

Department of Physics

Section of Electronic Physics and Systems

Master’s Degree on Control and Computing

**Thesis**

Playing tower defense

against Reinforcement Learning Agents

Maria Manolaki

(RN: 2015511)

Athens

November 2020

Contents

[1 Introduction 1](#_Toc56338283)

[2 Machine Learning 1](#_Toc56338284)

[2.1 Types of Machine Learning 1](#_Toc56338285)

[2.1.1 Supervised Learning 2](#_Toc56338286)

[2.1.2 Unsupervised Learning 3](#_Toc56338287)

[2.1.3 Reinforcement Learning 4](#_Toc56338288)

[3 Neural Networks 6](#_Toc56338289)

[3.1 Introduction to Neural Networks 6](#_Toc56338290)

[3.1.1 Learning Process of Artificial Neural Network with Backpropagation 11](#_Toc56338291)

[3.1.2 Parameters 12](#_Toc56338292)

[3.1.3 Hyperparameters 12](#_Toc56338293)

[3.2 Architectures of Neural Networks 14](#_Toc56338294)

[3.2.1 Feed Forward 14](#_Toc56338295)

[3.2.2 Recurrent Neural Network 15](#_Toc56338296)

[3.2.3 LSTM 16](#_Toc56338297)

[3.2.4 GRU 18](#_Toc56338298)

[3.2.5 Convolutional Neural Network (CNN) 20](#_Toc56338299)

[4 Reinforcement Learning 21](#_Toc56338300)

[5 Reinforcement Learning and Neural Networks 21](#_Toc56338301)

[6 Tools 21](#_Toc56338302)

[7 The Game 21](#_Toc56338303)

[7.1 Tower Defense 21](#_Toc56338304)

[7.1 Thesis’ Game description 22](#_Toc56338305)

[8 Game - RL and NN Synthesis 23](#_Toc56338306)

[9 Conclusions 23](#_Toc56338307)

[Appendix 1 Game Code 23](#_Toc56338308)

[Appendix 2 RL Code (A3C) 23](#_Toc56338309)

[Bibliography 23](#_Toc56338310)

[References 25](#_Toc56338311)

# 1 Introduction

* \* Τι θα φτιάξουμε;
  + \*\* Σύστημα Μηχανικής Μάθησης
    - * - αλληλοεπιδρά με τον χρήστη
      * - μαθαίνει από αυτόν
      * - Μαθαίνει προσπαθώντας να τον νικήσει
  + \*\* Μορφή
    - * - Δημιουργικός τρόπος αλληλεπίδρασης: Παιχνίδια
      * - Παιχνίδι, το οποίο προσαρμόζεται στον τρόπο παιξίματος του χρήστη
      * - Σκοπός: δεν είναι να προσαρμόζεται στο επίπεδο του χρήστη, αλλά να τον νικάει (άρα δεν υπάρχει "ταβάνι" στο πόσο μπορεί να προχωρήσει)
      * - (μπορείς να έχεις μετρική, π.χ. |score-σταθ\*χρόνο| για να διατηρείς το παιχνίδι ενδιαφέρον -- δεν είναι αυτή η προσέγγιση)

# 2 Machine Learning

Machine Learning (ML) is a subset of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience and to become more accurate at predicting outcomes without being explicitly programmed. It relies on underlying hypothesis of creating the model and tries to improve it by fitting more data into the model over time. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly. ML is applied in many areas, but it is mostly significant in data mining.

**Formal Definition:** A machine is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P** if its performance at tasks in **T**, as measured by **P**, improves with experience **E**. [1]

## 2.1 Types of Machine Learning

Machine learning approaches can be classified into 3 broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

* Supervised machine learning
* Unsupervised machine learning
* Reinforcement learning

Figure 1 demonstrates machine learning types combined with their main applications.

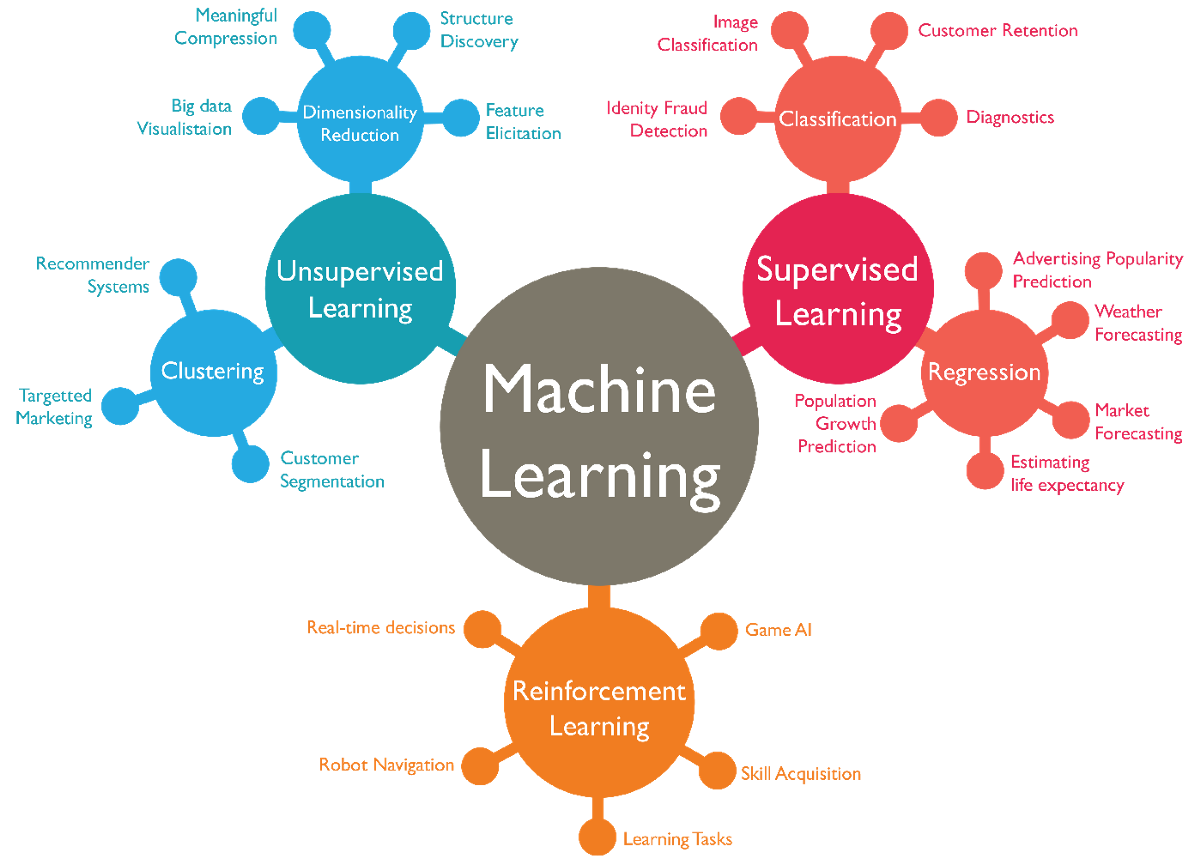


Figure . Diagram of Machine Learning subcategories **[[1]](#footnote-1)**

### 2.1.1 Supervised Learning

Supervised machine learning requires a training data and a human supervisor. The training data is an example of input-output pairs, each of which consists of data sample (typically a vector) used to make prediction and expected outcome called label or the supervisory signal. The human supervisor is necessary to assign the labels to the pairs. While training a supervised learning algorithm, data is searched for pattern that correlate with the desired outputs. After training, the algorithm takes in new unseen inputs and is going to determine which label the new inputs will be classified as based on prior training data. The objective of a supervised learning model is to predict the correct label for newly presented input data.

Supervised learning can be additionally categorized into: Classification and Regression.

**Classification** is the process of predicting the class of given data points. Another name of classes is targets/ labels or categories. The training data is a set of samples from a class-labelled data set for which their class labels are already known. Classifiers themselves can be divided in two groups based on the number of classes that they work with:

* Binary classifiers: only two classes or possible outcomes (e.g. spam mail detection)
* Multiclass classifiers: multiple classes (e.g. whether an image is apple or orange or banana)

**Regression** analysis consists of a set of samples as training data ,each of which is a real-value is a form of predictive modelling technique which investigates the relationship between a dependent variable, target/output signal which is a real-valued response, and independent real-valued variable, predictor. This technique can be used in various areas such as: Forecasting or Predictive analysis, Optimization, Error correction, Economics, Finance.

Typically, there are 3 types of Regression: Linear Regression, Non-Linear Regression, Logistic Regression. In the first one, the target is to determine the slope and the constant term (intercept) of the line that fits best the data. Similarly, in the non-linear case the target is to determine the characteristics of a curve that best fits the data e.g. Polynomial regression. Finally, the logistic regression uses a logistic function, which will be analyzed later, to model the probabilities and has two modes: the binary and multinomial and is widely used in classification problems.

### 2.1.2 Unsupervised Learning

Unsupervised learning is a process where class labels are not provided for each sample, neither is the structure of the data. The training data is of a set of input vectors only i.e. there are no corresponding labels. The goal may be to discover groups of similar examples within the data, this process is called *clustering*. Another case is to reduce the dimensions of the dataset; for example, the number of columns to be reduced, or a sphere-shaped data to be converted to circle. This process is called *dimensionality reduction*.

**Clustering** ‘s goal is to discover a structure in a collection of raw data (without any label). It could be described as the process of “organizing objects into groups whose members are similar in some way” [2]. A set of points, with a notion of distance between the points, are grouping into a number of clusters with the following rules:

• Internal distances should be small (members of the same cluster)

• External distances should be large (members of different clusters)

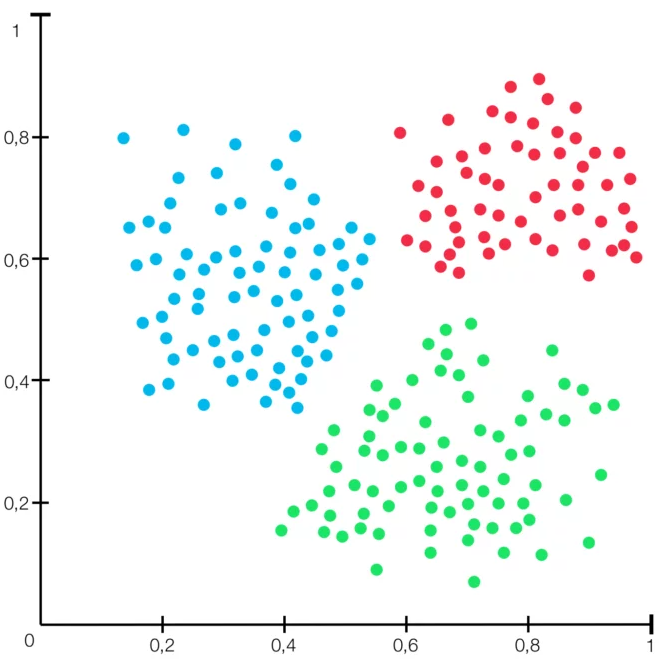


Figure . Clustering [[2]](#footnote-2)

In **dimensionality** **reduction**, the goal is to reduce the dimensions of a feature set. If the training of a machine learning model is based on many features, this model becomes dependent on the data used for training (overfitting), and in many cases results in low performance when it is applied on real data. Dimensionality reduction is an approach towards eliminating overfitting effect, improving model accuracy, minimizing computational burden and storage, and giving possibility to use algorithms which could not be used for large dimensions.

### 2.1.3 Reinforcement Learning

Reinforcement Learning (RL) deals with learning via acting and getting rewards as feedback. Two basic notions are the *agent*(s) and the *environment*, whereas the feedback is performed via *rewards* and punishments (negative rewards). An agent can perceive and interpret its environment, and can take actions and interact with it. The goal is to find a suitable action model that would maximize the total cumulative reward of the agent. The action-reward feedback loop of a generic RL model is demonstrated in the following figure.

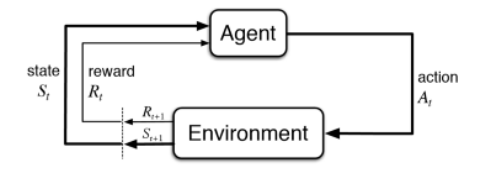


Figure . Action-Reward Feedback loop[[3]](#footnote-3)

The basic elements of a Reinforcement Learning algorithm are:

* Environment: the world, real or virtual, where an agent is trained how to make correct decisions for his actions.
* Agent: an entity that learns and makes decisions.
* Action: a status change in the environment caused by the agent.
* Reward: the evaluation of an action, positive or negative.

Some other important parts of this technique are:

* Policy: a function that makes the decisions of the agent, which actually maps the agent’s situations to actions.
* State: the state (situation) of the agent in the environment
* Value function: a function that returns a real number corresponding to a specific state after executing a particular policy. The returned value is used as the long-term reward.
* Model: how the agent perceives the environment, i.e. a map of state-action pairs to probability distributions. Not all RL agents use models of their environment.

Reinforcement algorithms can be applied in multiple areas such as:

* Resources management in computer clusters
* Personalized Recommendations
* Bidding and Advertising
* Games
* Deep Learning
* Traffic Light Control
* Robotics
* Web System Configuration
* Chemistry

The key difference between reinforcement and supervised learning lies in the feedback provided to the agent. In supervised learning the feedback is a set of actions to perform correctly a task, whereas in RL rewards and punishments are used to train the agent behavior.

In comparison to unsupervised learning, RL differs in the goals to be achieved. In unsupervised learning the goal is to find differences and similarities in dataset points, while in RL the aim is to search for an action model which would lead to maximum total cumulative reward.

# 3 Neural Networks

## 3.1 Introduction to Neural Networks

Artificial Neural Networks (ANN) or simply Neural Networks (NN) were introduced in 1943 by Warren McCullough and Walter Pitts [3] who proposed a model, called threshold logic, for neural networks based on mathematics and algorithms. Later in 1958 Rosenblatt developed the perceptron, a pattern recognition supervised learning algorithm, that was using two-layer network [4]. The research on NN slowed down after 1969 when Marvin Minsky and Seymour Papert discovered the limitations due to lack of sufficient computational power at that era demanded by large NN which would lead to very long run time [5]. In the mid of 70’s the interest for NN reflated due to both the development of computers with much higher processing ability and the effectiveness of the proposed backpropagation algorithm [6]. After a decade of low interest, a forceful comeback is taking place the last ten years mainly because of the increased processing power provided by graphics chips.

A NN is a network of artificial neurons which simulates the functionality of human brain. Their ordinary use is in clustering and classification. They present notable ability to cope with complex and raw data and conclude a meaning from them which is very useful for pattern recognition. They can derive trends from this data which cannot be discovered by other computer methods or even sensed by humans. The main advantages are:

* + - * *Use as an expert*. ANN can be perceived as an expert in the information area it has been trained for. It can analyze raw data and provide predictions when new situations arise.
      * *Adaptive learning*. They present learning ability on how to perform actions based on training data.
      * *Self-Organization*. They are capable to develop their own representation of the information provided during learning period.

A typical ANN architecture includes three layers: input, hidden and output layer.

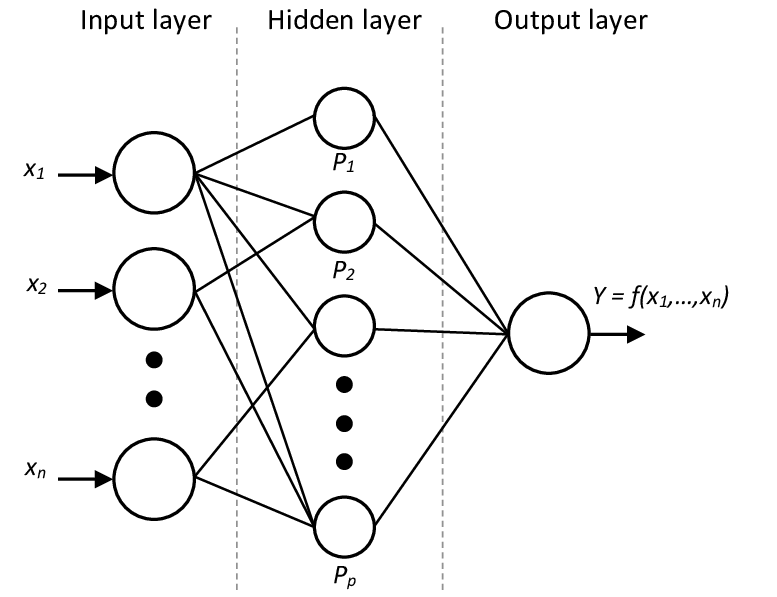


Figure . Neural Network with Input layer, Hidden layer, Output layer.[[4]](#footnote-4)

* *Input Layer*: it receives the raw information.
* *Hidden Layer*: the output of each unit (neuron or node) in this layer is a weighted combination of the inputs.
* *Output Layer*: similarly, the behavior of each unit in the output layer is a weighted combination of the outcomes of the hidden units.

Of course, there are architectures that include none or more than one hidden layer and none or more than one units in the output layers.

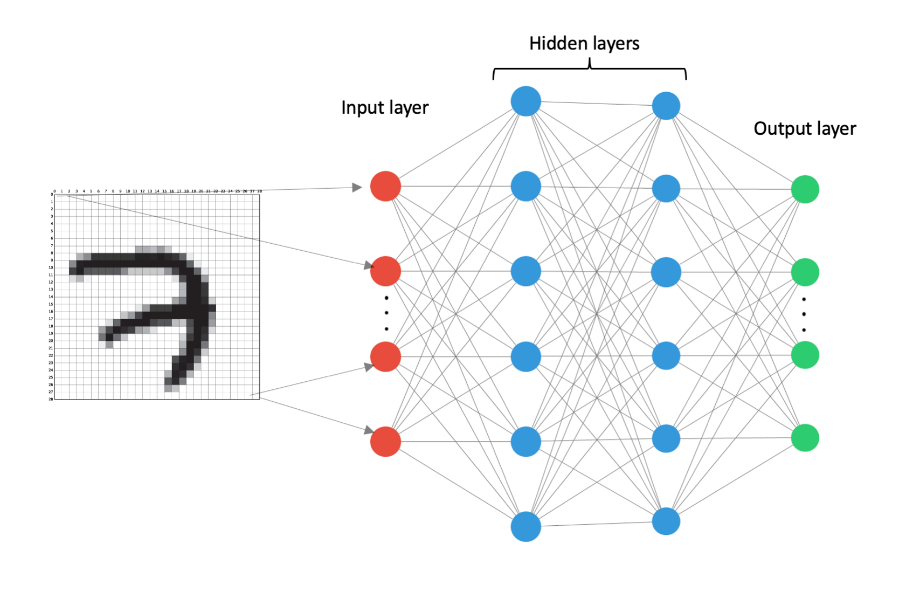


Figure . An Architecture with 2 hidden layers and n units in the output layer [[5]](#footnote-5)

The fundamental unit in a neural network is called neuron and it is represented as a node in the architecture diagrams. The neuron is fed with inputs by other nodes (in the case of hidden and output layers) or by an external source (in the case of input layer) and derives its output. Each node is connected with the other nodes with some associated weight (w) which represents the relative importance, as it can be seen in Figure 6.

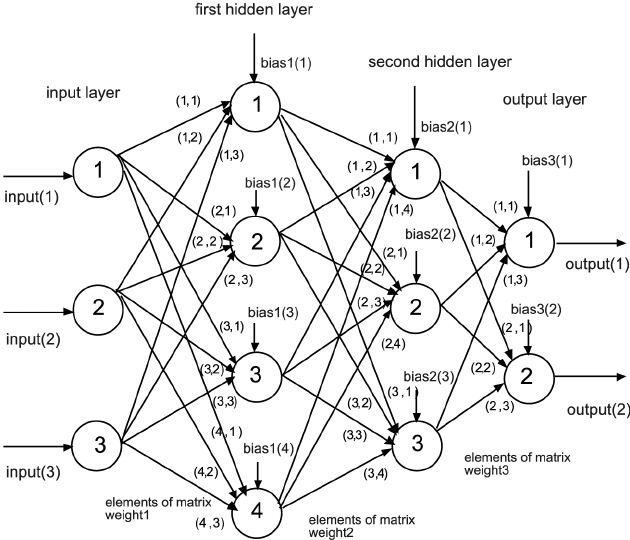


Figure . NN with 2 hidden layers and two output nodes [[6]](#footnote-6)

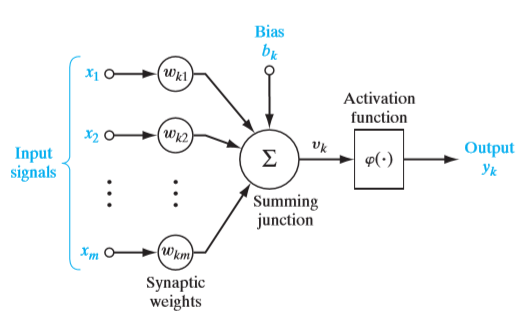


Figure . Model of a neuron [[7]](#footnote-7)

Figure 7 illustrates the model of the kth neuron in a layer. The function of the neuron is described by the following equations:

Where

xi is the jth input

wkj is the *synaptic* weight of input j of neuron k

uk is the result of the linear weighted *sum* operation

φ( ) is the *activation function* or *squashing function* as it is used to limit the amplitude of the output.

bk is the bias which performs an affine transformation to the adder output uk. Its main role is to provide an extra trainable constant value to neuron k.

Activation functions in the most common cases are non-linear. Introducing non-linearity in the neuron output, helps the neuron to learn since most real-world data are not linear.

Some of the main activation functions are shown in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | What it does | Plot [[8]](#footnote-8) | Function |
| Sigmoid\* | limits the output to the range [0, 1] |  |  |
| tanh | limits the output to the range [-1, 1] |  |  |
| ReLU\* | Rectified Linear Unit. Permits only positive values, otherwise 0 |  |  |
| ELU\* | Exponential Linear Unit. For positive values y = x, otherwise a small negative number |  |  |

\*used in this master thesis.

There are several other activation functions like Identity, Threshold, Binary Step, GELU, SELU, SQNL, Gaussian, arctan. Each activation function is designed to face specific application problems.

The selection of an appropriate activation function is part of the architecture design of an NN, and it is considered as a hyperparameter, a term that is going to be discussed in the next pages.

In Universal Approximation Theorem it is proved that any mathematical function y = f(x) can be approximated to an acceptable error magnitude by a neural network that includes one hidden layer containing a finite number of neurons under the condition that employs the appropriate activation function. George Cybenko proved it only for sigmoid activation functions in 1989 and Kurt Hornik extended the proof for all the activation functions in 1991. The key element to achieve performance is the structure of the neural network, not the type of the activation function. As a conclusion, the Universal Approximation Theorem states there is a type of universality in NNs, i.e. there is a neural network that can achieve an approximation of any given function.

### 3.1.1 Learning Process of Artificial Neural Network with Backpropagation

Training a neural network means that the values of weights wkj and biases bk must be adjusted in that way that the neural network obtains the desired behavior. This is an iterative learning process achieved by forward propagating the information of the inputs and by backpropagating the error

In **forward** **propagation** all input data pass through all neurons of the network which apply their transformation and finally the output layer will produce estimates for this specific training data. Then, a loss function is used to estimate the magnitude of the error, i.e. the difference from the desired response. Of course, the ideal cost function should be close as possible to zero. For this purpose, the synaptic weights and the bias of each neuron must gradually be tuned.

The output error will be propagated backwards (**backpropagation**). Each hidden layer neuron *k* receives only a part of the output error *ej*, based on the importance (weight, *wkj*) they had in estimation they had on the final estimate yj,, that is:

*ekj = ej \* wkj*

The total error of the hidden layer neuron *k* received backwardly from all output layer neurons is:

where *n* is the number of the output layer neurons. This information passes in the same way through all neurons starting from the output layer and going backwards layer by layer. So, all neurons will obtain backwardly a signal assessing their contribution to the final error. Then an optimization method can be applied, such as *Gradient Descent*, in order to minimize the output layer error by tuning all the network’s synaptic weights. Other optimization methods are SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax and Nadam.

Feeding again the same training data to the network, a better performance should be observed as the weights have been adjusted. Then the new and smaller error is propagated backwards again and so on. This is done iteratively in batches of data of all the dataset that is passed to the network.

Training error is the error observed in the training data, whereas test error is the error observed on the new input data. The aim of a machine learning algorithm is to make both the training error, and the difference between training error and testing error small. The term *under-fitting* means that the model cannot attain a low training error, while *over-fitting* means that the difference between training error and test error is large.

One of the techniques to minimize over-fitting is called *Dropout*. With this technique one or more units are randomly “dropped out”, i.e. omitted, during the training phase.

### 3.1.2 Parameters

Parameter is a variable internal to the model which is used to configure the NN and whose value could be changed by the data during the program iterations. More specifically these are the weights of the connections between the neurons and the biases.

***Weight initialization***: Initially the weights are set to small random values which should be different from neuron to neuron. Otherwise, if two neurons start with the same initial weight values, their values will be identical during the following iterations, which means canceling ability to learn different characteristics. Setting random values is not enough to obtain an efficient NN and generally the type of activation function should be taken into account to define the initial values heuristically.

### 3.1.3 Hyperparameters

The term hyperparameter, in contrast to parameters, refers to variables external to the model that are specified during design phase by the programmer. Such variables relate to the structure and topology of the NN (type of activation functions, number of layers, number of neurons, etc) whereas other relate to the learning algorithm (learning rate, momentum, batch size, epochs, optimization method etc.).

***Epochs***: It is the number of times (iterations) the training data pass through the NN during training phase.

***Batch size***: Usually the training data are split in batches to feed the NN. Batch size is the size of these batches

***Learning Rate***: In backpropagation with gradient descent, in order to update each synaptic weight wkj of a neuron in the network the following formula is used:

where

|  |  |
| --- | --- |
|  | Expresses the error change with respect to weight |
| a | Is the learning rate (a scalar value), otherwise called step size. |

For example, if the learning rate magnitude *a* is 0.01 and the gradient is 2, then the new weight wkj will be reduced by 0.02.

***Momentum***: It is based on the weighted average of the gradient of the previous steps. It is a method that accelerates the learning rate when its vector has the same direction with the current gradient and in other case it helps to avoid local minima of the loss function.

|  |  |
| --- | --- |
| Figure . GD with momentum [[9]](#footnote-9) | Figure . Stuck at a local minimum [[10]](#footnote-10) |

There are no simple and straight-forward methods to define the optimal values of the hyperparametres, especialy that of learning rate, momentum, batch size, type of activation function, number of layers and neurons etc. Expertise and usage of extensive trial and error are key elements for defining the optimal values of these parametres.

## 3.2 Architectures of Neural Networks

### 3.2.1 Feed Forward

The Feed Forward (FF) constitutes the first type of NN structure invented. In this network type the information proceeds only forwards as there are no circular connections.

The simplest form is called *Single-Layer Perceptron* (SLP) and consists of an input and output layer. Its representation is shown in Figure 10, where the output is derived after applying an activation/squashing function (usually a sigmoid) to the weighted sum of the inputs. An SLP is a linear classifier.

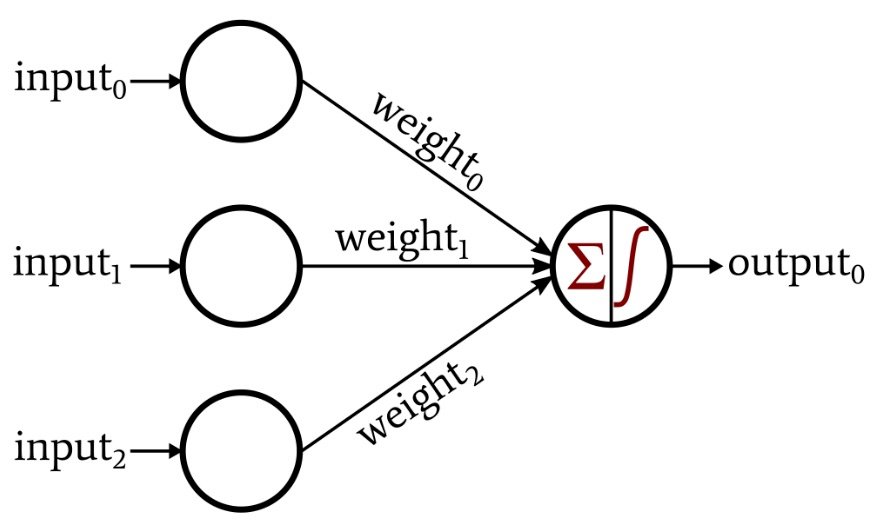


Figure . Single- Layer Perceptron [[11]](#footnote-11)

Another and more complete form of FF network is the Multi-layer Perceptron (MLP) which adds at least one hidden layer between input and output layer. MLP structure is depicted in the next figure.

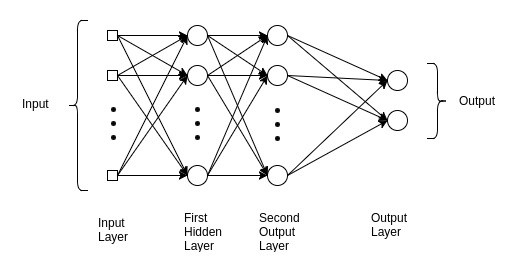


Figure . Multi- Layer Perceptron[[12]](#footnote-12)

Since there are no circles in the connections, the output is determined by the current input dataset. Past inputs do not influence the output of current data.

### 3.2.2 Recurrent Neural Network

Figure 10 shows the typical chain-like recurrent neural network (RNN) architecture:

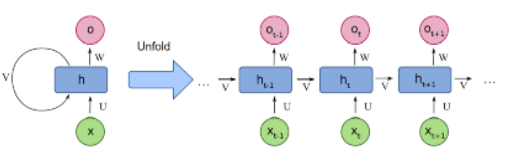


Figure . Folded and unfolded representations of an RNN[[13]](#footnote-13)

The difference from the FF is that they include a loop in the hidden layer. The consequence of this is that the output for the current input vector depends also on the output for the previous input vector. So, the RNN has *memory*. In contrast in an FF network, the output is not affected by the previous input data it handled and its memory is restricted only in things learnt during the training period.

In RNN the output is affected by both the weighted sum of the input and by a hidden “state vector” which is determined by the prior inputs. This can be described in a mathematical context as:

Where,

|  |  |
| --- | --- |
|  | denotes the input layer vector at time k |
|  | denotes the hidden layer vector at time k |
|  | denotes the output layer vector at time k |
|  | is an assisting vector |
| φ( ) | indicates the activation function (usually a sigmoid function σ ( )) |
| , | are the bias vectors |
| , , | are the  weighting matrices of the input-to-hidden connection, hidden-  to-output connection, and hidden-to-hidden connection  are the weighting matrices of the *input-to-hidden* connection, *hidden-to-output* connection, and *hidden-to-hidden* connection respectively |

There are applications that their output depends on the total input sequence (preceding and succeeding data vectors), whereas RNN’s output depends only on the past sequence so the standard RNN cannot give optimal results. A solution to this problem was proposed by M. Schuster [7] who introduced the bidirectional recurrent neural network (BRNN). BRNN employs two RNNs, one for each time direction. More specifically, the first one commences from the beginning of the data sequence and proceeds forward, whereas the second starts from the end and moves backwards. Figure 13 illustrates the BRNN architecture.

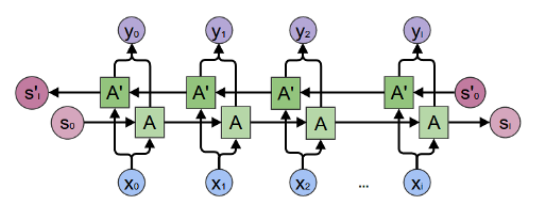


Figure . The BRNN structure [[14]](#footnote-14)

Speech, handwriting and image recognition are examples of applications that take advantage of the BRNN architecture.

### 3.2.3 LSTM

There are observed difficulties during the training of an RNN, because long term dependencies are covered by short term dependences. More specifically, for long term the gradient magnitudes frequently become smaller and smaller or rarely explode which lead to instability problems. Bengio et al. [8]. A first proposal towards eliminating or moderating this problem came from Hochreiter and Schmidhuber [9] and it is an evolution of the classic RNN called long short-term memory (LSTM). Later a new variant of the recurrent structure was proposed by Cho et. Al. [10] called Gate Recurrent Unit (GRU), which will be discussed in the subsequent subsection.

The innovation of LSTM is the introduction of a memory component called *cell*. There are three other components: the input gate, the forget gate and the output gate. The function of each component is:

* ***Cell***: Monitors the dependencies between the elements in the input data sequence.
* ***Input gate***: Defines the part of the new value which will flow into the cell.
* ***Forget gate***: Defines the part of the new value which will remain into the cell.
* ***Output gate***: Determines the part of the new value in the cell which will be used to calculate the output.

The term “new value” refers to the sum of the input and the previous output. The commonly activation function used in the LSTM gates is the logistic sigmoid function.

Figure 14 depicts the gates and the general structure of an LSTM neural network.

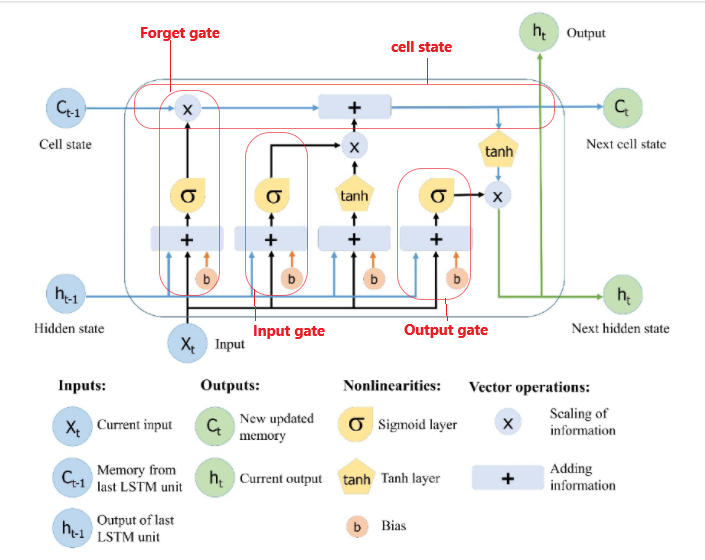


Figure . The Long Short-Term Memory structure [[15]](#footnote-15)

In this way, the LSTM earns the capability to keep learning long-term dependencies and to remember information for long periods.

Although the LSTM variant of RNN solves the problem of long-term memory dependencies, it does not solve sufficiently the exploding gradient problem.

### 3.2.4 GRU

Gated recurrent unit (GRU) is a simpler variation of the RNN similar to LSTM, but without an output gate, so it has fewer parameters. It tries also to solve the vanishing gradient problem. The main components of its structure are the update gate and the reset gate. The role of these gates (vectors) is to determine what information will be transferred to the output. More specifically:

* ***Update gate***: Determines the amount of the information coming from the past is needed to be transferred to the future
* ***Reset gate***: Defines the amount of the information coming from the past is needed to be forgotten.

Figure 15 demonstrated the architecture of the GRU.

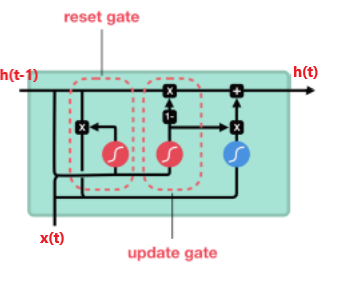




Figure . GRU structure [[16]](#footnote-16)

GRU and LSTM have similar performance on tasks like polyphonic music modeling, speech signal modeling and natural language processing, whereas GRU outperforms the latter for some smaller and rare datasets. However, GRU is not efficient in learning simple languages while LSTM is capable to learn them [11]. Moreover, LSTM units persistently exhibit better performance than GRU cells in "the first large-scale analysis of architecture variations for Neural Machine Translation” [12]

### 3.2.5 Convolutional Neural Network (CNN)

CNN is a feedforward NN with convolutional layers, pooling layers and fully connected layers, mostly applied to visual processing. They were inspired by the operation of visual cortex in human and animal brains. Their architecture follows the neurons’ connectivity pattern of this area.

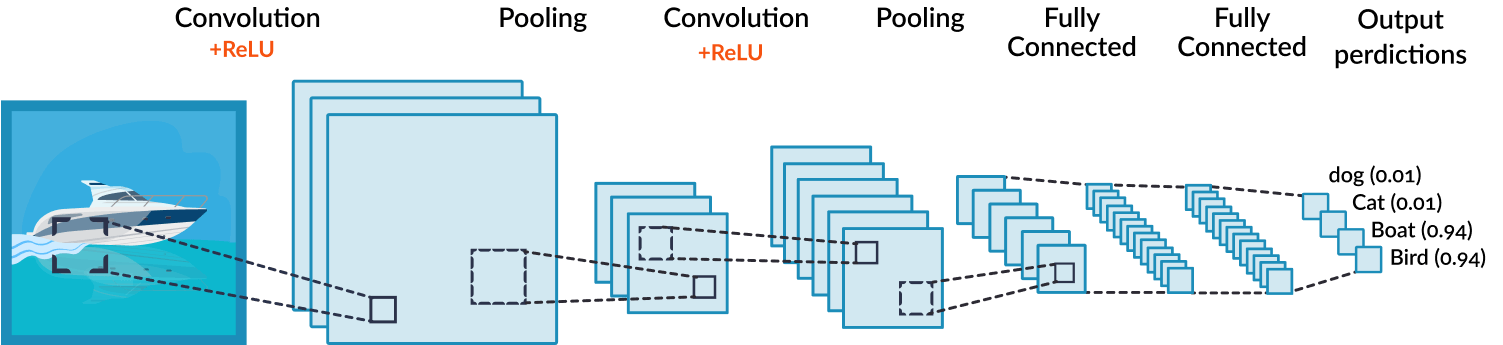


Figure . CNN architecture [[17]](#footnote-17)

* ***Convolutional layer***: Firstly, a filter is passed to view pixels each time, e.g. 4x4. The convolution is performed by a dot product of the input pixels values with the filter’s weight matrix. The result is a scalar value representing all the pixels observed by the filter. As a result, the convolution layer produces a matrix which is much smaller than the original. The activation function applied is usually ReLU.
* ***Pooling layer***: The purpose of this layer is to further reduce the size of the matrix. This is achieved by combining the outputs of the previous layer into one single node in the current layer. Pooling could be applied locally (where small clusters, e.g. 2x2, are processed) and globally (all neurons’ outputs of convolutional layer are processed at once). Furthermore, there are two types of pooling that are used: max pooling, that uses the maximum value in each cluster, and average pooling, where the average value of each cluster is selected.
* ***Fully connected layer***: Like in the typical multi-layer perceptron, all neurons in one layer are connected to all neurons in another layer. The goal of this layer is to classify the flattened matrix derived from the above-mentioned layers.

CNN are very effective in applications such: image classification, image and video recognition, medical image analysis, natural language processing.

# 4 Reinforcement Learning

It is a common perception that we learn by interacting with our environment. When an infant play, there is no explicit teacher, but it is directly connected with the environment via sensors. If for some action it receives positive feedback (reward) from the environment, it repeats this action, otherwise it stops. In this way, an abundance of information is produced about the consequences of actions and what to do in order to achieve goals. All theories of learning and intelligence consider learning from interaction as the fundamental principle.

An aim in artificial intelligence (AI) is to develop computational methods so that machines will become capable of learning from interaction with their environment, improving continuously through trial and error. Reinforcement learning (RL) is a mathematic framework for experience-driven autonomous learning.

(…………………Although RL had some successes in the past [141, 129, 62, 93], previous approaches lacked scalablity and were inherently limited to fairly low-dimensional problems. These limitations exist because RL algorithms share the same complexity issues as other algorithms: memory complexity, computational complexity, and in the case of machine learning algorithms, sample complexity [133]. What we have witnessed in recent years—the rise of deep learning, relying on the powerful function approximation and representation learning properties of deep neural networks—has provided us with new tools to overcoming these problems. …………………….)

The core principle of RL is learning *through interaction*. An agent acts, then observes the consequence of its action and learns to adjust its own behavior based on the reward/punishment it receives from the environment. The root of this trial-and-error learning approach comes from the behavioral psychology – behaviorism and constitutes one of the main foundations of RL [13]. *Optimal control* is the second key influencer on RL, which borrowed its mathematical formalism in this field.

An agent observes a state *st* of its environment at time *t*. Then the agent takes an action *at* that is affecting the environment which transitions to a new state *st+1*.

i.e. the new state derives from the current state and the action chosen.

The state condenses all the sufficient information of the environment in order for the agent to choose an optimal action. Please notice that in optimal control literature, state is denoted by **xt** and actions by **ut**. Each time the environment progresses to a new state *st+1* it also sends a reward *rt+1* to the agent as a feedback. The reward is a scalar value.

The aim of the algorithm controlling the behavior of the agent is to learn a *policy* (control strategy) *π* that will lead to maximum expected return (cumulative, discounted reward). Under this view, the problem of finding the optimal policy in RL is similar to optimal control. However, in optimal control there exists a model of the state transition dynamics which is not the case in RL, where the agent does not have such a model for the environment dynamics so it has to apply trial-and-error in order to learn about the consequence of its actions. Figure 12 presents this perception-action learning loop.

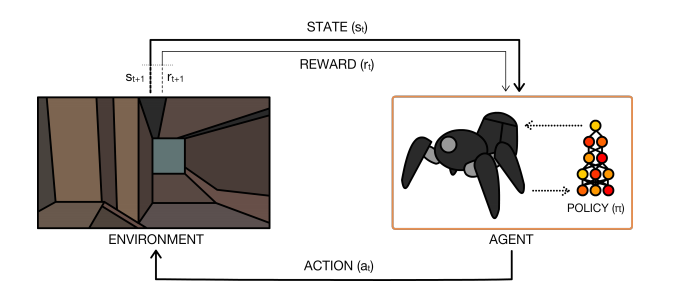


Figure . The perception-action-learning loop [[18]](#footnote-18)

What makes reinforcement learning different from other machine learning paradigms?

* There is no supervisor, only a reward signal
* Feedback is delayed, not instantaneous
* Time really matters (sequential, non i.i.d data)
* Agent’s actions affect the subsequent data it receives
  + \*\* Reinforcement Learning
    - * - RL problem statement / definition
      * - Rewards / Policies / Approaches to solving
      * - "Essentially we are trying to approximate two functions"
  + \*\* RL & NNs (deep RL)  
       <https://jonathan-hui.medium.com/rl-dqn-deep-q-network-e207751f7ae4>  
       <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>
    - * - How are NNs used in RL problems? -- (\*) Universal Apprx.
      * - DQN
      * - TD3?
      * - A3C (A2C)

# 5 Reinforcement Learning and Neural Networks

# 6 Tools

# 7 The Game

The game developed in this thesis belongs to the tower defense genre of games.

## 7.1 Tower Defense

Tower defense games is a subcategory of real-time strategy games as the player’s goal is the protection of his base against enemies by strategically placing defensive obstructions in the area that they are moving.

Usually, the enemies move in an area, through a path, trying to reach a destination (base) which is significant for the player, i.e. house, possessions, loved ones etc. They can also follow multiple paths, can be damaged and killed, and can appear in waves, each of which usually has a certain number and type of enemies.

On the other hand, the player aims to prevent the enemies from achieving their goal by stopping, attacking or destroying the enemies. For that case, the defensive obstructions, i.e. a tower, can be placed in everywhere the game area, except from the enemies’ path, and could damage the enemy. In modern tower defense games, some features are introduced in order to make the game more intriguing and engaging for the player. That could be the player’s ability to upgrade or repair the obstruction and to collect virtual money (game-currency) or points in order to purchase advanced features for defeating the enemy.

Notable examples of tower defense games are: Doom, Flash Element TD, Iron Grip: Warlord, Facebook platform’s Bloons TD.

## 7.1 Thesis’ Game description

As in any tower defense game, **the player** makes strategic decisions about the placement for the defense obstacle, which can be either a tower attacking the enemies on players request and direction, or an auto attacking tower. The latter tower selects by chance an enemy in the route and attacks at it by determining the direction of the attack based on enemies’ current movement.

The attack is performed by shooting of bullets. If a bullet hits an enemy, the enemy loses health power.

The reward of the player is determined according to the number of hit bullets and can be either an increment of the number of bullets that the auto shooting tower is firing or a slight increment of the bullet speed in on-demand shooting tower.

From **enemies**’ (and consequently the game’s) perspective, though, the ultimate goal is to reach the castle as many of them following a specific route/path and stay alive. A higher goal would be to destroy the damaging sources, to wit, the tower(s), but it has not been implemented in terms of this thesis.

The path is a zig-zagged trail that has the following hhhhhhh to start at the left up corner of the game window

.

• - Genre: Tower Defense

• - Goal description (Player)

• - player reward?

• - Goal description (Game / Enemies)

• - (High Goal) -> Destroy tower

• - (Short-term) -> Stay Alive

• - (Short-term) -> Reach the exit (castle)

• - Mechanics

• - Player places towers (player controlled)

• - Towers auto target (closest) enemies (\*) - they always succeed on dummy target

• - Enemies learn to avoid (RL)

• - Enemies can only see two closest "bullets"

• - Physics

• - General description

• - How are things modeled?

• - objects that collide

• - Game Images (Annotated)

# 8 Game - RL and NN Synthesis

# 9 Conclusions

# Appendix 1 Game Code

# Appendix 2 RL Code (A3C)

# Bibliography

|  |  |
| --- | --- |
| [1] | T. M. Mitchell, Machine Learning, 1997. |
| [2] | R. Bhardwaj, V. S. Dixit and A. Upadhyay, "A Fuzzy Intra-Clustering Approach for Load Balancing in Peer-to-Peer System," *Journal of Information and Computing Science,* vol. 7, no. 1, pp. 19-24, 2011. |
| [3] | W. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity.," *Bulletin of Mathematical Biophysics.,* vol. 5, pp. 115-133, 1943. |
| [4] | F. Rosenblatt, "The Perceptron: A Probalistic Model For Information Storage And Organization In The Brain," *Psychological Review,* p. 386–408, 1958. |
| [5] | M. Minsky and S. Papert, An Introduction to Computational Geometry, MIT Press, 1969. |
| [6] | P. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Thesis (Ph. D.)--Harvard University, 1975. |
| [7] | M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural net-works," *IEEE Transactions on Signal Processing,* vol. 45, no. 11, p. 2673–2681, 1997. |
| [8] | Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks,* vol. 5, no. 2, p. 157–166, 1994. |
| [9] | S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, p. 1735–1780, 1997. |
| [10] | K. Cho, B. v. Merrienboer, D. Bahdanau and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *In Machine Learning and Knowledge Discovery in Databases,* 2014. |
| [11] | G. Weiss, Y. Goldberg and E. Yahav, "On the Practical Computational Power of Finite Precision RNNs for Language Recognition," 2018. |
| [12] | D. Britz, A. Goldie, M.-T. Luong and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," 2018. |
| [13] | S. Mishra, "Unsupervised Learning and Data Clustering," 20 05 2017. [Online]. Available: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a. |
| [14] | J. Chung, C. C. K. Gulcehre and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.," in *NIPS : Deep Learning and Representation Learning Workshop*, 2014. |
| [15] | M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural net-works," vol. 45, no. 11, p. 2673–2681, 1997. |
| [16] | W. Feng, N. Guan, Y. Li, X. Zhang and Z. Luo, "Audio visual speech recognition with multimodal recurrent neural networks," pp. 681-688, 2017. |

# References

|  |  |
| --- | --- |
| [1] | T. M. Mitchell, Machine Learning, 1997. |
| [2] | R. Bhardwaj, V. S. Dixit and A. Upadhyay, "A Fuzzy Intra-Clustering Approach for Load Balancing in Peer-to-Peer System," *Journal of Information and Computing Science,* vol. 7, no. 1, pp. 19-24, 2011. |
| [3] | W. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity.," *Bulletin of Mathematical Biophysics.,* vol. 5, pp. 115-133, 1943. |
| [4] | F. Rosenblatt, "The Perceptron: A Probalistic Model For Information Storage And Organization In The Brain," *Psychological Review,* p. 386–408, 1958. |
| [5] | M. Minsky and S. Papert, An Introduction to Computational Geometry, MIT Press, 1969. |
| [6] | P. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Thesis (Ph. D.)--Harvard University, 1975. |
| [7] | M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural net-works," *IEEE Transactions on Signal Processing,* vol. 45, no. 11, p. 2673–2681, 1997. |
| [8] | Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks,* vol. 5, no. 2, p. 157–166, 1994. |
| [9] | S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, p. 1735–1780, 1997. |
| [10] | K. Cho, B. v. Merrienboer, D. Bahdanau and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *In Machine Learning and Knowledge Discovery in Databases,* 2014. |
| [11] | G. Weiss, Y. Goldberg and E. Yahav, "On the Practical Computational Power of Finite Precision RNNs for Language Recognition," 2018. |
| [12] | D. Britz, A. Goldie, M.-T. Luong and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," 2018. |
| [13] | S. Mishra, "Unsupervised Learning and Data Clustering," 20 05 2017. [Online]. Available: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a. |
| [14] | J. Chung, C. C. K. Gulcehre and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.," in *NIPS : Deep Learning and Representation Learning Workshop*, 2014. |
| [15] | M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural net-works," vol. 45, no. 11, p. 2673–2681, 1997. |
| [16] | W. Feng, N. Guan, Y. Li, X. Zhang and Z. Luo, "Audio visual speech recognition with multimodal recurrent neural networks," pp. 681-688, 2017. |

1. Figure 1*: https://medium.com/swlh/types-of-machine-learning-algorithms-62608e83d709* [↑](#footnote-ref-1)
2. Figure 2 *: https://rocketloop.de/en/clustering-with-machine-learning/* [↑](#footnote-ref-2)
3. *Figure 14: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292* [↑](#footnote-ref-3)
4. https://www.researchgate.net/figure/Architecture-of-a-multilayer-neural-network-with-one-hidden-layer-The-input-layer\_fig3\_270274130 [↑](#footnote-ref-4)
5. https://towardsdatascience.com/a-beginners-guide-to-neural-nets-5cf4050117cb [↑](#footnote-ref-5)
6. https://www.researchgate.net/figure/Diagram-of-a-NN-with-two-hidden-layers\_fig4\_235308454 [↑](#footnote-ref-6)
7. Neural Networks and Learning Machines, Simon Haykin, Prentice Hall [↑](#footnote-ref-7)
8. Images taken from https://en.wikipedia.org/wiki/Activation\_function [↑](#footnote-ref-8)
9. https://medium.com/ai%C2%B3-theory-practice-business/hyper-parameter-momentum-dc7a7336166e [↑](#footnote-ref-9)
10. https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651 [↑](#footnote-ref-10)
11. https://www.allaboutcircuits.com/technical-articles/how-to-perform-classification-using-a-neural-network-a-simple-perceptron-example/ [↑](#footnote-ref-11)
12. https://medium.com/@AI\_with\_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f [↑](#footnote-ref-12)
13. https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e [↑](#footnote-ref-13)
14. https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66 [↑](#footnote-ref-14)
15. Main part of the figure from: Yan, S. Understanding LSTM and Its Diagrams. Available online: https://medium.com/mlreview/

    understanding-lstm-and-its-diagrams-37e2f46f1714 (accessed on 26 June 2018) [↑](#footnote-ref-15)
16. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 [↑](#footnote-ref-16)
17. https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/ [↑](#footnote-ref-17)
18. [18] [↑](#footnote-ref-18)