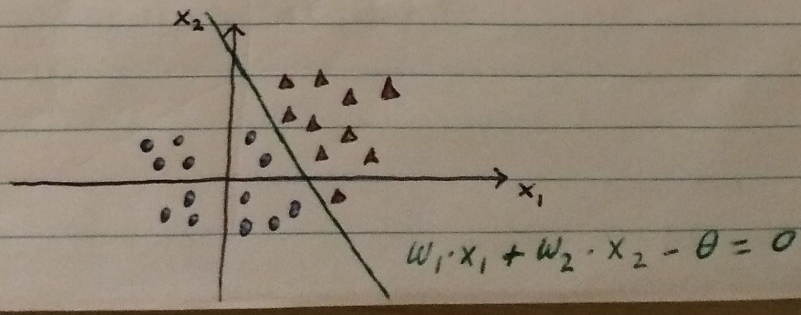
In mathematical notation, the perceptron output, which uses the bipolar step activation function, is given by:

$$y = \begin{cases} 1, & \text{if } \sum w_i \cdot x_i - \theta \geq 0 \iff w_1 \cdot x_1 + w_2 \cdot x_2 - \theta \geq 0 \\ -1, & \text{if } \sum w_i \cdot x_i - \theta < 0 \iff w_1 \cdot x_1 + w_2 \cdot x_2 - \theta < 0 \end{cases}$$

As the above inequalities are represented by linear equations, the classification boundary for this case (perceptron with two inputs) will be a straight line given by:

$$w_1 \cdot x_1 + w_2 \cdot x_2 - \theta = 0$$

Thus it is possible to conclude that the perceptron behaves as a pattern classifier whose purpose is to divide linearly seperable classes.



A perceptron with three inputs (three dimensions) will have the decision boundary represented by a plane.
A perceptron with n inputs (n dimensions) will have the decision boundary represented by a hyperplane.

Perceptron convergence theorem: For a perceptron to be applied as a pattern classifier the classes of the problem being mapped must be linearly seperable.

"Training process of the perceptron"

Hebb's learning rule: if the output produced by the perceptron coincides with the desired output, it's synaptic weights and threshold remain unchanged (inhibitory condition). Otherwise in the case the produced output is different from the desired value, then its synaptic weights and are adjusted proportionally to its input signals (excitatory condition).
This process is repeated sequentially for all training samples until the output produced by the perceptron is similar to the desired output for all samples. In mathematical notation, the rules for adjusting the synaptic weight w_i and threshold θ can se expressed respectively, by the following equations:

$$w_i^{current} = w_i^{previous} + (d^{(k)} - y) \cdot x_i^{(k)}$$

$$\theta^{current} = \theta^{previous} + (d^{(k)} - y) \cdot (-1)$$

As the same adjustment rule is applied to both the synaptic weights and threshold, it is possible to insert the threshold value θ within the synaptic weight vector. Thus

$$\vec{w}^{current} = \vec{w}^{previous} + \eta (d^{(k)} - y) \cdot x^{(k)}$$

In algorithmic notation, the inner processing performed by the perceptron can be described by the following expression:

$$\vec{w} \leftarrow \vec{w} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)}$$

where:

$\vec{w} = (\theta \quad w_1 \quad w_2 \quad \dots \quad w_n)^T$ is the vector containing the threshold and the weights.

$X^{(k)} = (-1 \quad x_1^{(k)} \quad x_2^{(k)} \quad \dots \quad x_n^{(k)})$ is the kth training sample.

$d^{(k)}$ is the desired value for the kth training sample.

y is the output produced by the perceptron.

η is a constant that defines the learning rate. $(0 < \eta < 1)$

Perceptron training algorithm

```
procedure perceptron_train(X, D):
        X = {x^(1), x^(2), ..., x^(n)}  training sample.
        D = {d^(1), d^(2), ..., d^(n)}  output of each sample

W := {w_1, w_2, ..., w_n}
for ∀w_i ∈ W:  w_i := rand()
η := a,  a ∈ (0, 1)
epoch := 0
do:
    error := "none"
    for ∀x_i ∈ X and ∀d_i ∈ D:
        u := W^T · X_i
        y := signal(u)
        if y ≠ d_i:
            W := W + η · (d_i - y) · x_i
            error := "existent"
    epoch := epoch + 1
while error == "existent"
end
```

Perceptron operation phase algorithm

procedure perceptron − predict(W, x_i):

x_i = some unseen input sample.

W = the adjusted weights of the perceptron.

$u := W^T \cdot x_i$

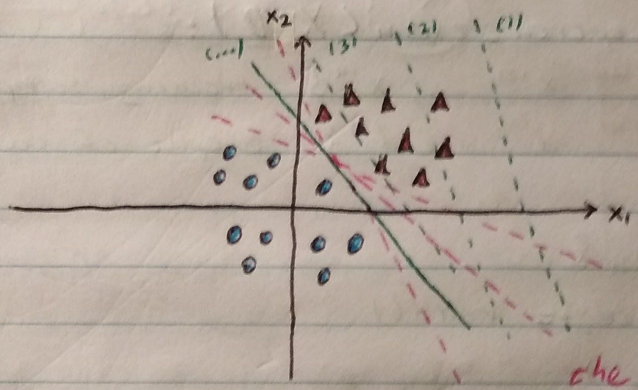
$y := signal(u)$

if $y == -1$: $x \in$ { the class associated with -1 }

else: $x \in$ { the class associated with 1 }

end

The perceptron training process tends to move the classification hyperplane continuously until it meets a decision boundary that allows the seperation of both classes.



← separation line adjustment after a certain number of epochs.

the seperabiling line produced is not unique and depending on the initial values of W the number of epochs may vary.

a) the network will diverge if problem is nonlinearly seperable. The strategy for this scenario is to limit the training process by a maximum number of epochs.

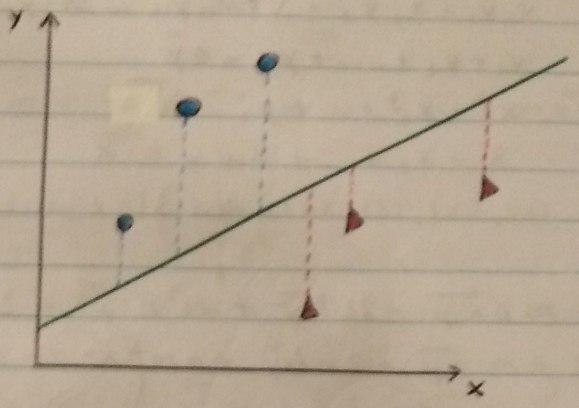
b) small value learning rate mitigate instability.

c) The number of epochs depends on the initial values of W.

d) The closer is the decision surface to the separating boundary, the fewer epochs are usually required for convergence.

e) normalizing input signals enhances performance.

"Lease mean square"
a.k.a "Delta rule"



$y = (mx + b)$

we want to find m, b such that it minimizes the squared error of all points from the line.

We define as error the vertical distance of a point from a line, namely $\text{error} = y_i - y = y_i - (mx_i + b)$. The reason we take the square is because of some nice mathematical properties such as being differentiable everywhere and having uniquely nice geometric properties, which the absolute does not have.

The total square error against the line $y = mx + b$ is the sum of the squared errors of each point (x_i, y_i) against the line.

$$E(m,b) = (y_1 - (mx_1 + b))^2 + (y_2 - (mx_2 + b))^2 + \dots + (y_n - (mx_n + b))^2$$

We want to find $(m^*, b^*) \in \mathbb{R}^2$ s.t. $E(m^*, b^*) \leq E(m, b)$

$$\begin{aligned}
E(w,b) &= y_1^2 - 2(mx_1 + b)y_1 + (mx_1 + b)^2 \\
&+ y_2^2 - 2(mx_2 + b)y_2 + (mx_2 + b)^2 \\
&+ \dots \\
&+ y_n^2 - 2(mx_n + b)y_n + (mx_n + b)^2 \\
&= y_1^2 - 2y_1 mx_1 - 2by_1 + m^2x_1^2 + 2mx_1 b + b^2 \\
&+ y_2^2 - 2y_2 mx_2 - 2by_2 + m^2x_2^2 + 2mx_2 b + b^2 \\
&+ \dots \\
&+ y_n^2 - 2y_n mx_n - 2by_n + m^2x_n + 2mx_n b + b^2 \\
&= (y_1^2 + y_2^2 + \dots + y_n^2) - 2m(x_1y_1 + x_2y_2 + \dots + x_ny_n) - 2b(y_1 + y_2 + \dots + y_n) \\
&+ m^2(x_1^2 + x_2^2 + \dots + x_n^2) + 2mb(x_1 + x_2 + \dots + x_n) + nb^2
\end{aligned}$$

we know that:

$$y_1^2 + y_2^2 + \dots + y_n^2 = n \cdot \overline{y^2} \quad (1)$$

$$x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \overline{x \cdot y} \quad (2)$$

$$y_1 + y_2 + \dots + y_n = n\bar{y} \quad (3)$$

$$x_1^2 + x_2^2 + \dots + x_n^2 = n\overline{x^2} \quad (4)$$

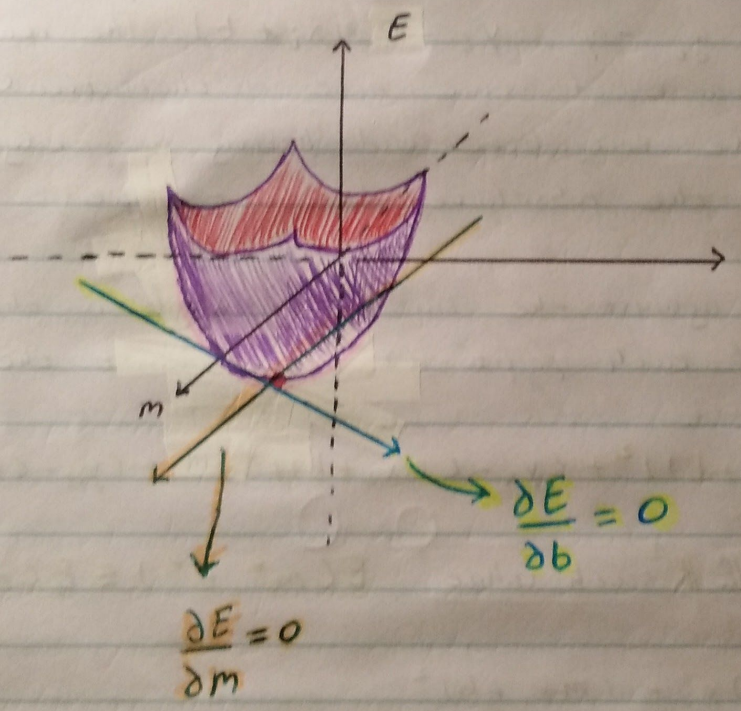
$$x_1 + x_2 + \dots + x_n = n \cdot \bar{x} \quad (5)$$

By substituting (1), (2), (3), (4) and (5) into $E(m,b)$:

$$E(m,b) = n \cdot \overline{y^2} - 2mn\overline{x \cdot y} - 2nb\bar{y} + m^2 n \cdot \overline{x^2} + 2mbn\bar{x} + nb^2 \iff$$

$$E(m,b) = n \cdot \overline{y^2} - 2 \cdot n \cdot m\ \overline{xy} - 2nb\bar{y} + nm^2\bar{x}^2 + 2nmb\bar{x} + nb^2$$

Because we want to find the minimums of $E(m,b)$, that occurs when the tangent line with respect to m is flat and parallel to the m-axis and the tangent line with respect to b is flat and parallel to the b-axis.



or equivalently $\nabla E(m,b) = \vec{0} \iff \left(\frac{\partial E}{\partial m}, \frac{\partial E}{\partial b}\right) = \vec{0}$

$$\iff \left(\frac{\partial E}{\partial m}, \frac{\partial E}{\partial b}\right) = (0,0) \iff \frac{\partial E}{\partial m} = 0 \text{ and } \frac{\partial E}{\partial b} = 0$$

And we know from theory that maximums or minimums exist at the points where the derivatives are equal to zero.

$$\frac{\partial E}{\partial m} = -2n\,\bar{xy} + 2n\,\bar{x}^2 m + 2bn\,\bar{x} = 0 \quad (6)$$

$$\frac{\partial E}{\partial b} = -2n\,\bar{y} + 2mn\,\bar{x} + 2bn = 0 \quad (7)$$

We have two equations with two unknowns. Without breaking the symmetry, we multiply (6) with $\frac{1}{2n}$:

$$-\frac{2n}{2n}\overline{xy} + \frac{2n}{2n}\overline{x^2}m + \frac{2n}{2n}b\bar{x} = 0 \Leftrightarrow -\overline{xy} + m\overline{x^2} + b\bar{x} = 0 \quad (8)$$

equivalently we multiply (7) with $\frac{1}{2n}$:

$$-\frac{2n}{2n}\bar{y} + \frac{2n}{2n}m\bar{x} + \frac{2n}{2n}b = 0 \Leftrightarrow -\bar{y} + m\bar{x} + b = 0 \quad (9)$$

From (8), (9) we get: $\left\{ \begin{array}{l} \overline{x^2}m + b\bar{x} = \overline{xy} \\ m\bar{x} + b = \bar{y} \end{array} \right\} \Rightarrow$

$b = \bar{y} - m\bar{x}$ and $m\overline{x^2} + (\bar{y} - m\cdot\bar{x})\bar{x} = \overline{xy} \Leftrightarrow$

$$m\overline{x^2} + \bar{y}\bar{x} - m(\bar{x})^2 = \overline{xy} \Leftrightarrow$$

$$-m(\bar{x})^2 + m\overline{x^2} = \overline{xy} - \bar{x}\bar{y}$$

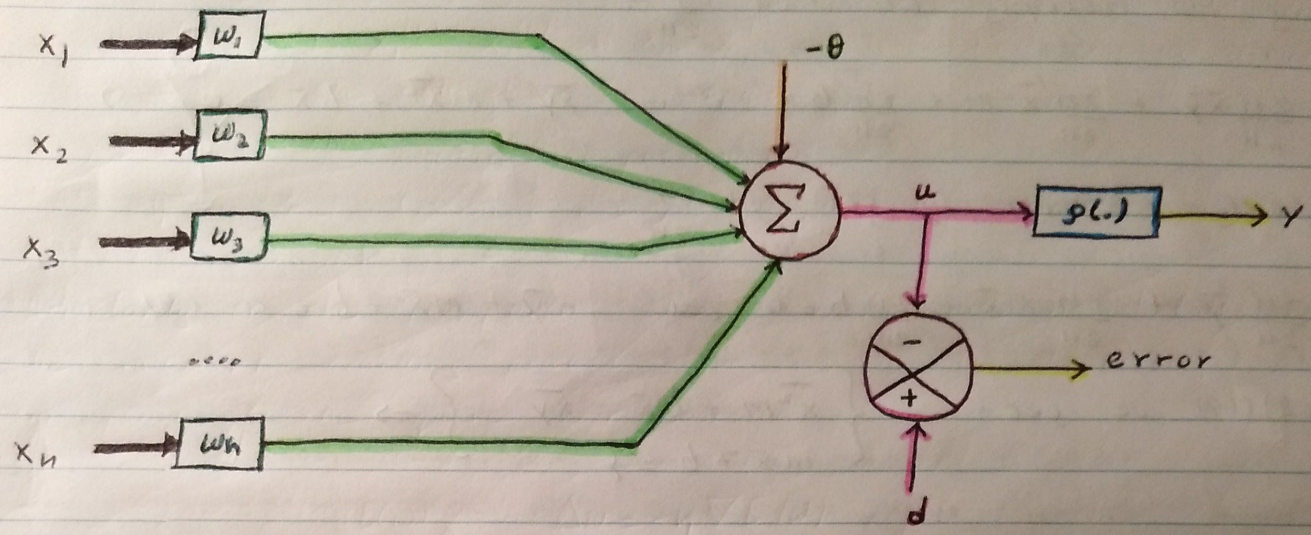
$$m(\overline{x^2} - (\bar{x})^2) = \overline{xy} - \bar{x}\bar{y} \Leftrightarrow$$

$$m = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - (\bar{x})^2}$$

Therefore:

$$\boxed{b = \bar{y} - m\cdot\bar{x}} \quad \text{and} \quad \boxed{m = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - (\bar{x})^2}}$$

"The Adaline Network"



The Adaline network is very similar to the perceptron network, that is, it belongs to the single-layer feedforward architecture and the classes from the problem being mapped must be linearly seperable in order to be completely identified by the network.

The main difference between the Adaline and the perceptron is on the learning rule used to adjust weights and threshold.

The rule performs the minimization of the squared error between u and d to adjust the weight vector $\vec{w} = [\theta \; w_1 \; w_2 \; \dots \; w_n]^T$ of the network. In summary the objective is to obtain an optimal $\vec{w}^*$ so that the squared error $E(\vec{w}^*)$ of the whole sample set is as low as possible. In mathematical notation, considering an optimal weight configuration, it can be stated that:

$$E(\vec{w}^*) \leq E(\vec{w}) \qquad \forall \vec{w} \in \mathbb{R}^{n+1}$$

The function of the squared error related to the p training samples is defined by:

$$E(\vec{w}) = \frac{1}{2} \sum_{k=1}^{p} (d^{(k)} - u)^2 \qquad (1)$$

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \theta \qquad (2)$$

substituting (2) into (1) we get:

$$E(\vec{\omega}) = \frac{1}{2} \sum_{k=1}^{p} \left(d^{(k)} - \left(\sum_{i=1}^{n} w_i \cdot x_i^{(k)} - \theta \right) \right)^2 \quad \Leftrightarrow$$

$$E(\vec{\omega}) = \frac{1}{2} \sum_{k=1}^{p} \left(d^{(k)} - \left(\omega^T \cdot x^{(k)} - \theta \right) \right)^2 \qquad (3)$$

Thus equation (3) computes the mean squared error by processing the p training samples provided for the training process of the Adaline network.

The next step consists of the application of the gradient operator on the mean squared error with respect to vector ω, in order to search for an optimal value for the squared error function given by (3), that is:

$$\nabla E(\vec{\omega}) = \frac{\delta E(\vec{\omega})}{\delta \vec{\omega}} \quad \Leftrightarrow$$

$$\nabla E(\vec{\omega}) = 2 \cdot \frac{1}{2} \sum_{k=1}^{p} \left(d^{(k)} - (\omega^T \cdot x^{(k)} - \theta) \cdot (-x^{(k)}) \right) \quad \Leftrightarrow$$

$$\nabla E(\vec{\omega}) = -\sum_{k=1}^{p} \left(d^{(k)} - (\omega^T \cdot x^{(k)} - \theta) \cdot x^{(k)} \right)$$

Finally the steps for adapting the weight vector must be executed in the opposite direction of the gradient, because the optimization goal is to minimize the squared error which in mathematical terms means to find the global minimum of $E(\vec{\omega})$. We know from calculus that the global extrema are located at $\vec{\omega}$ s.t. $\nabla E(\vec{\omega}) = \vec{0}$. But because computers cannot solve $\nabla E(\vec{\omega}) = \vec{0}$ the way we humans do, we have to approximate it via iterative steps η until convergence. In this condition, the variation $\Delta\vec{\omega}$ to update the Adaline weights vector is given by: $\Delta\omega = -\eta \nabla E(\vec{\omega}) \Leftrightarrow$

$$\vec{\omega}^{\,current} - \vec{\omega}^{\,previous} = \eta \sum_{k=1}^{p} \left(d^{(k)} - u \right) \cdot x^{(k)} \quad \Leftrightarrow$$

$$\vec{\omega}^{\,current} = \vec{\omega}^{\,previous} + \eta \cdot \left(d^{(k)} - u \right) \cdot x^{(k)}, \quad \forall k \in \{1, 2, \ldots, p\}$$

In algorithmic notation:

$$\vec{\omega}^{\,current} \leftarrow \vec{\omega}^{\,previous} + \eta \cdot \left(d^{(k)} - u \right) \cdot x^{(k)}, \quad \text{with } k = 1, 2, \ldots, p$$

where:

$\vec{w} = [\theta \quad w_1 \quad w_2 \quad \dots \quad w_n]^T$, vector containing threshold and weights

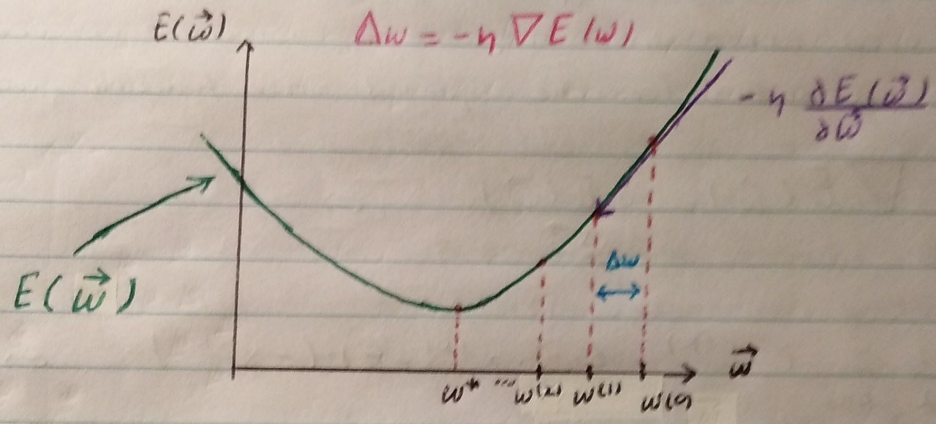
$x^{(k)} = [-1 \quad x_1^{(k)} \quad x_2^{(k)} \quad \dots \quad x_n^{(k)}]^T$, is the kth training sample

$d^{(k)}$ is the kth desired output signal.

u, is the activation potential.

η, the learning rate ($0 < \eta < 1$)

Similarly to the perceptron, the learning rate η defines how fast the network training process advances in the direction of the minimum point of the square error function.



The stopping criterion is stipulated by employing the mean square error $\bar{E}(\vec{w})$ with respect to all training samples, and is defined:

$$\bar{E}(\vec{w}) = \frac{1}{P} \sum_{k=1}^{P} (d^{(k)} - w)^2 .$$

The algorithm converges when the difference of the mean square between two successive epochs is small enough, that is:

$$| \bar{E}(\vec{w}^{\,current}) - \bar{E}(\vec{w}^{\,previous}) | \leq \varepsilon$$

where ε is the precision required for the convergence process. It is specified by taking into account the performance requirements of the application mapped by the Adaline network.

```
procedure   adaline_train(X, D, η, ε):
                X = { x^(1), x^(2), ..., x^(n) }, training sample.
                D = { d^(1), d^(2), ..., d^(n) }, output of each sample.
                η = learning rate.
                ε = precision of convergence.

W := { w_1, w_2, ..., w_n }
for ∀w_i ∈ W:  w_i := rand()
epochs := 0
do:
    mse_previous := mse(W, X, D)
    for ∀x_i ∈ X and ∀d_i ∈ D:
        u := W^T · x_i
        w := w + η·(d_i - u)·x_i
    epochs := epochs + 1
    mse_current := mse(W, X, D)
while abs(mse_current - mse_previous) ≤ ε
end


procedure   mse(W, X, D):
                W = [θ w_1 w_2 ... w_n]^T, the weights vector.
                X = { x^(1), x^(2), ..., x^(n) }, the training sample.
                D = { d^(1), d^(2), ..., d^(n) }, output of each sample.

value := 0
for ∀x_i ∈ X and ∀d_i ∈ D:
    u := W^T · x_i
    value := value + (d_i - u)^2
value := value / 101
end
```

procedure adaline_predict(X, W):

X = Some unseen input sample.

$W = [\theta \; w_1 \; w_2 \; \dots \; w_n]^T$, adjusted weights.

$u := W^T \cdot X$

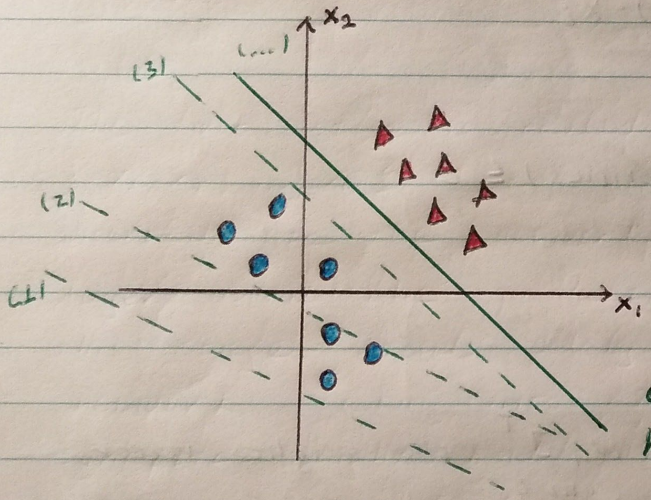
$y := signal(u)$

if $y == -1$: $x \in$ { the class associated with -1 }

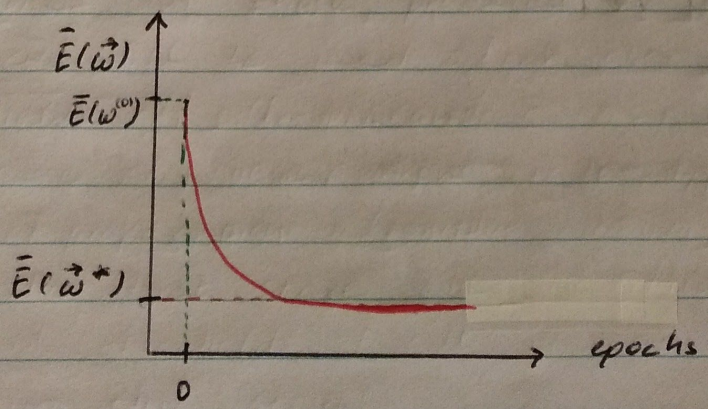
if $y == 1$: $x \in$ { the class associated with 1 }

end

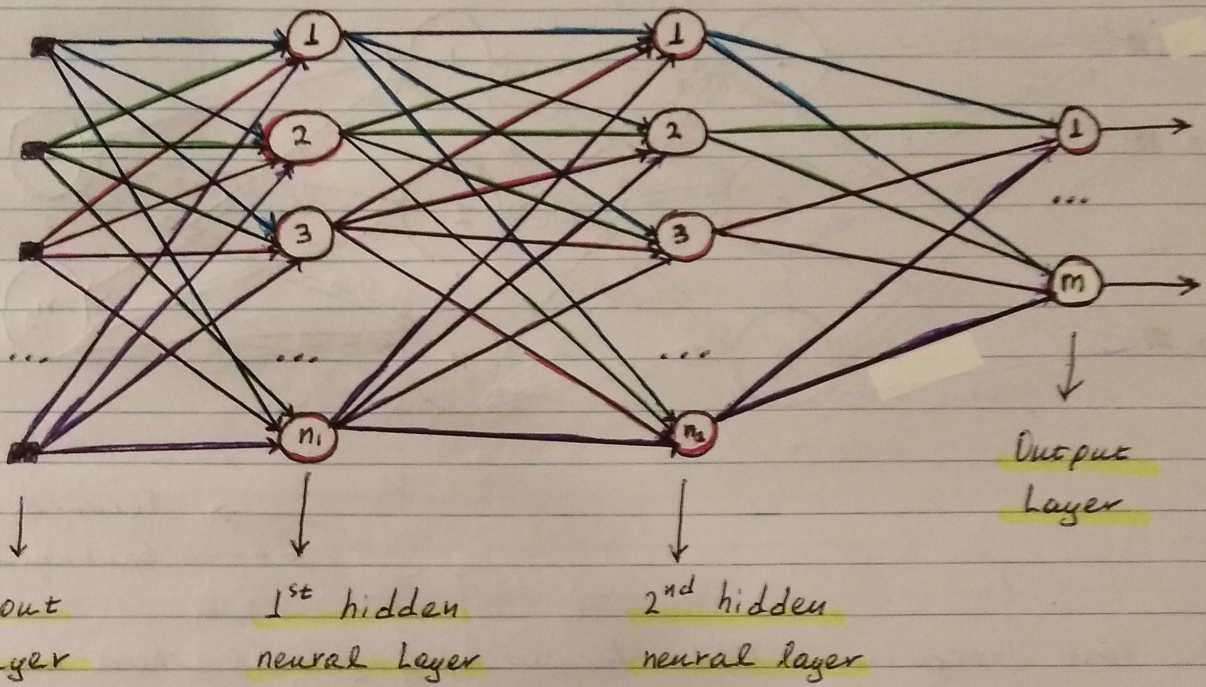
The convergence process moves the hyperplane toward the optimal seperasility boundary, which corresponds to the w^* value that minimizes the squared error function.



The mean squared error curve of the Adaline is always descendant meaning it decreases as the number of training epochs increases. It estaslishes at a constant value when the minimum point of the squared error function is reached.

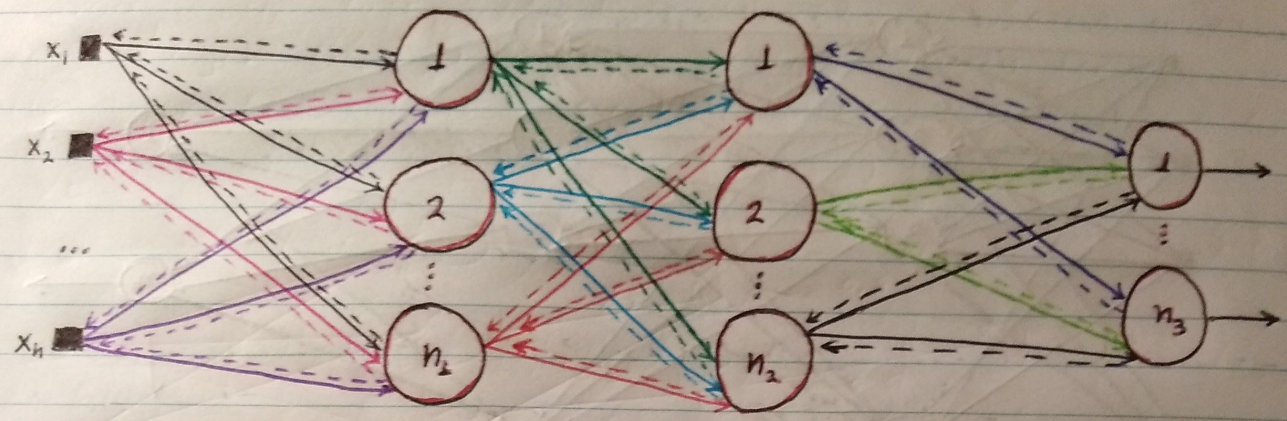


"Multilayer Perceptron Networks"



In summary, in contrast to the Perceptron or Adaline networks, in which a single neuron is responsible for the full mapping of the whole process, the knowledge related to the behavior of the input and outputs of the system will be distributed among all neurons composing the MLP. The stimuli or signals are presented to the network on its input layer. The intermediate layers, on the other hand, extract the majority of the information related to the system behavior and codify them using synaptic weights and thresholds of their neurons, thus forming a representation of the environment where the particular system exists. Finally the neurons of the output layer receive the stimuli from the neurons of the last intermediate layer, producing a response pattern which will be the output generated by the network.

"Training process of the Multilayer perceptron"



→ Forward stage

←--- backward stage

The first stage is called forward propagation, where the signals $\{x_1, x_2, \ldots, x_n\}$ of a given sample from the training set are inserted into the network inputs and are propagated layer-by-layer until the production of the corresponding outputs. Thus this stage intends solely in obtaining the responses from the network, taking into account only the current values of the synaptic weights and thresholds of its neurons, which will remain unmodified during the execution stage.

Next, the response produced by the network outputs are compared to the respective available desired responses, since it is a supervised learning process, as mentioned earlier. It is important to note that, considering an MLP network with n_3 neurons in its output layer, the respective n_3 deviations (errors) between the desired responses and those produced by the output neurons are calculated and will be used after that to adjust the weights and thresholds of all neurons.

Therefore, because of these errors, it is applied the second stage of the backpropagation algorithm, known as backward propagation. Unlike the first stage the modifications of the synaptic weights and thresholds of all neurons of the network are executed during this stage.

This successive application of forward and backward stages results in the gradual reduction of the sum of the errors.