ARISTOTLE UNIVERSITY OF THESSALONIKI

Thesis Title

by

Iakovos Marios Tsouros

A thesis submitted in partial fulfillment for the
Graduate degree

in the
Faculty of Sciences
School of Physics

Supervising Professor: Dr. Panagiotis Argyrakis

Date

Declaration of Authorship

I, [author's name], declare that this thesis titled, [thesis title] and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Inspiring quote goes here (optional)

Quote's attribution

ARISTOTLE UNIVERSITY OF THESSALONIKI

Abstract

Faculty of Sciences
School of Physics

Graduate Degree

by Iakovos Marios Tsouros

Abstract goes here.

Acknowledgements

Acknowledgements go here.

Contents

List of Figures

List of Tables

Abbreviations

Acronym	**W**hat (it) **S**tands **F**or

Dedication (optional)

Chapter 1

Introduction

1.1 Graphs

In this subsection, the main aspects of graph theory are briefly presented.

1.1.1 Introduction

In the real world, many problems can be described by a diagram connecting a set of points with lines, joining pairs of these points, or even creating loops on a single point. A simple example of that would be a set of points representing people with lines connecting acquintances, or points representing atoms and lines representing chemical bonds, creating a representation of a molecule as a graph attribute. In the examples above, the only information contained is whether two points are associated, with the manner being disregarded. The concept of a graph consists of a mathematical abstraction of the above. [1]

Definition 1.1. Mathematically, in its simplest form, a **graph** is an ordered pair[1] $G = (V, E)$ of:

- **V**, a set of vertices (also known as nodes).
- $E \subseteq \{\{x, y\} | x, y \in V \; x \neq y\}$, which is the set of **edges** which consists of unordered pairs of vertices that connect two nodes.

This type of object is called an **undirected simple graph** to avoid confusion with other types.

[1] An ordered pair (a, b) is a pair of objects in which the order of appearance or insertion is significant; the ordered pair (a, b) is different than (b, a) unless $a = b$. An unordered pair is a set of the form a, b is a set having two elements with no relation between them and $a, b = b, a$.

Definition 1.2. A *graph* G is an ordered pair $(V(G), E(G))$ consisting of a set $V(G)$ of *vertices* (also called *nodes* or *points*) and a set $E(G)$, disjoint from $V(G)$ which consists of *edges* (also called *links* or *lines*) together with an incidence function ψ_G that associates with each edge of G an unordered pair of not necesserily distinct vertices of G. If e is an edge and u and v are vertices such that $\psi_G = u, v$ then e is said to *join* u and v, and the vertices u and v are called the *ends* of e. We denote the numbers of vertices and edges G by $u(G)$ and $e(G)$ which two parameters are called the *order* and *size* of G, respectively [1].

In short, we can define a **graph** as an ordered triple $G = (V, E, \phi_G)$ consisting of:

- V, a set of *vertices*
- E, a set of *edges*
- $\phi_G : E \to \{\{x, y\}|x, y \in V \; and \; x \neq y\}$ an *incidence function* mapping every edge to an unordered pair of vertices - an edge associated with two distinct vertices. The incidence function is a function of the edges.

This type of object is called an *undirected multigraph*, to avoid confusion. Note, that the above definition of the *incidence function* does not allow for *loops* (mappings of an edge on the same vertex).

A *loop* is a an edge that allows a connection of a vertex to itself and a graph can be defined to either allow or disallow the presence of loops. Some authors allow for loops to exist on *multigraphs* [2], while other consider these kind of graphs to exist in a different category, called *pseudographs* [3]. Allowing loops requires modifying the incidence function so they can be supported. The new incidence function can be written as:

$$\phi_G : E \to \{\{x, y\}|x, y \in V\} \tag{1.1}$$

The example presented below should better illustrate clarify the definition (of a pseudograph).

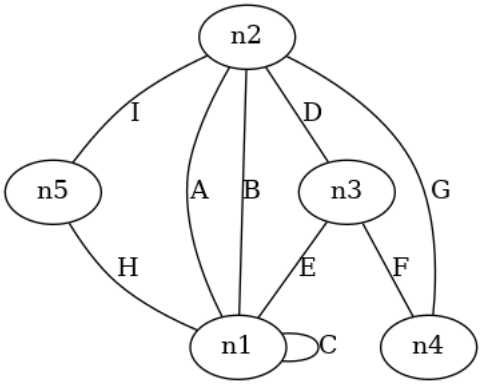


FIGURE 1.1: An undirected pseudograph with labeled nodes and edges.

Example 1.1.

For the graph presented in Figure 1.1 the following can be assumed:

$$G = (V(G), E(G))$$

and

$$V(G) = \{n_1, n_2, n_3, n_4, n_5\}$$
$$E(G) = \{A, B, C, D, E, F, G, H, I\}$$

and the incidence function is defined as:

$$\psi_G(A) = n_1n_2, \quad \psi_G(B) = n_1n_2, \quad \psi_G(C) = n_1n_1, \quad \psi_G(D) = n_2n_3,$$
$$\psi_G(E) = n_1n_3, \quad \psi_G(F) = n_3n_4, \quad \psi_G(G) = n_2n_4, \quad \psi_G(H) = n_1n_5,$$
$$\psi_G(I) = n_2n_5$$

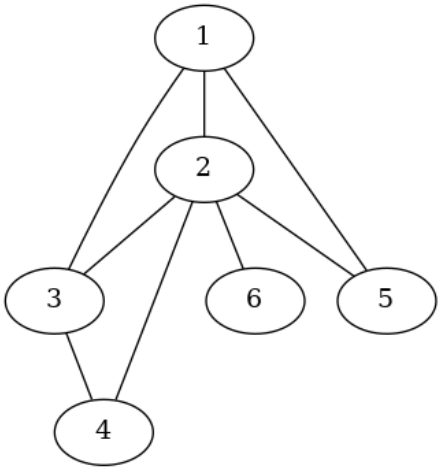
It should now be clear that with the newer definition of ϕ_G, self loops are now possible. Additionaly, even though this was not prohibited by the previous definition, it is worth noting that a node can be connected to another with multiple edges (or multiedges), or that it can have zero connections to other nodes. Generally, V is assumed to be a non-empty set, but E can be empty.

It is now possible to define some characteristic attributes of graphs:

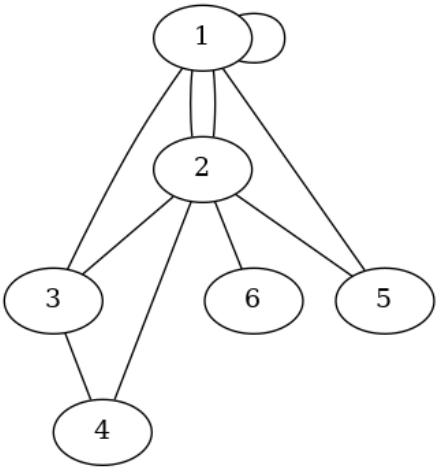
- $|V|$: the **order** of a graph is the number of its vertices.
- $|E|$: the **size** of a graph is the number of its edges.
- The **degree** (or **valency**) of a single node is the number of edges connected to it. The **degree** of a graph is the maximum number of edges connected to a single vertex that belongs to it.
- The edges of create a *homogenous relation*[2] $\sim$ on the vertices of the graph that is called **adjacency relation**; for each edge (x, y), its endpoints *x, y* are said to be **adjacent** to each other, denoted by $x \sim y$. This property will be particulary useful when the adjacency matrix is defined in the following section.

It can be inferred from the above definitions and attributes that for an undirected graph of order *n*, the maximum *degree* of a node is $n - 1$ and and maximum *size* of a graph is $n(n - 1)/2$.

[2] A **homogenous relation** (or **endorelation**) over a set *X* is a set of assignments (binary relations) over *X* and itself; i.e. it is a subset of the cartesian product $X \times X$



(A) Multigraph with no loops and multiple edges.



(B) Mutligraph with loops and multiple edges.

FIGURE 1.2: Two undirected multigraphs.

In this section only *undirected* graphs were considered, which are graphs with edges with no orientation. A whole other class of graph objects with edges which have orientation exists, called *directed graphs*. These kind of graphs objects are out of scope for this thesis and will not be presented.

1.1.2 Adjacency Matrix

Definition 1.3. The *adjacency matrix* is the fundamental mathematical representation of a graph. It is a square matrix, the elements of which represent which pair of nodes are *adjacent* or not. Thus, the adjacency matrix **A** of a graph of order n is the $n \times n$ matrix with elements A_{ij} such that:

$$A_{ij} = \begin{cases} 1 & \text{if there exists at least one edge connecting } i \text{ and } j \\ 0 & \text{if there no edges connecting those edges directly.} \end{cases} \tag{1.2}$$

Considering the simple undirected graph of Figure 1.3a we can construct the following adjacency matrix:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

TABLE 1.1: Adjacency matrix for Figure 1.3a

For this simple network, which has no loops and only one edge connect two nodes, the diagonal matrix elements are always zero and the matrix is symmetric, as for each edge connecting i and j there is a representation for the j to i connection as well.

In a more complex case, such as the one presented in Figure 1.2b where loops and multiedges are present an adjacency matrix can still be constructed. In this case, a multiedge is represented by setting the value of the corresponding A_{ij} value equal to the multiplicity of the edge. In this case, $A_{12} = A_{21} = 2$.

For loops, the most common representation in the case of undirected graphs is to still set the value of the A_{ii} element equal to 2 (i.e. $A_{11} = 2$ in the example presented). Essentially, an edge of a loop has two ends that connect to the same node, thus the result [4, p. 68]. Additionaly, defining the matrix in such manner, allows for better computations and is consistent with the definition of the representation of an edge connecting two nodes of an undirected graph [5, p. 108].

Thus, the adjacency matrix for the graph of Figure 1.2b is

$$A = \begin{pmatrix} 2 & 2 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

TABLE 1.2: Adjacency matrix for Figure 1.2b

Remark 1.4. Note that the degree of a node can be easily found by summing the values of the column or row of the adjacency matrix that correspond to said node.

1.1.2.1 Adjacency List

An alternative to the adjacency matrix is the *adjacency list.* An adjacency list is a collection of lists, one for each node i. Each list contains the labels of the nodes that i is connected to, and its the most common method for storing networks on computers as it requires less space. It is also possible to represent edge attributes in an adjacency list by appending an extra column which holds these values. An example for the graph presented in Figure 1.2b with multiedges and loops:

Node	Neighbors
1	1,2,2,5,3
2	1,1,3,5,6,4
3	1,2,4
4	3,2
5	1,2
6	2

TABLE 1.3: Adjacency list for Figure 1.2b

Each edge of the network appears twice, thus for a network with m edges the size of the adjacency list would be $2m$, much smaller compared to the $n \times n$ matrix required to build an adjacency matrix. This is particularly useful in networks which are relatively *sparse*[3], but have a high order.

1.1.3 Graph Laplacian

The graph lacplacian is another representation matrix representation of a graph. In its simplest form, for a simple, undirected and unweigheted network of order n, is a $n \times n$ matrix **L** with elements:

$$L_{ij} = \begin{cases} k_i, & \text{if } i = j \\ -1, & \text{if i} \neq \text{j and there exists an edge connecting } i \text{ and } j \\ 0, & \text{everywhere else} \end{cases}$$

where k_i is the degree of the node. Another way to write the graph Lacplacian is as

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

where **D** is a diagonal matrix containing the degrees of the nodes.

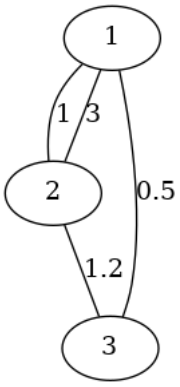
[3] *Sparse* networks are networks with a much lower number of edges than those possible.

In a similar manner the Laplacian matrix can be constructed for weighted networks, by replacing the the degree k_i of a node with the relevant matrix elements.

The Laplacian matrix has many applications in the study of dynamical systems, random walks and graph visualization. It has also found applications in graph neural networks, as its spectral decomposition allows the construction of low dimensional embeddings with applications in graph neural networks, such as ChebNet [6] which will be discussed in depth later.

1.1.4 Edge weights

So far, while presenting edges, we have considered graphs where connections between nodes represented binary relations between them; they either existed or they did not. In many situations when studying graphs, it is useful to represent edges as connections which carry some kind of attribute or value, commonly called *weight*. This weight could be any real number that fits a particular example, such as the distance between two airfields on an airline network, the kinship of connections on a social network (negative values can represent animosity and vice versa) or any type of relational attribute that can be quantified and characterizes the connection between nodes that belong in the same network[5, p. 109]. A simple example is presented in the figure below.



(A) Multigraph with no loops and multiple edges.

$$A = \begin{pmatrix} 0 & 4 & 0.5 \\ 4 & 0 & 1.2 \\ 0.5 & 1.2 & 0 \end{pmatrix}$$

(B) Corresponding adjacency matrix.

FIGURE 1.3: Simple example of an unordered graph with weighted edges

Generally, edges and nodes can hold any type of variable as values, such as vectors, the usefulness of which will become apparent when computer algorithms for graph representations and graph neural networks are discussed in later sections and chapters.

1.1.5 Distance between nodes and shortest paths

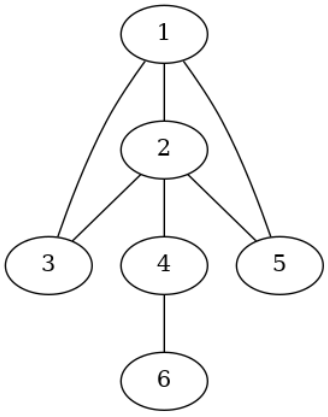
On a graph, any route that traverses nodes along the edges connecting them creates a sequence which is called a *walk*. Walks are not prohibited from traversing previously visited nodes and edges, but walks that do not intersect themselves are called *paths*.

Remark 1.5. *Adjacency Matrix Powers* Before continuing, it is worth mentioning that the powers of the adjacency matrix A^c directly provide the number of walks of length c among two nodes. If there is a connection between two nodes i and j, then $A_{ij} = 1$ else it is 0. Moving to the second power and taking for example an intermediate node k which might lie between i and j, the product $A_{ik}kj$ would be 1 if there is a node and 0 if there is not [5, p. 131]. Generalizing to walks of length c which traverse nodes i to j, their total number is:

$$N_{ij}^{(c)} = [A^c]_{ij}$$

The *shortest path* between two nodes is the shortest walk between those two nodes, the walk which traverses the least amount of nodes. In terms of edges, the least number that must be traversed is called *shortest distance* or often just "distance". Mathematically, the shortest distance between two nodes i and j is the walk with the smallest value c where $[A^c]_{ij} > 0$

Example 1.2. *For instance consider the following graph:*



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

FIGURE 1.4: A graph with a maximum path of 3 (nodes 1 to 6).

In this example, it can be visiually determined that the minimum walk between nodes 1 and 6 is of length 3. Indeed, raising the adjacency matrix to a power of three yields for these nodes:

$$\dots$$

$$N_{1,6}^{(2)} = \sum^{n} A_{1k}A_{k6} = [A^2]_{1,6} = 0$$

$$N_{1,6}^{(3)} = \sum^{n} A_{1k}A_{kl}A_{l6} = [A^3]_{1,6} = 1$$

1.1.6 Node and edge properties

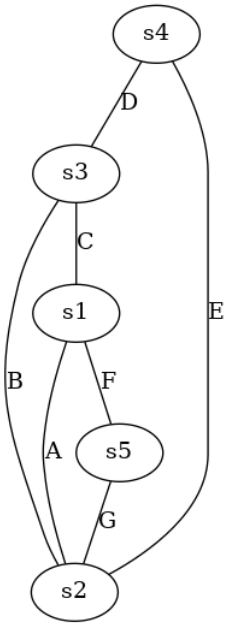
As mentioned in Section 1.1.4, edges can hold more information than just the binary relations between nodes. In fact, this concept can be generalized to nodes and even whole graphs. When studying graphs and networks, especially when using computational methods for applications like complex dynamics, it is the most natural way to phrase data on a graph. The data can be any real number or even categorical data, such as colors.

The matrices which hold the information mentioned before are

- **V: vertex (or node) attributes** (i.e. a label or number of neighbors)
- **E: edge (or link) attributes** (i.e. a label or edge weight)
- **U: global (or master node) attributes** (i.e. number of nodes or number of paths of length 2)

Global attributes usually get their values as an aggregation of the attributes of the nodes and edges, and methods applied on them. For instance, a molecule might have chemical elements as node attributes, types of bonds as edge attributes and the toxicity of the molecule as graph attributes. An example of a classroom should better illustrate the concept.

Example 1.3. *In this example, a classroom of 5 classmates is presented. Each student has a number of* ***node*** *attributes, their age, heigh and average grade (from F to A).* ***Edge*** *attributes between students hold information about their physical proximity when seated for class, and their kinship (as a real number between* $0 - 1$*). Finally,* ***global*** *attributes consist of information about the classroom, such as total number of students and class's failure rate.*



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

FIGURE 1.5: Example graph of a small classroom with labeled edges and nodes

Node	Age	Height	Grade
s1	11.5	135	C
s2	12	140	B
s3	12	142	A
s4	11.5	132	A
s5	12	143	B

(A)

Edge	Distance	Kinship
A	1.5	0.5
B	1.2	0.5
C	0.5	0.8
D	2	0.35
E	2	0.3
F	0.5	0.9

(B)

TABLE 1.4: A) Node properties B) Edge properties

	No. Nodes	Failure %
G	5	0

TABLE 1.5: Graph Properties

1.2 Network Dynamics

Frequently, it is useful to consider cases where the status of the networks studied changes over time. This could mean that the topology of the network changes (i.e. nodes and edges are added or removed), the internal state of the network changes (the properties discussed in the previous section) or both.

The main concern of this thesis is with networks with a fixed topology , but with elements (nodes and edges) whose properties constitute dynamical quantities which can change over time. Following some dynamical rule, nodes can interact with their neighbors or change their own properties, with edges dictating which interactions are possible. In fact, the study of network dynamics combines graph theory with non-linear dynamics [7].

For many real world situations, a proper model of their processes consists of dynamical systems acting on networks. This can range from opinion spreading, epidemics, flow of electricity on grids, spread of packages on internet networks and many more systems whose dynamics are evolving on a network. In fact, many network processes can be understood

Chapter 2

Graph Neural Networks

2.1 Background - Artificial Neural Networks

2.1.1 Introduction

2.1.1.1 Historical Background

Artificial Neural Networks (ANN), or sometimes simply called **neural networks** is a class of computational models that mimic the way biological neural networks work, such as the human brain. Interest on the subject sparked after the seminal paper *"A Logical Calculus of the Ideas Immanent in Nervous Activity"* [8] by Warren McCulloch and Walter Pitts, where they proposed a computationally functional model of neural networks. Their suggestions showed that in principle, any function a digital computer can compute, a neural net should too. The models they described had weights and thresholds, but they lacked a training method.

The suggestions of McCulloch and Pitts lead Frank Rosenblatt to create the *Perceptron* in 1958 [9], a binary classifier algorithm based on supervised learning[1]. Although initially promising, single layer perceptrons were not able to train on multiple classes of patterns and were eventually proven incapable of learning a XOR function [2] in the book *Perceptrons* [10], as the way they worked was by "separating" data linearly. This lead to a stagnation in machine learning research dubbed "AI winter", until the proposal of **backpropagation**[3] by Paul John Werbos in 1975 [11].

[1] Supervised learning is a machine learning training technique that optimizes a model based on examples input-output pairs.

[2] XOR (Exclusive or, $x \oplus y$) is a logical operation that is true only if its arguments differ.

[3] Backpropagation is a method of fine tuning a neural network based on the error rate obtained from previous runs of the program. It will be discussed in detail later in this thesis.

A renewed interest in the field lead to the development of the Cresceptron [12] in 1992, a method of training large networks with pooling layers (**max-pooling**) and down-sampling. GPU[4] usage made possible the training of larger networks, while new types of networks emerged such as the **Recurrent Neural Networks (RNNs)**. **Convolutional Neural Networks** have recently proven to be far superior for image classification tasks.

In recent years, neural

2.1.2 Basics

2.1.2.1 Summary

One can think of ANNs as a directed graph, with a collection of nodes which are densely connected (called **artificial neurons**), transimiting signals to each other. These nodes are usually organized in sets of layers, with signals moving in one direction (i.e. *feed forward networks*) through weighted connections. Signals received on a single neuron are real numbers, and the output of a single neuron is the output of an aggregation of a non-linear function of the sum of its inputs. This function is called an *activation function* and its results are propagated to all of the nodes outgoing connections. Thus, each neuron can be thought as a simple processing unit. The weighted connections between nodes might have an excitatory or inhibitory effect, based on these weights which can be positive, negative or very close to zero, having no effect [13, Chap. 1]. While training, example data is passed through the input layer and gets radically transformed through the layers until it reaches the output layer. The weights and other trainable parameters are then adjusted until the training data consistently reaches satisfactory results.

[4]GPU - Graphics Processing Units is a specialized electronic circuit, a central part of modern computers which excels in efficient computation of algorithms which process large blocks of data parallelly, thus exceling in machine learning applications.

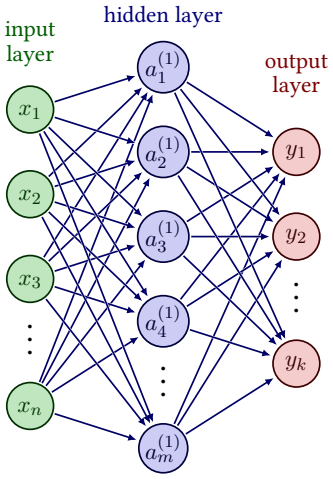


FIGURE 2.1: A simple neural network demonstrating the layered structure and and flow of data from input to output.

2.1.2.2 Building Blocks

As mentioned before, the building blocks of ANNs are its artificial neurons organized into layers. In Figure 2.1 a basic ANN is presented, demonstrating the different layers that are typically present in a feedforward neural network [5] while the building blocks are widely considered [14] to be:

- **Input layer** This layer's purpose is to act as an entrypoint to the neural network and performs no computations. Data moves from this layer to the hidden layer succeeding it.
- **Hidden layer** Networks typically have one or more hidden layers, in which some computation takes place. Data moves from the input layer to a hidden layer where it is transformed and together with its weights moves to the next hidden layer or the output layer.
- **Output layer** Transformed data "exits" the network here, where it can pass through some function to reach the desired output format
- **Edges and Weights** Each node in a layer is connected to a set (usually all) of the nodes of the following layer with a weighted edge. Signals from node i of layer k will be the input of node j of layer l, multiplied by a weight W_{ij}.
- **Activation Functions** An activation function takes as input some form of aggregate (usually the weighted sum) of the signals arriving at a node and produces an output.

[5] Feedforward Neural Networks (FNNs) are the simplest type of neural networks, with the information moving only in one direction ("forward" through the layers), without any loops or cycles [14]. Different types of neural networks are discussed later.

This function is typically nonlinear and differentiable for reasons which will be discussed later.

- **Learning** The learning process in a ANN involves modifying its weights and other learnable parameters to improve the accuracy of the result on the output layer. Learning usually involves a cost function which is evaluated on a predefined basis and adjustments are made accordingly. One of the most widespread learning techniques is *backpropagation*, where the error is propagated backwards through the network.

Along with the data from previous layers aggregated at a neuron, a bias is typically added which acts in the same way the intercept does in a linear equation. It adjusts the output of the activation function along with the weighted sum of the inputs to the neuron. It is also a trainable constant value provided to each node of a layer. Biases are node level parameters and do not depend on values provided by previous layers.

All of the computations that take place in layer can be represented in a compact matrix form, as shown below.

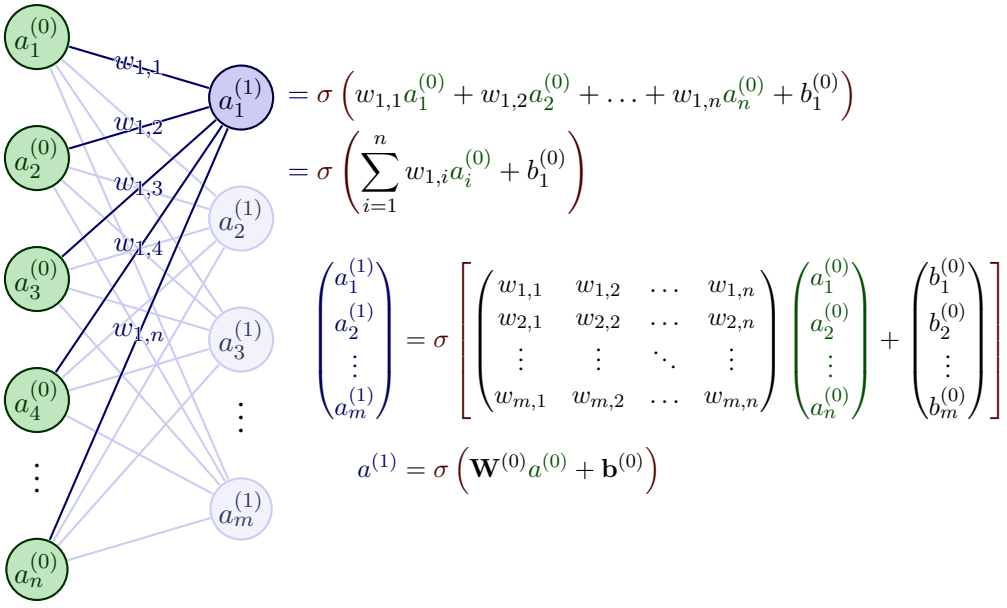


FIGURE 2.2: Input to a single neuron in a feedforward network. Here σ represents the activation function (a sigmoid function in this case) and the exponents represent layers. The activation of a layer can be conveniently represented in matrix form. Biases are added to the input of the node.

2.1.2.3 Feedforward Networks

Feedforward neural networks (FNNs) are the exemplary of ANNs, and the first to be conceived and created by Rosenblatt in 1958 with the creation of the perceptron [9]. Their goal is to approximate some function f^*; i.e. a classifier uses the function $y = f^*(\boldsymbol{x})$ to map the input

$\boldsymbol{x}$ to some category y. The goal of a feedforward network would then be to define a mapping $\boldsymbol{y} = f(\boldsymbol{x};\ \boldsymbol{\theta})$ and train in a way that the values of the parameter vector $\boldsymbol{\theta}$ provide the best approximation of the function f^*.

These networks are called feedforward as information flows from the input layer $\boldsymbol{x}$ through intermediate computational layers used to define $\boldsymbol{f}$ and finally to the output $\boldsymbol{y}$. There are no loops providing (called **feedback connections**) information from the output back into the input or other intermediate layers of the network.

Feedforward networks can be thought as a chain of functions, composing the final structure of the network. As an example, consider a network with three of these functions as $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$. In this case, the exponents denotes the layer, with $f^{(1)}$ beign the first layer, $f^{(2)}$ the second etc. The total number of these functions is called the **depth** of the NN and the terminology "deep learning" arose from this layered structure.

During training the goal is to modify the parameters of the network in a way that $f(\boldsymbol{x})$ closely matches $f^*(\boldsymbol{x})$. The training set is composed of pairs of $\boldsymbol{x}$ and labels $y \approx f^*(\boldsymbol{x})$ and the output of the network is evaluated at different training points. Training data defines the exact result expected from each input $\boldsymbol{x}$, a value as close as possible to y. The rest of the layers can have arbitary behaviours as long as they transform the data in a way defined by the training goal. The learning algorithm can "use" them in any way that is useful to it, and the training data has no immediate effect on their behaviour. They are thus called **"hidden layers"** as they do not produce any meaningful result, related to the function [15, p. 160].

2.1.2.4 Perceptrons, learning algorithms and simple NNs

The perceptron serves as great precursory example to neural networks, as it introduces many concepts that are common in more complex NN paradigms. Its in fact a simple neuron, a computational unit, which takes a binary vector as input and produces a binary output.

Inputs are multiplied with *weights*, and the output of the neuron is either 0 or 1, determined by whether the weighted sum of its inputs $\sum_i w_i a_i$ is greater than a threshold value $threshold$. All of these numbers are real numbers and a parameter of the neuron itself. The algebraic form of this can be written as:

$$\text{Output} = \begin{cases} 0 & \text{if } \sum_i w_i a_i \leq \text{threshold} \\ 1 & \text{if } \sum_i w_i a_i > \text{threshold} \end{cases}$$

By modifying the values of the threshold, the output can be changed. The above equation can be written in a more compact form by replacing the sums with the dot product of the weights

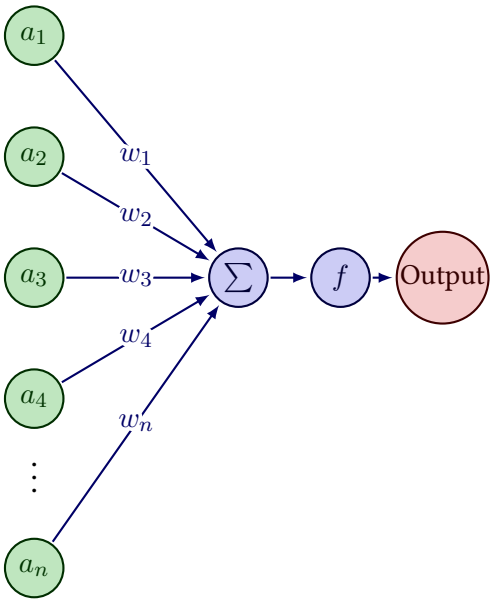


FIGURE 2.3: A simple perceptron

and the input vector, $\sum_i w_i a_i = \boldsymbol{w} \cdot \boldsymbol{a}$. Moving the threshold to the left side of the equation and replacing it by what is referred to as the *bias* yields:

$$\text{Output} = \begin{cases} 0 & if \boldsymbol{w} \cdot \boldsymbol{a} + b \leq 0 \\ 1 & if \boldsymbol{w} \cdot \boldsymbol{a} + b > 0 \end{cases}$$

where b = - threshold. The bias can be seen as a measure of the tendency of the neuron to fire. Larger bias's numbers makes the neuron easy to activate and output 1, while smaller ones require larger inputs and positive weights.

Remark 2.1. As mentioned before, perceptrons are able to solve problems which are linearly seperatable with the slope represented by the $\boldsymbol{w} \cdot \boldsymbol{a}$ term and the bias acting as the intercept. This excludes problems which are not, the most famous being the classic XOR gate, as there does not exist one single line capable of seperating the predictions.

Multiple perceptrons can be combined in a network to compute any logical function, including the XOR gate. These kind of NNs are called *multilayer perceptrons*, and they are the basis modern artificial neural networks. As discussed before, these neuros (or nodes) are organized in layers and can have a depth based on the number of these layers. Typically, at least 3 layers are present.

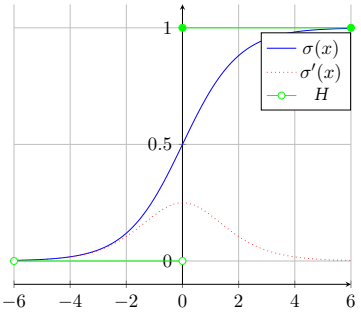
Learning Algorithms

So far, the way perceptrons transform data through a function has been described, but while they can produce sensible results as any other computing device, weights and biases had to be manually configured. It is possible to introduce a learning algorithm which does the tuning of these parameters automatically, in response to input-output (training data) pairs.

The perceptrons described used the Heaviside step function [**?**], often denoted with H. This function has a binary nature, with its output value being 0 or 1 based on a threshold, which presents a problem when fine tuning a perceptron or a NN based on them. Minor changes in a weight or bias value can completely alter the output trigger a perceptron to flip, and possibly changing the output of the whole network. A better choice for an activation function is the *sigmoid function* σ also called the *logistic function.*

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

(A) Mathematical definition of the sigmoid function.



(B) Plot of the sigmoid function, its first derivative and the Heaviside step function.

2.2 Mathematics of Artificial Neural Networks

2.2.1 Gradient Descent

Gradient based optimizations methods are of great importance to NNs as they are most common method of training these models. They are tasked with minimizing or maximizing some function $f(\boldsymbol{x})$ which is oftenly called **objective function**. In NN training scenarios, where the goal is to minimize it the same function is also commonly called a **cost**, **loss** or **error** function[15].

Gradient descent is a method of minimizing a function $f(x)$ and finding a local minimum, given that its differentiable. The derivative $f'(x)$ is the slope of $f(x)$ at x, so it specifies how a change in x would provide a step towards the local minimum. For instance, for small values of ϵ, $f(x - \epsilon \operatorname{sgn}(f'(x)))$[6] is less than $f(x)$ and traversing the slope to ever decreasing values

[6] sgn is the *signum function*, a piecewise function which returns the sign of its input or 0 if input is 0. (i.e. $\operatorname{sgn}(-1) = -1$ and $\operatorname{sgn}(12) = 1$).

if possible through following the direction with the opposite sign of the derivate. This method is called **gradient descent** and it was first proposed by Cauchy [16] in 1847.

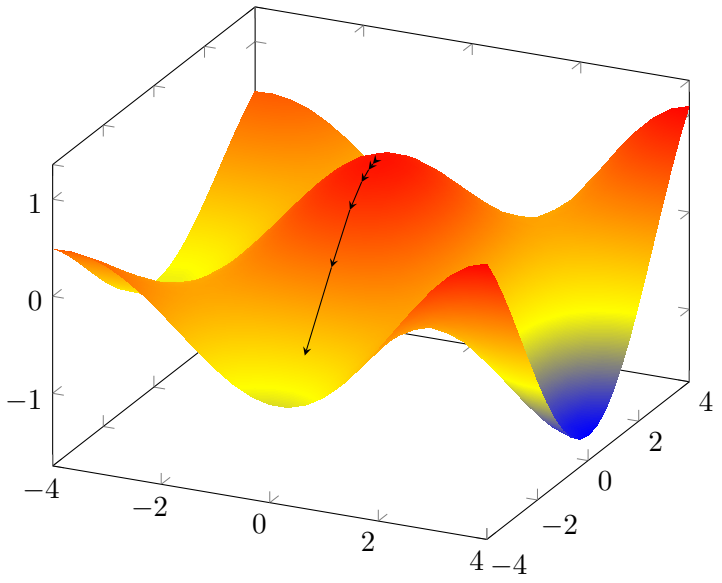


FIGURE 2.5: Gradient descent in three dimensional space

Definition 2.2. Consider a multi-variable function $F(\boldsymbol{x})$ which is **defined** and **differentiable** in a neighborhood of a point a. The value of $F(\boldsymbol{x})$ will decrease the fastest when moving from a towards the negative gradient direction of F, which is $-\boldsymbol{\nabla} F(\boldsymbol{a})$. Thus when:

$$\boldsymbol{a}_{n+1} = \boldsymbol{a}_n - \gamma \nabla F(\boldsymbol{a}_n) \tag{2.2}$$

then $F(\boldsymbol{a}_n) \geq F(\boldsymbol{a}_{n+1})$ and therefore the values of $F(\boldsymbol{a})$ move towards the local minimum. A sequence $\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, ...\boldsymbol{x_m}$ that follows the rule set by Equation 2.2 will lead to the monotic sequence $F(\boldsymbol{x}_0) \geq F(\boldsymbol{x}_1) \geq F(\boldsymbol{x}_2), ...$ and the sequence $\boldsymbol{x_n}$ will converge to the local minimum. If some special choices are made for γ[7] the function is guaranteed to reach a local minimum. Additionaly, if the function is *convex* local minima are global minima and gradient descent converges to a global solution.

For a function to be convex, a line segment connecting two of its points must lay on or above its curve. Mathematically for two points x_1 and x_2, this can be expressed as

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \tag{2.3}$$

where λ is a location on a section line and $0 \leq \lambda \leq 1$.

[7] γ in machine learning is also known as the "learning rate" and its one of the hypermateres of NNs. Special choices made here include a selection of γ via line search which satisfies the Wolfe conditions or the Barzilai-Borwein method [17]

2.2.2 Stochastic Gradient Descent (SGD)

2.2.3 Types of the most common ANNs

Common FNNs

Single-layer Perceptron

2.2.4 Training

2.2.5 Convolutional Neural Networks

Chapter 3

Notation & Fundamentals

Chapter 4

Basic Principles and Implementation Framework for an [Problem to be Solved]

Chapter 5

Implementation and Core Components of [Platform Title]

Chapter 6

Experimentation & Validation

Chapter 7

Conclusions & Future Work

Bibliography

[1] U.S.R. Murty Adrian Bondy. *Graph theory*. Graduate texts in mathematics 244. Springer, 3rd corrected printing. edition, 2008. ISBN 1846289696; 9781846289699.

[2] Bela Bollobas and Endre Szemerédi. Girth of sparse graphs. *Journal of Graph Theory*, 39: 194 – 200, 01 2002. doi: 10.1002/jgt.10023.

[3] Ping Zhang Gary Chartrand. *A First Course in Graph Theory*. Dover Books on Mathematics. Dover Publications, 2012. ISBN 0486483681; 9780486483689.

[4] A. Ashikhmin and A. Barg. *Algebraic Coding Theory and Information Theory: DIMACS Workshop, Algebraic Coding Theory and Information Theory, December 15-18, 2003, Rutgers University, Piscataway, New Jersey*. DIMACS series in discrete mathematics and theoretical computer science. American Mathematical Soc. ISBN 9780821871102. URL https://books.google.gr/books?id=wp7XsCAm9EC.

[5] Mark Newman. *Networks*. Oxford University Press, 2 edition, 2018. ISBN 0198805098; 9780198805090.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. 06 2016.

[7] Jürgen Jost. *Dynamical Networks*, pages 35–62. Springer London, London, 2007. ISBN 978-1-84628-780-0. doi: 10.1007/978-1-84628-780-0_3. URL https://doi.org/10.1007/978-1-84628-780-03.

[8] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943. doi: 10.1007/bf02478259.

[9] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[10] M. Minsky and S. Papert. *Perceptrons; an Introduction to Computational Geometry*. MIT Press, 1969. ISBN 9780262630221. URL https://books.google.gr/books?id=Ow1OAQAAIAAJ.

[11] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975. URL https://books.google.gr/books?id=z81XmgEACAAJ.

[12] J. Weng, N. Ahuja, and T.S. Huang. Cresceptron: a self-organizing neural network which grows adaptively. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 576–581 vol.1, 1992. doi: 10.1109/IJCNN.1992.287150.

[13] Kevin N. Gurney. *An introduction to neural networks*. 1997.

[14] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2014.09.003. URL https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *C. R. Acad. Sci*, 25:536–538, 1847.

[17] Xiao Qi Yang Liqun Qi, Kok Lay Teo. *Optimization and control with applications*. Applied Optimization. Springer, 1 edition, 2005. ISBN 9780387242545; 0387242546.