Deep Feedforward Neural Network Classifier with Polynomial Layer and Shared Weights

Filippou Konstantinos

Supervisor: Tsekouras Georgios

February,2023

Computational Intelligence (CI) Research Group,
Department of Cultural Technology and Communications

University of the Aegean

University of the Aegean

Deep Feedforward Neural Network Classifier with Polynomial Layer and Shared Weights

Filippou Konstantinos

Tsekouras George
Professor
Department of Cultural Technology
and Communications
University of Aegean

Anagnostopoulos Christos
Professor
Department of Cultural Technology
and Communications
University of Aegean

Caridakis George
Lecturer
Department of Cultural Technology
and Communications
University of Aegean

Abstract

In this dissertation, we present a unique feedforward neural network for classification problems. Many dense hidden layers are followed by a polynomial layer, whose nodes are inferred using activation functions written as Hermite polynomials. The polynomial layer permits linear combinations of the preceding layers' outputs and extends them into numerous Hermite series with truncated terms. In order to deduce the network's judgment, the Hermite series are aggregated and given to the output layer. The aforementioned technique produces a network with enhanced performance for the following reasons: Hermite polynomials are orthogonal across the whole set of real numbers and have excellent modeling and approximation skills. In addition, they can cope with extremely nonlinear data successfully. In this study, Hermite polynomials are implemented utilizing shared weights across nodes in the polynomial layer, which reduces the amount of design parameters. The performance of the network is evaluated by performing comparative simulation tests using four additional approaches over a number of data sets. The comparison is conducted using t-test conclusion statistics. The experimental results demonstrate that the suggested network performs much better than the other techniques.

Keywords: Deep feedforward neural network, Hermite polynomials, shared weights, classification

Conference

1. The scientific work of this thesis has been accepted and presented in the 4th International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI 2022), Corfu, Greece, October 19-21, 2022

Acknowledgements

First, I would want to express gratitude to God, Jesus Christ, for His help to implement this master thesis, as the journey was not easy ... ***without Me, you can't do anything. (John, 15:5)***. Also, I am grateful to elder monk Gabriel from mount Athos, for his prayers and the general spiritual empowerment.

On a more personal note, I would like to convey my profound appreciation to my adviser, George Tsekouras, for their essential assistance and advice during my master's degree. Their knowledge and support assisted me in doing this study and writing my thesis.

In addition, I would like to thank my University of Aegean colleagues for their support and assistance throughout my study. In particular, I would like to thank George Aifantis for his company during the whole process and for his aid with the Weka simulations, as well as Mavrikos Emmanouil for his assistance with statistical concerns.

Lastly, I am thankful to my family, including my father Nikolaos, my mother Vasiliki, and my sister Styliani, for their unwavering support in all aspects of my life.

Contents

1 Artificial Intelligence

1.1 Introduction

This section is an introduction to Artificial Intelligence. We can say that this scientific field is originated back in 1947 when an English mathematician, Alan Turing gave a lecture about this topic. A lot of scientific and technological ideas were born after imitating physical systems that surround us like the airplane/birds example. Something analogous is happening in AI (abbreviation term for Artificial Intelligence) and human brain. Scientists and engineers try to understand human intelligence and import it, as possible as can be, inside machines. In other words, humanity is trying to *create smart machines*. With one difference, AI does not focus only in procedures that biological systems use but they are moving one step beyond trying to include computational methods that humans may do not have. John McCarthy in 1955 said: *'The goal of AI is to develop machines that behave as though they were intelligent'* [1].

Nevertheless, the difficulty is to distinguish which procedures can be called computational intelligent and which not. Alan Turing in 1950 in his article *Computing Machinery and Intelligence* describe what condition should be exist to characterize a machine as intelligent and is well known as Turing test. In this test, a human is sitting in front of two terminals typing various questions. Behind the first terminal is hidden a human and behind the second terminal just a computer. After the human finishing his question decides which is machine and which is not. We can say that a machine passes the Alan Turing test if it is able to trick human at least 30 % [1].

The final goal is to transform machines having a human intelligence level i.e anticipate and solving problems like humans [2].

The large computational power can perform task that humans can't like a two 20-digit multiplication so someone can argue that computers' intelligence is already higher than humans in some domains. How can we tackle an argument like this and what's the optimal definition of AI ? Elaine Rich shed some light giving a more timeless definition of AI:

Definition 1. *Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better [1].*

In the below graph, we can see a brief history indicating some milestones of various areas in AI.

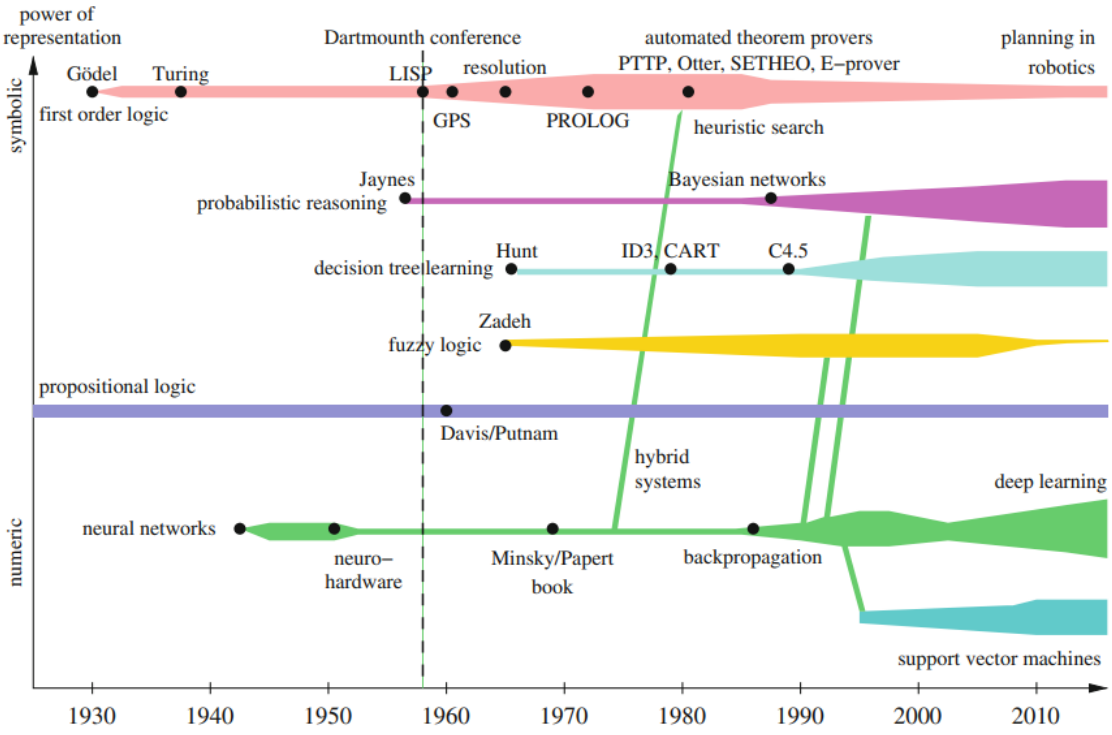


Figure 1: Brief history of various AI's fields.

In next sections, we will try to approach these various AI domains.

1.2 First order logic

Propositional logic can only represents information that accept two logical values *true* or *false*. But, this representation way, in some more complicated cases like natural languages sentences, can be limited. For example, the sentence 'Bob like playing football' is not able to be described using propositional logic and a more powerful mechanism such as the First Order Logic (FOL) needs to be introduced [3].

In artificial intelligence, FOL contains representation techniques that extend propositional logic and are able to express natural language assertions and sentences. It is sometimes referred to as textitpredicated logic [3]. Among of the fundamental characteristics that characterize FOL are:

1. Assume that the physical world is made of facts,objects (like people), relationship between objects (like brother of), functions.
2. FOL considered as another natural language with **syntax** and **semantics**.

The fundamental building blocks of FOL are the subject and the predicate. The subject is the primary component of a statement, and the predicate is analogous to a function that connects the subject and predicate. The statement "Every man respects his parents" is an example of a FOL representation.

$$\forall x \, \text{man}(x) \rightarrow \text{ respects } (x, \text{ parent }) \tag{1}$$

where the subject is man and the predicate is respect(x,y), where x equals parent and y equals man.

Various applications are appearing in AI that use first order logic some examples are the sentence compression model constituting one big challenges in natural language processing [4], the infrastructure for context awareness as FOL gives the ability to express inductive and deductive reasoning using a combination of Boolean operators,universal and existential quantifiers [5], top down induction decision trees [6], the ability to produced inferences known as automated reasoning [7] regression and classification trees [8]. Also part of FOL is contained in more special kind of logics such as fuzzy logic, temporal logic, high order logic and the well known theory of probabilities [9].

1.3 Probabilistic Reasoning

Probabilistic reasoning is a knowledge representation technique that can handle situations where predicates cannot, i.e., when we are uncertain about a statement's prediction. There are several factors in the actual world that make the outcome of an event unpredictable, such as climate change, temperature volatility, experimental mistakes, and inaccurate information sources. In probabilistic reasoning, a mix of logic and probability theory is used to account for the uncertainty of diverse occurrences. In this sector, the Bayes' rule and Bayesian statistics are the most used methods for dealing with uncertain information [10].

Probabilistic reasoning is used in AI with plenty of applications like in [11] that has been used to develop techniques inside predictive expert systems, in [12] for terminological knowledge representation, in the phD thesis [13] has been used for automate software tuning, in [14] that probabilistic reasoning was used by a robot to self recognize if an item belongs to its part or not. Also, in [15] authors created a probabilistic model approach that can classify if logs from a web server are coming from a crawler or by human a problem known as *Web Robot detection*, in economics, authors have been used it for developing intelligent stock trading systems [16] or in language research for spoken dialogue management [17] and many more.

1.4 Decision Trees (DT)

In data mining, statistics, and machine learning, decision trees are a popular and commonly used supervised learning approach due to its simplicity and clarity. Their fundamental purpose is to function as part of a process that utilizes observations to create outcomes. Depending on the nature of the output, they may be separated into two primary categories: classification and regression decision trees. The first are used when the goal variable, i.e. the result, has discrete values consisting of leaves representing class labels and branches indicating conjunctions, whereas the second are used when the target variable has continuous real number values [18]. There are a number of benefits that make decision trees prominent in the field of artificial intelligence. Among them are:

1. They are **simple to understand** and their ability of being graphically displayed produces an easier **interpretation**.
2. They can be used both for categorical and numerical data.
3. Minimum dataset preparation (e.g no need for normalization).
4. They use the **white box** approach giving the ability for boolean logic explanations instead for black box like in in neural networks that the output after a training is difficult to understand.
5. They have good performance with large datasets.
6. Good for approximation boolean functions like XOR.
7. Appears robustness against co-linearity.
8. A model using decision tree can be statistical tested.

On the other hand, some **limitations** of them are:

1. High sensitivity for the outcome when changes in the training data occurs.
2. In decision trees is difficult an optimal tree to be found a problem known as **NP-complete**.
3. Over complex decision trees structures can forall into the **overfitting** trap.

A huge number of scientific papers exist using decision trees handling through them AI challenges. In [19] a combined method using artificial neural networks (will refer analytically later to them) and decision tree is proposed for making prognosis of the breast cancer relapse. In [20] oil price is predicted through them (decision stump, random forest and random tree algorithms have been used) using a dataset over 24 years. In [21] DT have been used for a failure analysis happening through internet request gathering and processing requests' properties using automated machine learning and data mining methods. In [22] authors propose the idea of evolving decision trees through the usage of genetic algorithms. In [23] DTs have been used for ecological purposes developing an applications that examines population dynamics and for different organisms, ecosystems and environmental conditions. Also, in [24] a novel algorithm has been proposed using DTs for handling nonlinear metrics. An interesting application of DTs has been proposed in [25] where authors used them inside hardware architecture (examples of hardware DTs implementation are oblique, non linear or axis parallel) showing that their presence can save up to 56 % less resources of hardware.

1.5 Fuzzy logic

The terms fuzzy logic represents states where an truth value of a variable can take any real number in the interval [0,1] in contrast to classic boolean logic that the value can be either in 0 or 1 state. Basically, this type of pseudo boolean variable handles a concept known as 'partial truth'. Historically, this term introduced on 1965 by Lotfi Zadeh. In the foundations, in fuzzy logic someone takes a decision based non numerical and uncertain data. For example, if we ask a group of people to identify a specific color (i.e a specific wavelength) then answers will vary and the truth will vary base on the distinctive ability of their eyes. Hence, fuzzy logic are constrained by models trying to recognize,describe,manipulate, interpret ambiguous information. We have to note that, fuzzy logic at first glance may be similar with probability theory as both take value from 0 to 1 but fuzzy logic try to approach vagueness while probabilities ignorance. Fuzzy logic and artificial intelligence have common structure as the logic behind neural networks is fuzzy, in other words, the result of the neural network training process is a numeric number between 0 and 1. If the output is hence closer to 0 or 1 a choice is made [26].

Fuzzy logic is applied a lot in the medical decisions with applications in image analysis, signal analysis from biomedical data, and feature extraction. Some still ongoing challenges although are how to produce that appropriate fuzzy data that coming from patients, how to validate and how to assess the quality of them. An example constitutes CAD or in other words Computer Aided Diagnosis. This is as set of tools that helping aid physicians for making diagnosis from medical images like MRI or x-Rays. When an physician recognizes an potential disease in an patient's medical examination he/she can use CAD to interpret this lesion and the fuzzy logic takes place and helps to describe the main properties of it [26].

A popular fuzzy system is known as Mamdani and follows three rules first 'fuzzify' inputs into fuzzy functions, then run all possible rules to calculate the output fuzzy functions and finally 'defuzzify' the output fuzzy functions to get the output values. The term known as *"fuzzification"* is the procedure where the numerical input is being corresponded to fuzzy sets. In the picture 2, for instance, the temperature states cold, warm, and hot are mapped to a temperature scale where each point corresponds to three truth values for each state. The arrows in the vertical line shows exactly this i.e for a specific temperature what's the correspond state. The temperature that the red arrow (representing the hot state) indicates points a zero value means that this temperature is "not hot". The orange arrow (warm state representative) points, for the temperature values under discussion, in a value around 0.2 and the blue arrow in 0.8. Hence, this temperature has zero membership for the 'hot' fuzzy set, 0.2 membership in the 'warm' set and 0.8 in 'cold' set [26].

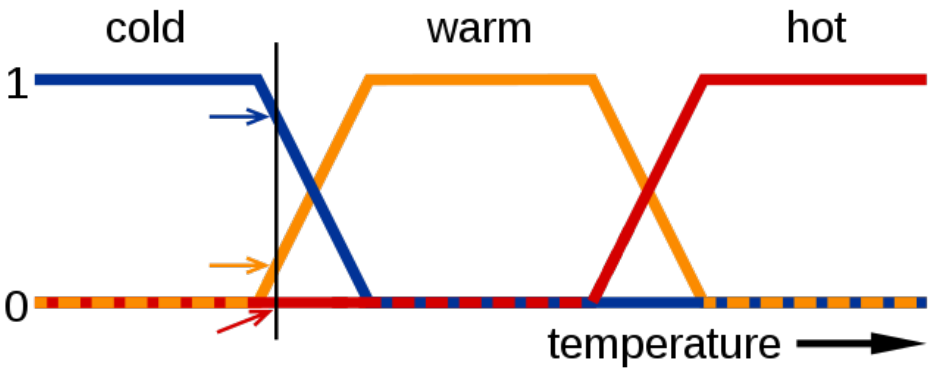


Figure 2: A fuzzy logic example using temperature states cold, warm,hot

Fuzzy sets can be defined using triangle or trapezoids indicating that a value is increasing, reach a peak and then decreasing again. Also, sets can be represented by use of the sigmoid function:

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

Numerous applications of fuzzy logic can be found in the artificial intelligence domain. In [27] is discussed about fuzzy logic in industrial engineering, in [28] a review of applications in the building technology is presented focusing on the improvement of indoor and energy conservation problem, in [29] solving more complex power systems problems, in to evaluate student performance, in [30] fuzzy logic has been combined with genetic algorithms to address an automated works transport organization problem, in image edge detection [31], for ground and air vibrations prediction [32],

1.6 AI, Machine Learning and Deep Learning

Artificial Intelligence, Machine Learning, and Deep Learning are three linked computer science disciplines that have a growing effect on how people live, work, and communicate. These technologies are fast growing and will have a significant impact on the future of computer and human connection. In this part, we will explore what AI, Machine Learning, and Deep Learning are, as well as their similarities and distinctions, as well as some of their most important applications and consequences [33].

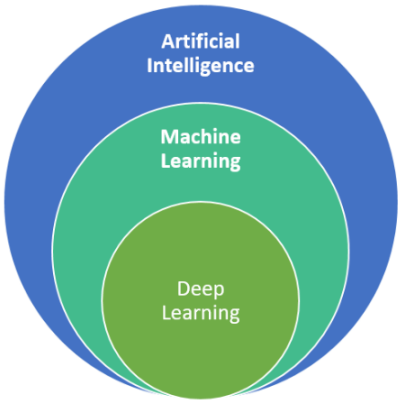


Figure 3: Artificial Intelligence, Machine Learning & Deep Learning.

Artificial Intelligence (AI) is a vast branch of computer science that entails the creation of computers and algorithms capable of doing activities that normally require human intellect. The objective of artificial intelligence is to develop systems that can think, learn from experience, comprehend natural language, identify objects and patterns, and make judgments. AI systems may be classified into two distinct categories: rule-based and learning-based. Rule-based systems make judgments based on predefined rules, while learning-based systems employ algorithms to learn from data and generate predictions [33].

Machine Learning (ML) is a subfield of artificial intelligence that focuses on the creation of algorithms that allow computers to learn from data and make predictions. Depending on the nature of the data they are trained on, ML algorithms may be supervised, unsupervised, or semi-supervised. Unsupervised ML algorithms are trained on unlabeled data where the output is unknown. Supervised ML algorithms are taught on labeled data when the right output is already known. Semi-supervised machine learning algorithms are taught using both labeled and unlabeled input [33].

Deep Learning (DL) is a kind of machine learning (ML) that is based on artificial neural networks inspired by the structure and function of the human brain. Deep Learning algorithms use numerous layers of artificial neurons to study and learn from vast quantities of data, allowing them to generate very accurate predictions. Particularly well-suited are DL algorithms to problems involving picture and audio recognition, natural language processing, and autonomous systems [33].

AI, ML, and DL all entail the creation of algorithms that allow computers to accomplish activities that would otherwise need human intellect. They all depend on data to learn and make predictions, another commonality. The breadth and concentration of their applications is a major distinction between these technologies. AI is a wide area that includes both rule-based and learning-based systems, while ML is primarily concerned with learning-based systems. DL, on the other hand, is primarily concerned with neural networks and their capacity to learn from vast volumes of data [33].

The applications of AI, ML, and DL are quite diverse and impact almost every industry, from finance and healthcare to retail and transportation. In finance, for example, AI and ML are used to analyze market data, predict stock prices, and detect fraud. In healthcare, AI and ML are used to diagnose diseases, predict patient outcomes, and improve patient care. In retail, AI and ML are used to personalize customer experiences, optimize supply chain management, and increase sales [33].

AI, ML, and DL are quickly growing technologies that have a substantial influence on how we live, work, and communicate. While there are many similarities between these technologies, such as their dependence on data to learn and generate predictions, there are also significant distinctions, such as their application breadth and emphasis. It is conceivable that as these technologies continue to develop, they will play an increasingly important role in shaping the future of computer and human interaction [33].

1.7 Section References

[1] Wolfgang E., Introduction to Artificial Intelligence, Second Edition, Springer

[2] McCarthy, J. (2007). What is artificial intelligence.

[3] First-Order Logic in Artificial intelligence, https://www.javatpoint.com/first-order-logic-in-artificial-intelligence

[4] Huang, M., Shi, X., Jin, F., & Zhu, X. (2012). Using first-order logic to compress sentences. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 26, No. 1, pp. 1657-1663).

[5] Ranganathan, A., & Campbell, R. H. (2003). An infrastructure for context-awareness based on first order logic. Personal and Ubiquitous Computing, 7, 353-364.

[6] Blockeel, H., & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. Artificial intelligence, 101(1-2), 285-297.

[7] Klinov P., Mouromtsev D., Knowledge Engineering and the Semantic Web, 4th International Conference, KESW 2013 St. Petersburg, Russia, October 2013 Proceedings

[8] Kramer, S., & Widmer, G. (2001). Inducing classification and regression trees in first order logic. Relational Data Mining, 140-159.

[9] What is first-order logic in Artificial Intelligence? https://www.educative.io/answers/what-is-first-order-logic-in-artificial-intelligence

[10] Probabilistic reasoning in Artificial intelligence https://www.javatpoint.com/probabilistic-reasoning-in-artifical-intelligence

[11] Spiegelhalter, D. J. (1986). Probabilistic reasoning in predictive expert systems. In Machine Intelligence and Pattern Recognition (Vol. 4, pp. 47-67). North-Holland.

[12] Jaeger, M. (1994, January). Probabilistic reasoning in terminological logics. In Principles of Knowledge Representation and Reasoning (pp. 305-316). Morgan Kaufmann.

[13] Sullivan, D. G., Seltzer, M. I., & Pfeffer, A. (2004). Using probabilistic reasoning to automate software tuning. ACM SIGMETRICS Performance Evaluation Review, 32(1), 404-405.

[14] Gold, K., & Scassellati, B. (2009). Using probabilistic reasoning over time to self-recognize. Robotics and autonomous systems, 57(4), 384-392.

[15] Stassopoulou, A., & Dikaiakos, M. D. (2009). Web robot detection: A probabilistic reasoning approach. Computer Networks, 53(3), 265-278.

[16] Bao, D., & Yang, Z. (2008). Intelligent stock trading system by turning point confirming and probabilistic reasoning. Expert Systems with Applications, 34(1), 620-627.

[17] Roy, N., Pineau, J., & Thrun, S. (2000, October). Spoken dialogue management using probabilistic reasoning. In Proceedings of the 38th annual meeting of the association for computational linguistics (pp. 93-100).

[18] Decision Trees, wikipedia, https://en.wikipedia.org/wiki/Decision_tree_learning

[19] Jerez-Aragonés, J. M., Gómez-Ruiz, J. A., Ramos-Jiménez, G., Muñoz-Pérez, J., & Alba-Conejo, E. (2003). A combined neural network and decision trees model for prognosis of breast cancer relapse. Artificial intelligence in medicine, 27(1), 45-63.

[20] Nwulu, N. I. (2017, September). A decision trees approach to oil price prediction. In 2017 International Artificial Intelligence and Data Processing Symposium (IDAP) (pp. 1-5). IEEE.

[21] Chen, M., Zheng, A. X., Lloyd, J., Jordan, M. I., & Brewer, E. (2004, May). Failure diagnosis using decision trees. In International Conference on Autonomic Computing, 2004. Proceedings. (pp. 36-43). IEEE.

[22] Papagelis, A., & Kalles, D. (2000, November). GA Tree: genetically evolved decision trees. In Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000 (pp. 203-206). IEEE.

[23] Jørgensen, S. E., & DeAngelis, D. (2011). Modelling Complex Ecological Dynamics: An Introduction into Ecological Modelling for Students, Teachers & Scientists. Springer Science & Business Media.

[24] Demirović, E., & Stuckey, P. J. (2021, May). Optimal decision trees for nonlinear metrics. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 5, pp. 3733-3741).

[25] Struharik, J. R. (2011, September). Implementing decision trees in hardware. In 2011 IEEE 9th International Symposium on Intelligent Systems and Informatics (pp. 41-46). IEEE.

[26] Jafari, R., Contreras, M. A., Yu, W., & Gegov, A. (2020). Applications of fuzzy logic, artificial neural network and neuro-fuzzy in industrial engineering. In Industrial and Robotic Systems: LASIRS 2019 (pp. 9-14). Springer International Publishing.

[27] Fuzzy logic, wikipedia, https://en.wikipedia.org/wiki/Fuzzy_logic

[28] Kolokotsa, D. (2007). Artificial intelligence in buildings: A review of the application of fuzzy logic. Advances in Building Energy Research, 1(1), 29-54.

[29] Song, Y. H., & Johns, A. T. (1997). Applications of fuzzy logic in power systems. Part 1: General introduction to fuzzy logic. Power Engineering Journal, 11(5), 219-222.

[30] Gola, A., & Kłosowski, G. (2018). Application of fuzzy logic and genetic algorithms in automated works transport organization. In Distributed Computing and Artificial Intelligence, 14th International Conference (pp. 29-36). Springer International Publishing.

[31] Mathur, S., & Ahlawat, A. (2008). Application of fuzzy logic on image edge detection.

[32] Mohamed, M. T. (2011). Performance of fuzzy logic and artificial neural network in prediction of ground and air vibrations. JES. Journal of Engineering Sciences, 39(2), 425-440.

[33] Campesato, O. (2020). Artificial intelligence, machine learning, and deep learning. Mercury Learning and Information.

2 Neural Networks

As the main ingredient of this master thesis are neural networks we will present an extensive analysis of them in this separate section. Artificial neural networks (ANN) or simply Neural Networks (NN) are computational systems trying to mimic biological neural networks. Their main functionality that are trying to adapt is 'learning' in the same sense as our brain learns. For example, a NN is able to *learn* to classify if something is cat,dog or sofa or are able to predict given specific inputs what will be the next price (regression process).

2.1 Structure

Let's take closer look in the basic neural network structure which can be presented in the below image (22):

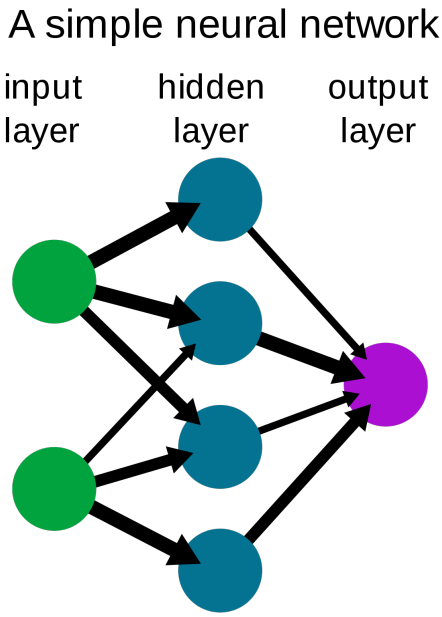


Figure 4: Basic neural network architecture

As we can see, there are five main components of a neural network: the input layer, the hidden layer, the output layer, neurons, and synapses or weights. There is one more part that is hidden and exists inside neurons known as activation function. In every neuron a specific calculation occurs. It accepts numerical values from previous neurons and produce an output that is a single value. In the below image (5), three input scalar values insert inside the neuron. These values are connected with w1,w2 and w3 weights. Then a very specific calculation takes places, which constitutes the core of the neural network. It contains two subparts, first the inner product $w_i x_i$ (sometimes a term is being add name as *bias* $w_i x_i + bias$) and this result is then sent as an argument to a function called *activation function*. The outcome, in this case Y, is the characteristic output of the neuron.

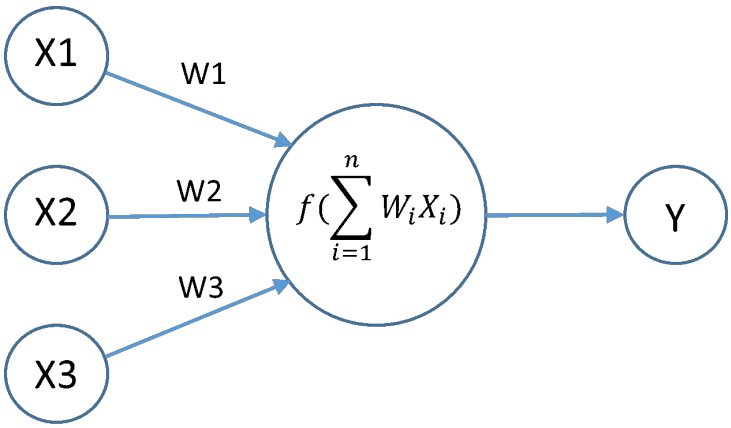


Figure 5: The core calculation behind neural networks

Analogously, in a more complex structure these calculations are happened in every neuron producing the final result.

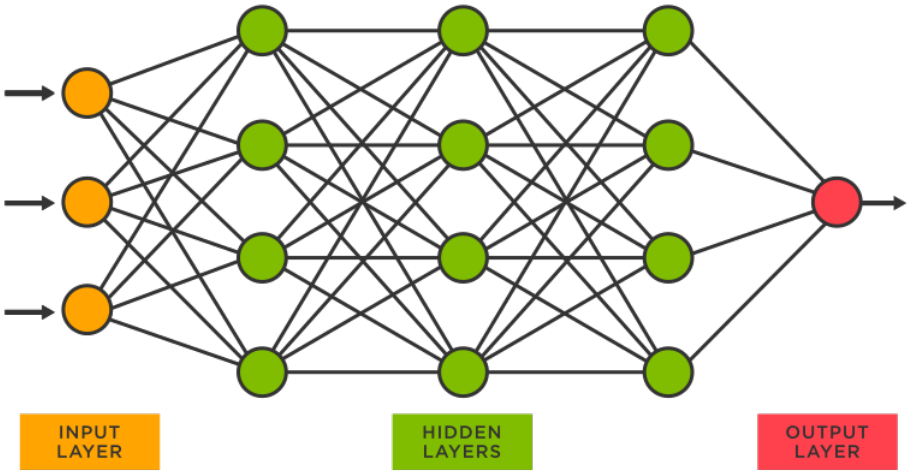


Figure 6: A more complex neural network structure.

2.2 Weight initialization

The initialization of weights is crucial to the training of neural networks. The objective of the optimization method is to minimize the loss function with regard to the weights, or to identify the ideal weights combination that minimizes loss. First, we must begin with certain basic weights that will fluctuate during the training period. Varying beginning values indicate distinct training and performance [34].

2.2.1 Why does machine learning provide different outcomes each time?

The nature behind machine learning models is to produce each time different results. In machine learning there are two concepts that has to be distinguish model and algorithm. The algorithm is all this steps that run on a dataset that produce a model. The later is used to make predictions on data. Or in other words, a machine learning model is a programme that learns using historical data via an automated way. The automation contains a mapping function between inputs and outputs i.e they try to approximate this mapping function [35].

In contrast to classical programmes that follow an if/else logic machine learning models are not deterministic and then they produce different results each time we run them i.e different error level and difference performance level. Four basic components that they can lead to difference training/testing results are the nature of each dataset, the stochastic learning algorithms (like stochastic gradient descent), the diversity of the evaluation methods and also the platform itself that we train/test the model [35].

Different results due to training data

The variance of a machine learning algorithm is a measure of the algorithm's sensitivity to particular training data or, in other words, the sensitivity of the algorithm with respect to specific training data. Taking into account that the model is a function approximator and the law of large numbers, methods with a greater apparent sensitivity often need more training data. In general, training a machine learning algorithm on various training datasets will result in distinct models, i.e. different prediction abilities with different estimators (loss,accuracy) [35].

Different results due to learning algorithms

There are several machine learning algorithms that are stochastic and as a result produce slightly different outcomes (prediction,performance etc) when they run on the same dataset multiple times or in other words these algorithms learn slightly different models [35]. A process can be called stochastic if it governed by stochastic variables i.e variables that they carry some kind of randomness in them. Popular stochastic processes are the Brownian motion, Monte Carlo sampling, the Markov process etc [36].

In machine learning, optimization and learning algorithms appears to be stochastic as they may use probabilities to make decisions. Stochastic optimization algorithms leverage randomness to optimize an objective function or they try to optimize an already random objective function. Generally, inside their process they are trying to find the optimal position in a search space taking account probabilistically the previous positions [36].

The stochastic gradient descent technique is a common one. Stochastic Gradient Descent (SGD) is an optimization process used to update the parameters of a machine learning model in order to minimize its error. The gradient of the error function is estimated using random data samples (or "mini-batches"), as opposed to the whole dataset. This may make the method faster and more efficient, especially when dealing with enormous datasets. The fundamental concept is to update the parameters repeatedly in the opposite direction as the gradient of the error function, with a step size specified by a learning rate. Several machine learning techniques, such as linear regression and neural networks, use it extensively [48]. Overall:

1. Stochastic algorithms may assist to prevent being caught in local optima, which can be a problem in classic deterministic algorithms.

2. Moreover, randomness may be employed to increase the variety of training data, which can lead to improved generalization performance.

3. Stochastic algorithms can be more efficient than deterministic algorithms, notably when dealing with massive quantities of data.

4. Random sampling can be used to approximate complex distributions and find the optimal solution in a computationally efficient manner.

5. It is also essential to mention that the data itself is inherently stochastic and algorithms that can model the noise in the data are more likely to perform well.

There are many approaches to mitigate the unpredictability generated by stochastic algorithms. Initially, by establishing the seed for the algorithm's random number generator. This will guarantee that the exact same sequence of random integers is created each time the method is executed, allowing for repeatable results. Second, by using methods like as cross-validation, in which the data is divided into various subgroups and the algorithm is trained and evaluated numerous times using these subsets. This may assist to predict the algorithm's generalization performance and limit the impact of randomness. In addition, several algorithms have hyperparameters that regulate the amount of randomization. In Random Forest, for instance, the number of trees and the number of features to evaluate at each split are hyperparameters that may be modified to regulate the unpredictability [38]. Although randomness cannot be completely controlled, it is feasible to limit its impact by strategies such as establishing the seed, cross-validation, and modifying the hyperparameters [39].

Differences due to evaluation process

However, the assessment procedure may provide different results even when the identical method and data are used [40]. The *train-test* split and *k-fold cross validation* are two prominent assessment techniques. The first step entails separating the data into two sets: the training set and the test set. The training set is used to train the model, whilst the test set is used to assess the model's performance. It is crucial to highlight that the train-test split approach is a basic assessment method with significant limitations, such as the model's performance being heavily reliant on the particular data split and test set sample size. With the second method, k-fold cross validation, the previously noted issue may be resolved. It separates a dataset into k non-overlapping subsets and utilizes one subset as the test set and the remaining subsets as the training set.

2.3 Activation function

As we notice in the previous section, each neuron makes a calculation

$$f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) \tag{1}$$

where w_i are weights, x_i are inputs, b_i is the bias and f the activation function.

2.3.1 Why we use activation function

Neural networks accepts inputs (input layer) and produce outputs (output layer). Suppose that we do not use activation function in the NN structure, then the result would be a straightforward linear combination or in other words a polynomial order one i.e the NN will act as a classical linear regression model. Introducing an activation function inside the Neural Network mechanism, NN can learn to detect, develop, and extract information from complicated, non-linear, high-dimensional datasets. Hence, a NN can not only be a linear regression model but perform complicated tasks like processing audio,speech,images etc [41].

When a NN is constructed it has to be generic able to process any random dataset that maybe contains non-linearity. Hence, the choice of activation function has to fullfil this requirement. In other words, non-linearity of the initial dataset can be handled by a non-linear activation function. Also, another appropriate requirement is to be differentiable function as in the training process back propagation (procedure that computes derivatives) is used to optimize the loss function [41].

2.3.2 The vanishing gradient problem

In this section, we will briefly make a reference to the vanishing gradient issue that is close related with the nature of the chosen activation functions. In machine learning, when artificial neural networks are trained via backpropagation and gradient methods (like gradient descent) the optimal parameters of the NN (i.e weights and biases) are being found through an update that happens on them that is proportional with the partial derivative of the loss function with respect to a certain weight. In other words,if this derivative is small implies a small weight update. The worst-case situation is that the derivative will be so little that training will cease and hence the network not to be the optimal. This issue is referred to as **the vanishing gradient problem** [44]. To further explain this issue, we shall provide an illustration. We suppose a sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ where its graphical representation and its derivative are given by:

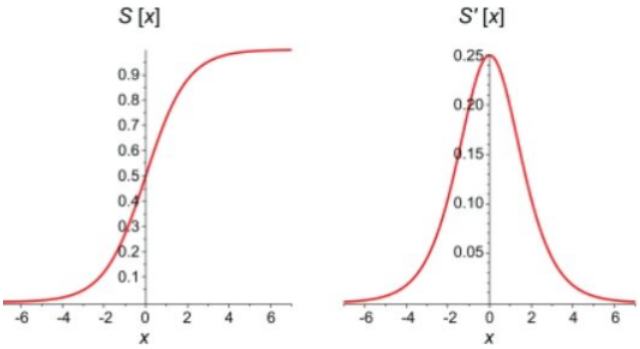


Figure 7: The sigmoid and its derivative.

In the backpropagation the new updated weight is given by the formula:

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} - \eta * \frac{\partial C}{\partial w} \tag{2}$$

where w_{new} is the updated weight, w_{old} the old weight, η is the learning rate parameter and $\frac{\partial C}{\partial w}$ is the gradient of the loss function relative to a certain weight. Applying the chain rule to the above formula we can format a type that yields the gradient of the loss function as the product of the derivatives of all activation functions relative to their weights i.e the chain rule [42]

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} \tag{3}$$

where $z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l$ is the activation function. Hence, we can note that the weight update rate of nodes is rely on the activation functions' gradients. All nodes that contain sigmoid, base on the graph its partial derivative can be a max value of 0.25. When more layers exists, products of these order of magnitude, produce even smaller values until there is one point that the value becomes almost zero [43]. Error (loss gradient) decreases exponentially with n, where n is the number of hidden layers [44]. Hence, in deep network architectures using sigmoid can be a significant performance factor [43]. Simpler, in a neural network that face *vanishing gradient* weights are not being update and the network does not learn.

An obvious way to overcome this issue is to replace the activation function with another one that its derivative does not take small value. A good candidate is the ReLu where its derivative is 1 for input values greater than zero and zero for negative values avoiding the gradient from being vanished turing training.

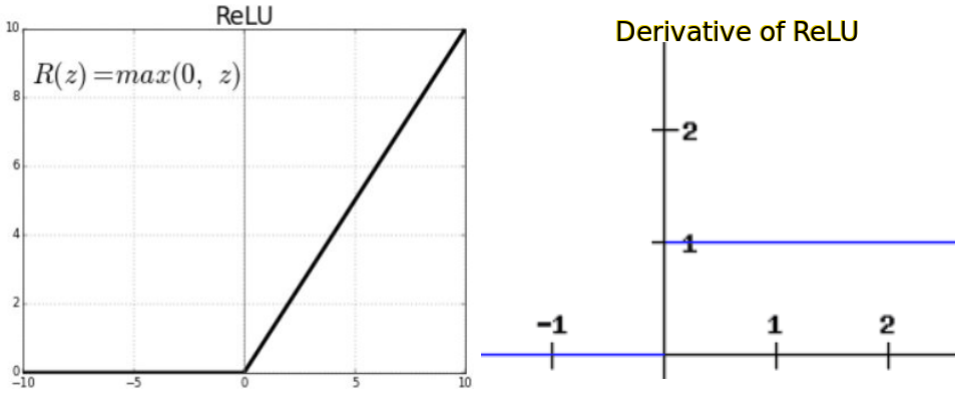


Figure 8: The ReLU and its derivative. As we can see its derivative is > 0 overcoming the vanishing gradient problem.

We have to note that also ReLU can take zero values when the input is negative and hence training does not being applied for these nodes. This can be treated using the leaky ReLU. Generally is being defined as:

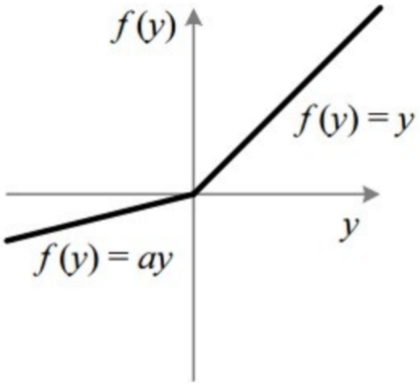


Figure 9: Leaky ReLU

where a is the slope coefficient for negative inputs that is being defined before the training [45].

A second way to face the vanishing gradient is to initialize properly the weights in order during the training period not be vanished [44].

2.3.3 Types of activation functions

Activation functions, also known as threshold functions or transfer functions, represent a scalar to scalar transformation [41]. There are a lot of different candidates, and below we will present some of the most popular.

Sigmoid: It is one of the most used in NNs, it is non-linear, converts the values in the range 0 to 1, is continuously differentiable and its mathematical is defined as:

$$\text{sig}(t) = \frac{1}{1 + e^{-t}} \tag{4}$$

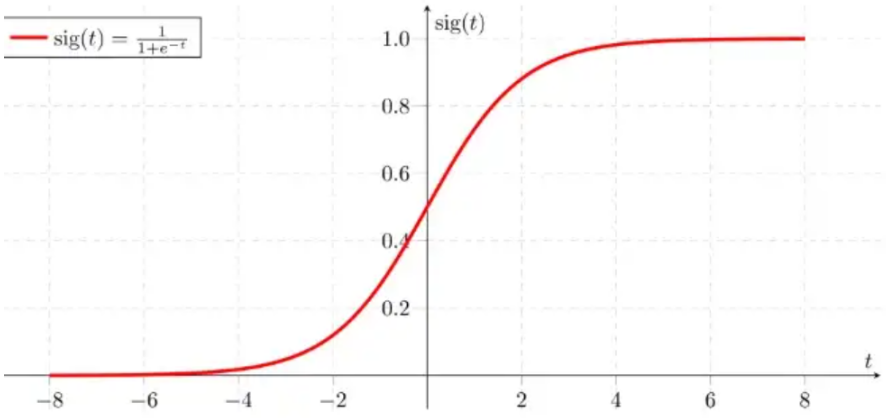


Figure 10: Sigmoid

Tanh: The Tangent hyperbolic activation function (Tanh) transforms values in the -1 to 1 range (i.e cover the previous layer's symbol with the following layer's.).It is also continuously differentiable, zero centered and mathematically defined as:

$$\tanh x = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{5}$$

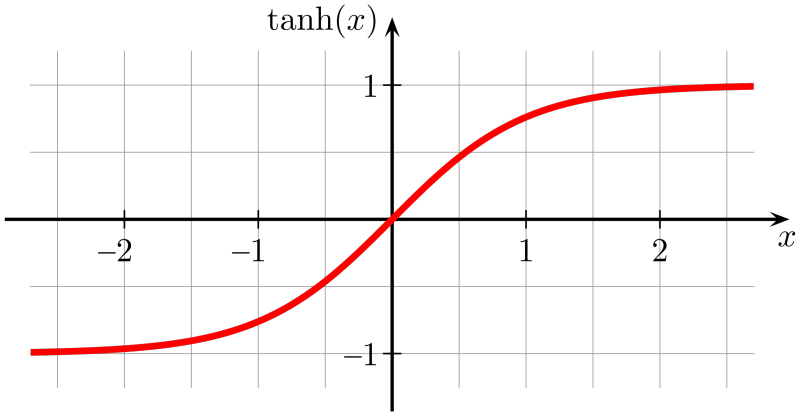


Figure 11: Tangent hyperbolic activation function (Tanh)

ReLU: ReLU is shorthand for rectified linear unit. It is a commonly used activation function in neural networks, having applications in voice recognition and computer vision [46]. Mathematically defined as:

$$f(x) = x^+ = \max(0, x) \tag{6}$$

where x is the input to the neuron.

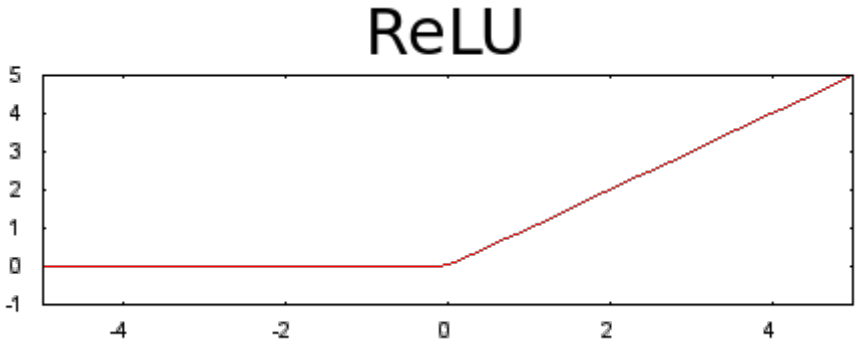


Figure 12: ReLU (Rectified Linear Unit Function)

As seen in the graph above, when ReLU is used, activates the neurons that contain greater than zero input values. This property known as sparse activation function is make in it more efficient from other activation functions as at a certain time not all neurons are activated [41]. Also it is scale invariant i.e the equation (6) becomes $\max(0, ax) = a\max(0, x)$ for $a \geq 0$. On the other hand, in zero is not differentiable and usually the derivative at this point is chosen randomly to has a value i n the range (0,1][46] or just neither weights nor biases are updated during the backpropagation phase [41].

2.4 Neural Network training

In this part, we will examine the process of training Neural Networks. Training a neural network entails modifying the network's parameters so that it can execute a job on a given dataset, such as classification or regression.

In the thesis, **forward propagation** was used. In forward propagation, the inputs are sent through the network to generate predictions for the output. The output is generated by multiplying the inputs by the network's weights (parameters) and passing them through activation functions (e.g. sigmoid, ReLU, etc.). The difference between the expected and actual outputs is then computed and utilized to modify the parameters in the subsequent stage.

A **Loss function** is used to compute the difference between the expected and actual outputs (e.g. mean squared error, cross-entropy, etc.). The loss function quantifies the network's performance on a given task, and the objective is to minimize the loss [47].

Next, the **Backpropagation** procedure occurs. Backpropagation is the process of calculating the loss gradient with regard to the network's parameters. The gradient is used to adjust the parameters so as to minimize loss. Typically, optimization algorithms such as gradient descent, Adam, etc. are used for this purpose. Gradient descent is a well-known optimization process that modifies parameters in the direction of the negative gradient. This indicates that the settings are modified so as to minimise the loss. The size of the update is determined by the learning rate, which controls the update's step size [47].

Several **epochs** are used to repeat the steps of forward propagation, computing the loss, backpropagation, and parameter update. An epoch is one run of the dataset in its entirety. The objective is to repeat this procedure numerous

times so that the parameters may be changed and the network can gain knowledge from the data. Typically, in the training phase, something known as **batch size** is used. The batch size is the number of training data samples that are processed in a single forward-backward trip through the network. In other words, the batch size affects the number of samples used in each training cycle. For instance, if the batch size is 32, each iteration will use 32 samples from the training data. Batch size has an immediate effect on the training process. A reduced batch size might result in a noisy gradient estimate, which can hinder convergence. A higher batch size, on the other hand, may result in a more stable estimate of the gradient, but it can also demand more memory and take longer to calculate. Sometimes, epochs and batch size are perplexing notions, particularly for novices. Epochs, on the other hand, relate to the number of times the full set of training data is processed, as we said earlier. In each epoch, the network processes the training data in batches (the batch size determining the batch size) until all samples have been seen. After one epoch, the weights are revised and the procedure is repeated. Repeat this procedure until the appropriate number of epochs has been attained. In conclusion, batch size controls the number of samples processed during each iteration, while epochs dictate the number of times the complete training data set is processed. Batch size and epochs are key hyperparameters to consider while training a neural network [47] since they may have a major influence on the network's performance.

Next, in some trainings, **regularizations** techniques are selected. Regularization is a method used to avoid overfitting, which happens when a network grows too complicated and begins to remember the training data instead of generalizing to new data. Regularization adds a penalty term to the loss function to deter models with excessive complexity. Generalization is the capacity of a neural network to perform effectively not just on the data it was trained on, but also on data it has never seen before. It is a crucial component of machine learning since the ultimate objective is to create models that can generalize to new, unobserved data. Overfitting is the antithesis of generalization and happens when a model grows too complicated and begins to remember the training data instead of generalizing to new data. Many strategies may be used to enhance the generalization of a neural network:

- The most prevalent regularization methods in neural networks are **L1 regularization** (also known as Lasso), **L2 regularization** (also known as Ridge), and dropout. L1 and L2 regularization add a penalty term dependent on the size of the weights to the loss function, while dropout randomly eliminates neurons during training [48].
- **Early Stopping:** Early stopping is a training approach that terminates when the performance on a validation set begins to deteriorate. T This helps minimize overfitting by restricting the amount of training epochs [49] the network may undergo.
- **Cross-Validation:** Cross-validation is a method used to assess the performance of a model using data that has never been seen. Cross-validation divides the data into numerous folds and trains and evaluates the model on each fold. The performance on each fold is then summed to assess the overall performance of the model [50]. Cross-validation was employed in this thesis to predict overfitting.

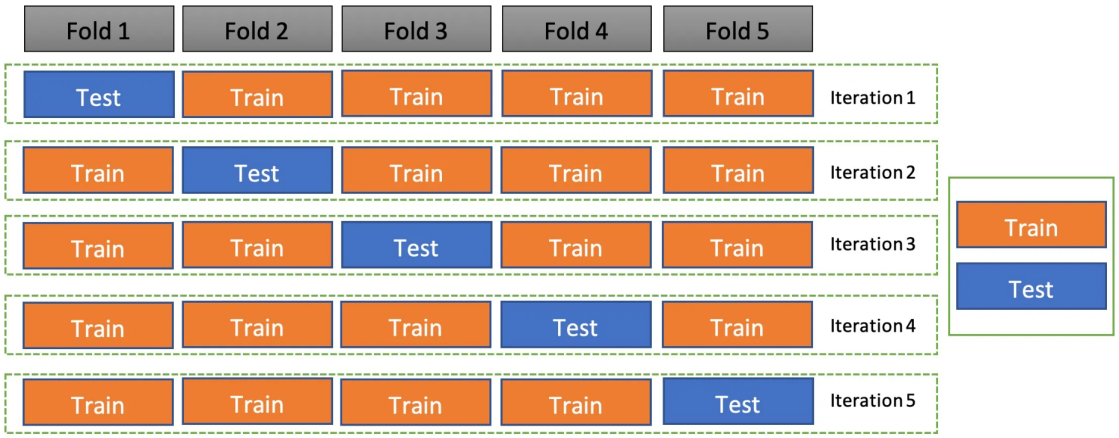


Figure 13: Cross validation technique, the original dataset is splitted in k-folds (here 5-folds). In each iteration of this example, different 80 % of the dataset are utilized for training and the remaining 20 % are used for testing.

- **Ensemble Methods:** Ensemble approaches combine numerous models into a single, more precise model. A random forest, for instance, is a collection of decision trees. Ensemble approaches may increase a model's generalization by leveraging the strengths of numerous models and minimizing the influence of overfitting on any particular model [51].
- **Data Augmentation:** Data augmentation is a method for increasing the quantity of training data by producing extra synthetic examples. This may enhance the generalizability of a model by reducing its susceptibility to overfitting [52].

During the training of a neural network, a number of additional factors have a significant influence on its performance. This includes the number of layers, number of neurons in each layer, activation functions, learning rate,

regularization strength, batch size, etc. These parameters are referred to as **hyperparameters** and must be specified before the training. The optimal hyperparameters are typically determined through a process called hyperparameter tuning.

Tuning hyperparameters is a crucial phase in the training process since the choice of hyperparameters may have a substantial effect on network performance. A big learning rate, for instance, may result in a network that converges slowly or not at all, while a modest learning rate may result in a network that converges too slowly [53].

There are several techniques that can be used to perform hyperparameter tuning, including [53]:

1. **Grid Search:** Grid search is a basic and easy approach that includes testing the performance of a model for each combination of hyperparameters in a preset grid. Based on the various values for each hyperparameter, the grid is created. The optimal combination of hyperparameters is then chosen based on the model's performance for each hyperparameter combination.

2. **Random Search:** Random search is a more efficient strategy than grid search, since it includes selecting hyperparameters at random from a predetermined distribution for each hyperparameter. The model's performance is then assessed for each set of hyperparameters, and the set of hyperparameters resulting in the best performance is chosen.

3. **Bayesian Optimization:** Bayesian optimization is a more complex approach that models the link between the hyperparameters and the performance of the model using a probabilistic model. The probabilistic model is updated after each assessment of the model's performance, enabling the algorithm to learn which hyperparameters are most likely to result in high performance and to concentrate its search on those regions.

In conclusion, hyperparameter tuning is the method of determining the optimal collection of hyperparameters for a neural network.

The **training** dataset is another key component in the training of a neural network that may have a considerable influence on the performance of the model. The quality and quantity of training data may have a direct impact on the network's capacity to learn and generalize to new, unknown input. Following are some specifics on the effect of training data on the performance of neural networks [54,55].

Firstly, the **quality of the training data** is crucial since it directly influences the network's capacity to learn and generalize. For instance, if the training data includes mistakes or outliers, it might have a detrimental effect on the network's performance. In contrast, high-quality training data that precisely portrays the issue may aid in enhancing the network's performance.

Second, the **amount of the training data** may also have a substantial effect on the network's performance. In general, a bigger training set may increase the network's generalization since it enables the network to learn from more different instances. Yet, a very large training set may be computationally costly and may not be required f or simpler tasks.

Third, the **diversity of the training data** might also have an impact on the network's performance. Insufficiently varied training data might result in overfitting, when the network memorizes the training data rather than learning the underlying patterns. A diversified training set, on the other hand, may help the network generalize better and prevent overfitting.

Fourth, the **representativeness** of the text: The training data should be indicative of the issue and data distribution that will be encountered by the network during inference. If the training data is not representative, underfitting might occur, where the network is unable to predict the issue effectively.

In addition, something known as **batch normalization** is considered in the neural network procedure. Deep learning employs batch normalization to equalize the inputs to each layer of a neural network. It is intended to enhance the stability and speed of training, as well as the network's generalization [56].

The objective of batch normalization is to minimise the internal covariate shift, which is the variation in the activation distribution as the network trains. This shift in distribution might slow down the network's training and make it more difficult for it to learn. Normalizing the inputs to each layer using batch normalization helps to reduce this issue by stabilizing the training process. Typically, batch normalization is achieved by adding an extra normalization-performing layer to the network. This layer adds a learnt scale and shift to the activations from the preceding layer. The normalized activations are subsequently sent to the subsequent network layer. The batch normalization layer has two hyperparameters: scale and shift. During training, these parameters are learnt and utilized to rescale the normalized activations such that they have the appropriate distribution.

Batch normalization has several benefits, including improved **stability and speed of training**, **reduced overfitting**, and **improved generalization**. By normalizing the inputs to each layer, batch normalization may assist lessen the internal covariate shift, which can increase the training process's stability. It may also aid in reducing overfitting by regularizing network activations. Additionally, batch normalization can improve the generalization of the network by normalizing the activations, which can help to **reduce the network's dependency on the details of the training data.** [57].

2.5 Loss function

As previously discussed, loss functions play a significant role in the training of neural networks since they are used to quantify the difference between the network's predictions and the real values. The loss function offers an objective measure of the inaccuracy in the network's predictions and directs the training optimization process. In this part, we will go further into this notion and the categorical crossentropy loss functions used in the simulations for this thesis.

The loss function offers a scalar number representing the inaccuracy in the network's predictions, and the objective of the training optimization process is to decrease this loss. There are a variety of loss functions, such as mean squared error, mean absolute error, cross-entropy, and hinge loss. Each form of loss function is tailored to a unique class of issues and may be more or less suitable for a variety of applications [58].

Mean Squared Error is a loss function that estimates the average squared difference between the predicted and actual values. MSE is often used to regression issues in which the objective is to predict a continuous value and is give by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{1}$$

where n is the sample size, Y_i is the true value for sample i and $\hat{Y}_i$ is the predicted value for sample i [58].

Hinge Loss: Hinge loss is a typical loss function for support vector machines and is also used by some deep learning models. Hinge loss gauges the difference between the predicted and actual values, but penalizes insufficiently confident predictions.

$$L(y, \hat{y}) = \sum_{i=0}^{N}\left(\max\left(0, 1 - y_i \cdot \hat{y}_i\right)^2\right) \tag{2}$$

where y is the true label (1 for positive class, -1 for negative class) and $\hat{y}_i$ is the predicted score (a continuous value). Also, Hinge loss is used for binary classification problems and penalizes predictions that are not confident enough. The hinge loss takes the value of 0 when the prediction is correct and confident ($y_i * \hat{y}_i$ ¿ 1), and it takes a positive value otherwise, proportional to the difference between the prediction and the true label.The objective of the training optimization procedure is to reduce hinge loss so that predictions are as precise and confident as feasible [59].

Cross-Entropy Loss: Cross-entropy loss, also known as categorical cross-entropy loss, is a typical loss function used for multi-class classification problems in which the objective is to predict a categorical label from many probable classes. The cross-entropy loss quantifies the disparity between the expected probability distribution and the actual class label [61]. Given the expected probabilities, cross-entropy loss is defined as the negative log-likelihood of the true class. For a single sample, the formula for cross-entropy loss is as follows:

$$\text{Loss } = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i \tag{3}$$

where y_i is a one-hot encoded vector representing the true class, $\hat{y}_i$ is the sample's estimated probability distribution, and log is the natural logarithm. Cross-entropy loss quantifies the disparity between the expected probability distribution and the actual class label. The loss approaches infinite as the anticipated probability of the correct class drops from 1 to 0. This indicates that the training optimization process assigns high probability to the correct class and low probabilities to all other classes [61]. In practice, the predicted probabilities are usually obtained using a softmax activation function applied to the output of the network Figure 14. The softmax function takes as input a vector of real-valued scores and returns a probability distribution over the classes. The softmax function is defined as:

$$S\left(y_i\right) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{4}$$

where $S(y_i)$ is the predicted probability of the real outputs y_i.

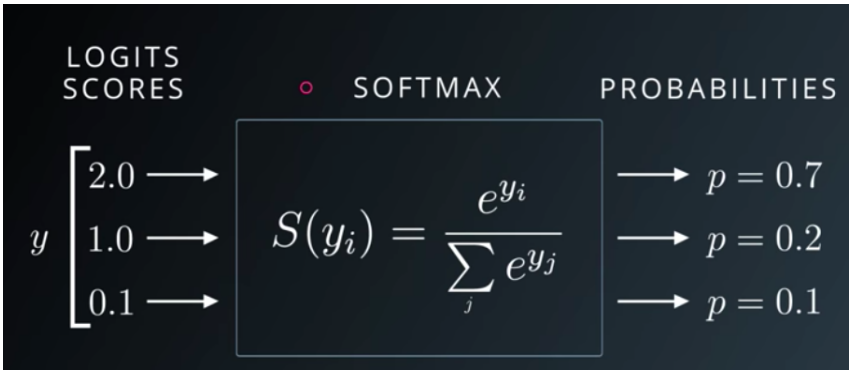


Figure 14: Compute probabilities with Softmax

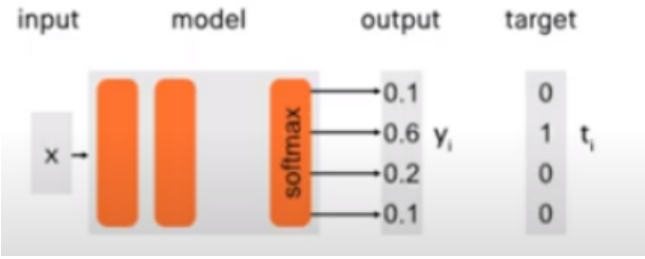


Figure 15: If an example belongs to a class it will have a target probability of 1 and all the others 0, in other words 1 is being assigned to the highest probability output that usually is computed by the softmax activation function.

In conclusion, the objective of the training optimization procedure is to minimize the cross-entropy loss such that the predicted probabilities are as near as feasible to the real class label. Reducing the cross-entropy loss guarantees that the network will predict high probability for the correct class and low probabilities for alternative classes.

2.6 Optimizers

Optimizers are a crucial component of neural network training. Optimizers are algorithms that change network parameters using gradient-based optimization. The gradient of the loss function with respect to the network parameters indicates the direction in which the loss function descends most steeply. The optimizer seeks to minimize the loss function by changing parameters in the direction of the gradient. There are several sorts of optimizers, each with its own advantages and disadvantages. Some of the most popular optimizers are [62]:

- Stochastic Gradient Descent: This is the most basic and elementary optimizer. The parameters are updated by subtracting a tiny multiple of the gradient from the gradient of the loss with respect to the parameters. The size of the update step is determined by the learning rate.
- Momentum: Momentum optimizer builds on SGD by adding a momentum term. The momentum term helps the optimizer overcome saddle points and converge faster in practice.
- Nesterov Accelerated Gradient (NAG): NAG is a variation of Momentum that computes the gradient update using a different method. It utilizes the gradient of the future location of the parameters rather than the gradient of their present position, resulting in enhanced convergence.
- Adagrad: Adagrad is an optimizer that adjusts the learning rate independently for each parameter. The learning rate for each parameter is updated based on the historical gradient information for that parameter, with parameters with high gradient magnitude receiving smaller updates.
- Adadelta: Adadelta is an extension of Adagrad that replaces the accumulation of historical gradient information with the accumulation of historical updates. This results in a more stable and efficient optimization process.
- RProp: RProp is a gradient-based optimizer that modifies parameters according to the sign of the gradient. The learning rate for each parameter is updated independently and is limited to an interval.
- Adam: Adam is a gradient-based optimizer that combines the ideas of Momentum and RProp. Using a moving average of gradient information, the learning rate for each parameter is dynamically adjusted.

The choice of optimizer is dependent on the nature of the issue and the network in use. Adam is a solid default option for the majority of issues, although other optimizers may perform better in particular situations. It is essential to experiment with various optimizers and learning rates to determine the optimal combination for a given task.

Optimizers have several hyperparameters that control their behavior. The learning rate, which defines the size of the update step, is the most significant hyperparameter. Other hyperparameters include the momentum factor for Momentum and NAG, the decay rate for Adagrad and Adadelta, and the learning rate updates for RProp and Adam.

2.6.1 Adam optimizer

As in the previous sections, here we are going to analyze more the optimizer that we have used in this thesis i.e the Adam optimizer.

Adam is a gradient-based optimization approach used extensively in deep learning. Adam stands for Adaptive Moment Estimation. It was presented by Kingma and Ba in a 2014 publication, and it has subsequently become a popular default option for many deep learning tasks.

The Adam optimizer combines the ideas of Momentum optimization and RProp (Resilient Backpropagation) optimization to dynamically adapt the learning rates of each parameter in the network. The adaptive learning rates for each parameter are estimated using two exponential moving averages, one for the first instant (mean) of the gradients and another for the second moment (uncentered variance) [63].

Given a parameter w and its gradient g, the updates to the parameters are computed as follows [64]:

1. Initialize the exponential moving averages for the gradients' mean (m) and variance (v) with 0.
2. For each time step t:
   (a) Compute the slope of the loss in relation to w i.e $\frac{\partial L}{\partial w_t}$
   (b) Update the moving averages of the mean and variance of the gradients as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\partial L}{\partial w_t} \right] \tag{1}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2 \tag{2}$$

   (c) Correct the bias in the moving averages:

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{3}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{4}$$

   (d) Update the parameters

$$w_{t+1} = w_t - \widehat{m_t} \left(\frac{\alpha}{\sqrt{\widehat{v_t}} + \varepsilon} \right) \tag{5}$$

where:

- a is the learning rate, which determines the update's step size.
- β_1 and β_2 controls the decay rates of moving averages using hyperparameters.
- ε to avoid division by zero, a minor constant is added to the denominator.
- ∂L is the derivative of the Loss
- ∂W_t derivative of weights at time t
- w_t weights at time t
- v_t sum of square of past gradients i.e sum(∂L/∂Wt $-$ 1) initially $v_t = 0$
- m_t is aggregate of gradients at time t initially, $m_t = 0$.

The Adam optimizer has several advantages over other optimization algorithms. It is computationally efficient, as it only requires computing moving averages and element-wise divisions. Since the learning rates are adaptively changed for each parameter based on its previous gradient information, it is also well-suited for high-dimensional issues. Furthermore, Adam has been shown to excel in a wide array of deep learning tasks and is a good default choice for many problems. In the below Figure 16 we can see its comparative analysis among other popular optimizer algorithms.

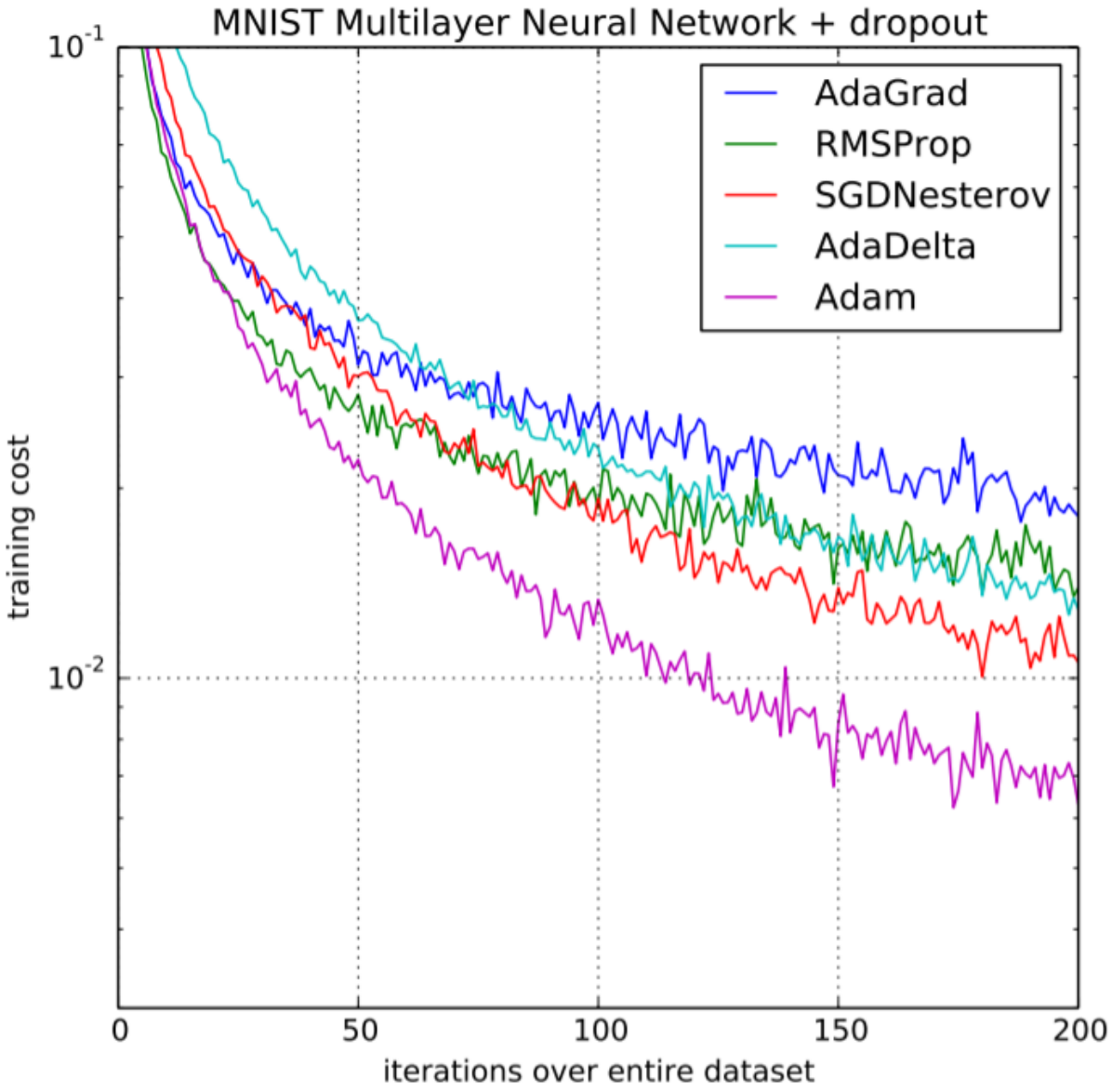


Figure 16: Performance analysis of various popular optimization algorithm. We can see that Adam outperforms for the MNIST dataset. (image source: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/)

2.7 Data preprocessing - one-hot encoding

Data preprocessing is an essential phase in the training of neural networks, since it has a significant influence on the model's performance. The transformation of categorical variables into numerical representations that may be utilized as input to the neural network is an essential part of data preprocessing. In multiclass classification issues, where the target variable has more than two potential values, one-hot encoding is a standard approach used to translate categorical variables into numerical representations [65], and we used this strategy in our dissertation.

One-hot encoding entails transforming categorical data into a numerical format, with each category represented by a binary vector. The categorical variable in a multiclass classification issue is represented by a matrix of binary values, where each row corresponds to a distinct category and each column corresponds to a unique class. The matrix's binary values reflect the existence or absence of a certain category inside a specific class [65].

For example, consider a multiclass classification problem where the target variable has three possible classes: "dog", "cat", and "horse". If the categorical variable has two possible categories, "domestic" and "wild", the one-hot encoding of the variable would look like this:

#	Class	Domestic	Wild
0	Dog	1	0
1	Cat	1	0
2	Horse	0	1

Table 1: One-hot encoding technique applied in multiclass classification problems.

Each class is represented by a unique binary vector and each category by a unique column in this representation. A "1" in the appropriate cell indicates the existence of a given category inside a particular class, whereas a "0" indicates the lack of the category.

One-hot encoding has several advantages for neural network training. **Firstly**, it eliminates the problem of ordering or magnitude of the categorical variables, which can sometimes cause issues when using other encoding methods, such as label encoding. **Secondly**, one-hot encoding provides a clear and concise representation of the categories and classes, making it easy to interpret the results of the neural network.

However, there are also some limitations to one-hot encoding. In the first place, it might increase the dimensionality of the data, which can be computationally expensive and may cause overfitting. Secondly, one-hot encoding can lead to sparse data representations, where the majority of matrix entries are zero, which can cause problems for some neural network algorithms. To address these limitations, various alternative encoding methods, such as embeddings and ordinal encoding, have been developed.

2.8 Section References

34. Brownlee, J. (2021). Weight initialization for deep learning neural networks. Machine Learning Mastery, 7.

35. Why Do I Get Different Results Each Time in Machine Learning?, Jason Brownlee, https://machinelearningmastery.com/different-results-each-time-in-machine-learning/

36. What Does Stochastic Mean in Machine Learning?, Jason Brownlee, https://machinelearningmastery.com/stochastic-in-machine-learning/

37. Robbins, H., & Monro, S. (1951). A stochastic approximation method. The annals of mathematical statistics, 400-407.

38. Breiman, L. Statistics Department University of California Berkeley, CA 94720. 2001. RANDOM FORESTS.

39. Bottou, L. (2012). Stochastic gradient descent tricks. Neural Networks: Tricks of the Trade: Second Edition, 421-436.

40. Brownlee, J. (2020). Why do I get different results each time in machine learning?. Machine Learning Mastery, 17.

41. Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. Towards Data Sci, 6(12), 310-316.

42. Simeon Kostadinov,Understanding Backpropagation Algorithm, https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

43. Tina Jacob,Vanishing Gradient Problem, Explained, https://www.kdnuggets.com/2022/02/vanishing-gradient-problem.html

44. Chen, M. (2022). Vanishing Gradient Problem in training Neural Networks.

45. Parhi, R., & Nowak, R. D. (2020). The role of neural network activation functions. IEEE Signal Processing Letters, 27, 1779-1783.

46. He, J., Li, L., Xu, J., & Zheng, C. (2018). Relu deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973.

47. Da Silva, I. N., Hernane Spatti, D., Andrade Flauzino, R., Liboni, L. H. B., dos Reis Alves, S. F., da Silva, I. N., ... & dos Reis Alves, S. F. (2017). Artificial neural network architectures and training processes (pp. 21-28). Springer International Publishing.

48. Li, F., Zurada, J. M., & Wu, W. (2018). Smooth group L1/2 regularization for input layer of feedforward neural networks. Neurocomputing, 314, 109-119.

49. Doan, C. D., & Liong, S. Y. (2004, July). Generalization for multilayer neural network bayesian regularization or early stopping. In Proceedings of Asia Pacific association of hydrology and water resources 2nd conference (pp. 5-8).

50. Chen, D., & Hagan, M. T. (1999, July). Optimal use of regularization and cross-validation in neural network modeling. In IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339) (Vol. 2, pp. 1275-1280). IEEE.

51. Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8(4), e1249.

52. Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of big data, 6(1), 1-48.

53. Kadhim, Z. S., Abdullah, H. S., & Ghathwan, K. I. (2022). Artificial Neural Network Hyperparameters Optimization: A Survey. International Journal of Online & Biomedical Engineering, 18(15).

54. Walczak, S. (2001). An empirical analysis of data requirements for financial forecasting with neural networks. Journal of management information systems, 17(4), 203-222.

55. Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2008, December). RUSBoost: Improving classification performance when training data is skewed. In 2008 19th international conference on pattern recognition (pp. 1-4). IEEE.

56. Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2008, December). RUSBoost: Improving classification performance when training data is skewed. In 2008 19th international conference on pattern recognition (pp. 1-4). IEEE.

57. Schilling, F. (2016). The effect of batch normalization on deep convolutional neural networks.

58. Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. Advances in neural information processing systems, 31.

59. Yalcin, O. G., & Yalcin, O. G. (2021). Deep learning and neural networks overview. Applied Neural Networks with tensorflow 2: API oriented deep learning with python, 57-80.

60. Díaz-Vico, D., Fernández, A., & Dorronsoro, J. R. (2021, September). Companion losses for deep neural networks. In Hybrid Artificial Intelligent Systems: 16th International Conference, HAIS 2021, Bilbao, Spain, September 22-24, 2021, Proceedings (pp. 538-549). Cham: Springer International Publishing.

61. Semenov, A., Boginski, V., & Pasiliao, E. L. (2019). Neural networks with multidimensional cross-entropy loss functions. In Computational Data and Social Networks: 8th International Conference, CSoNet 2019, Ho Chi Minh City, Vietnam, November 18-20, 2019, Proceedings 8 (pp. 57-62). Springer International Publishing.

62. Pavan Vadapalli, Types of Optimizers in Deep Learning Every AI Engineer Should Know, https://www.upgrad.com/blog/types-of-optimizers-in-deep-learning/

63. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, Jason Brownlee, GentleIntroductiontotheAdamOptimizationAlgorithmforDeepLearning

64. Intuition of Adam Optimizer, https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

65. Why One-Hot Encode Data in Machine Learning?, Jason Brownlee, https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

3 Hermite Polynomials

In this part, we describe and discuss some of the fundamental features of Hermite polynomials that represent the core of this thesis. (for a more in-depth explanation of the theoretical foundation of Hermite polynomials, see section 6) In addition, we will discuss some related works that have used Hermite (or other polynomial types such as Legendre) polynomial as a potential activation function candidate within structure of neural networks to enhance network performance (the same goal as our proposed structure, which is described analytically in Section 6).

3.1 Introduction

Hermite polynomials are a set of orthogonal polynomials that are commonly used in physics and engineering. They are named after Charles Hermite and are defined as the Hermite differential equation solutions [66]:

$$H''(x) - 2xH'(x) + 2nH(x) = 0 \tag{1}$$

where $H(x)$ is the n^{th} degree Hermite polynomial and x is the independent variable. The polynomials are orthogonal with respect to the weight function e^{-x^2}.

The first few Hermite polynomials are:

$$\begin{aligned} H0(x) &= 1 \\ H1(x) &= 2x \\ H2(x) &= 4x^2 2 - 2 \\ H3(x) &= 8x^3 3 - 12x \\ H4(x) &= 16x^4 4 - 48x^2 2 + 12 \end{aligned} \tag{2}$$

Hermite polynomials have many important properties and applications in physics, quantum mechanics, and engineering. In quantum physics, they are utilized in the solution of the Schrodinger equation and the study of harmonic oscillators. They also emerge in the study of statistical mechanics and Brownian motion theory [67]. Some of the advantages of Hermite polynomials are [66]:

1. **Orthogonality**: Hermite polynomials are appeared to be orthogonal regarding the weight function. e^{-x^2}, which means that they satisfy the property:

$$\int (\mathrm{Hn}(x)\,\mathrm{Hm}(x))e^{\wedge}\,(-x^{\wedge}2)\,dx = 0 \tag{3}$$

   for $n \neq m$. This orthogonality property allows for easy calculation of integrals involving Hermite polynomials.

2. **Completeness**: Hermite polynomials are a complete set of functions, therefore any function satisfying the weight function and boundary conditions may be expressed as a linear combination of Hermite polynomials. This virtue of completeness renders Hermite polynomials helpful for solving differential equations, such as the Schrodinger equation in quantum physics.

3. **Recurrence relations:** The simple recurrence relation is met by Hermite polynomials is:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_n - 1(x), \tag{4}$$

   This recurrence relation allows to calculate the polynomials for any value of n, given the first few polynomials and also gives the opportunity to apply computational algorithms (programming for loops) that contains Hermite polynomials.

4. **Symmetry properties:** Hermite polynomials contain symmetries that make them helpful for solving certain kinds of differential equations and for studying statistical mechanics. Even-degree Hermite polynomials are even functions, while odd-degree Hermite polynomials are odd functions. Thus, they may be utilized to solve differential equations involving these forms of symmetry. For instance, if an x-axis differential equation is symmetric, the solution can be stated as a linear combination of even Hermite polynomials, and if it is antisymmetric, the solution may be expressed as a linear combination of odd Hermite polynomials. Also, Hermite polynomials are used in the study of statistical mechanics, and their symmetry properties are useful in this field. In the study of the harmonic oscillator, for instance, the symmetry properties of Hermite polynomials are used to calculate the statistical properties of the system, such as the probability distribution of the position and momentum of a particle. Furthermore, as we mentioned Hermite polynomials may be used to solve partial differential equations (PDEs) with symmetry properties. For instance, the heat equation and the wave equation are symmetric with respect to the x-axis, hence their solutions may be expressed in terms of even or odd Hermite polynomials. In quantum mechanics, the symmetry properties of Hermite polynomials are used to classify the

energy eigenstates of a harmonic oscillator used to determine the parity of the energy eigenstates and the wave functions of the system [68]. Finally, in engineering Hermite polynomials are used in control systems, signal processing, and numerical analysis for example for the design of optimal filters, control laws and other algorithms.

3.2 Related works

3.2.1 "Constructive feedforward neural networks using Hermite polynomial activation functions"

In this article, a constructive OHLs network is given in which the activation function of each hidden unit is a different polynomial. In the construction of the network, structure-level and function-level adaptation mechanisms are employed. Functional level adaptation guarantees that the "developing" or constructive network has distinct activation functions for each neuron, allowing it to more efficiently capture the underlying input-output map. The recommended activation functions are orthonormal Hermite polynomials. Comprehensive simulations demonstrate that the proposed network outperforms existing networks with equal sigmoidal activation properties [69]. In the below Figure 17 the proposed neural network architecture is presented:

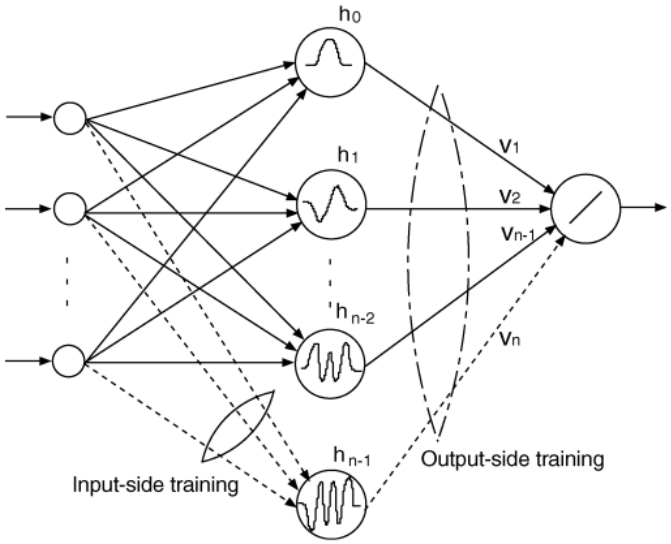


Figure 17: A constructive OHL-FNN employing orthonormal Hermite polynomials as activation functions for its hidden units.

In summary, the proposed FNN algorithm is contain the below four steps (for more details see [make reference] p.5):

1. Initialize the network with a single hidden layer and activation h_0 (hermite zero order).
2. Just train the input-side weights of the n_{th} node (h_{n-1} hermite activation).
3. The output-side weights of all hidden units are being trained.
4. Assess the performance of the networks on the training dataset using the percent of variance unexplained (FVU) measure.

$$FVU = \frac{\sum_{j=1}^{P} \left(\hat{g}\left(\mathbf{x}^j\right) - g\left(\mathbf{x}^j\right)\right)^2}{\sum_{j=1}^{P} \left(g\left(\mathbf{x}^j\right) - \bar{g}\right)^2} \tag{1}$$

where $g()$ is the function that the FNN is try to approximate, $\hat{g()}$ is the estimated output after the training and $\bar{g()}$ is the mean value of $g()$.

Authors of [69] implemented couple of simulations for regression and classification tasks (Cancer and two-category) point out the outperforming performance of their algorithm.

3.2.2 "Hermitian Polynomial for Speaker Adaptation of Connectionist Speech Recognition Systems"

Using model modification strategies, automatic speech recognition (ASR) systems may reduce the typical gap between training and test settings. This study investigates the issue of performance degradation when switching from speaker-dependent (SD) to speaker-independent (SI) conditions for connectionist (or hybrid) hidden Markov model/artificial neural

network (HMM/ANN) systems in the context of large vocabulary continuous voice recognition (LVCSR). Adapting hybrid HMM/ANN systems to a little amount of adaptation data has proven to be a difficult task, which has hindered the broad use of hybrid techniques in operational ASR systems.

Consequently, addressing the crucial issue of speaker adaptation (SA) for hybrid HMM/ANN systems can have a significant impact on the connectionist paradigm, which will play an important role in the design of next-generation LVCSR in light of the great success reported by deep neural networks—ANNs with many hidden layers that employ the pre-training technique—on a variety of speech tasks. Existing adaptation solutions for ANNs based on injecting an adaptable linear transformation network linked to the input or output layer are unsuccessful, especially when dealing with a little amount of adaptation data, such as a single adaptive syllable. This study provides a novel method for overcoming these limits and making it resistant to limited adaption resources. Modify the hidden activation functions rather than the network weights. Using Hermitian activation functions makes this a possibility. Experimental findings on an LVCSR test demonstrate the effectiveness of the proposed technique. in the below Figure 18 [70].

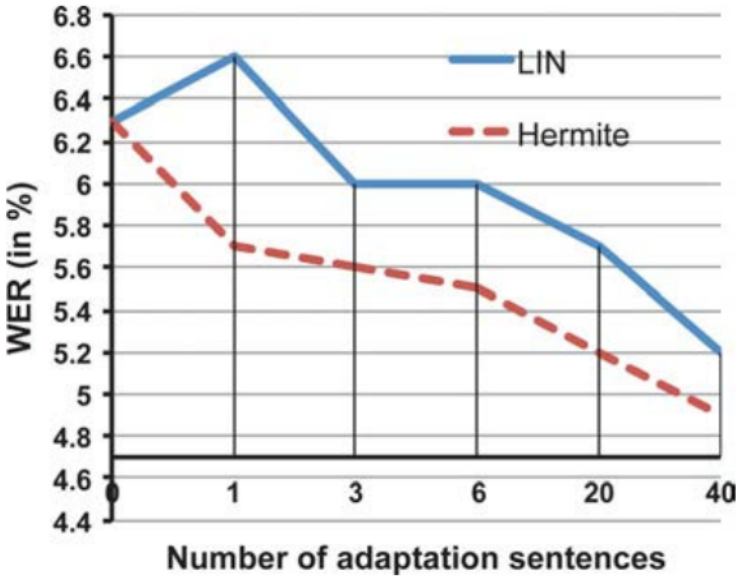


Figure 18: Impact of adaption data quantity on the WER of the SA ASR system. The dashed line represents the ASR system based on Hermite, whereas the solid line represents the LIN system. The WER with zero modified sentences represents the performance of the SI system.(source [70])

3.2.3 "Constructive Hermite Polynomial Feedforward Neural Networks with Application to Facial Expression Recognition"

Since the 1970s, computer-based facial expression recognition has been a significant area of research. Intelligent interaction between humans and machines is the ultimate goal. Recent research has shown that constructive One-Hidden-Layer Feedforward Neural Networks (OHL-FNNs) are effective at identifying facial expressions. Generally speaking, the activation functions of the hidden units in a FNN are identical sigmoidal functions. Nevertheless, it has not been shown that using the same activation functions for all hidden units results in the most optimal or efficient network architecture. In this paper, a new polynomial OHL-FNN is proposed for pattern recognition. As activation functions for the buried units, standard Hermite polynomials will be used. Each time a new hidden unit is added to the network, its activation function will be a Hermite polynomial with an increased order. Using 2-D DCT on the whole face image and then feeding the reduced 2-D DCT coefficients to the constructive network training, the proposed method is used to the facial expression detection problem. In addition, the merits and limitations of the constructive polynomial OHL-FNN for pattern recognition are emphasized [71].

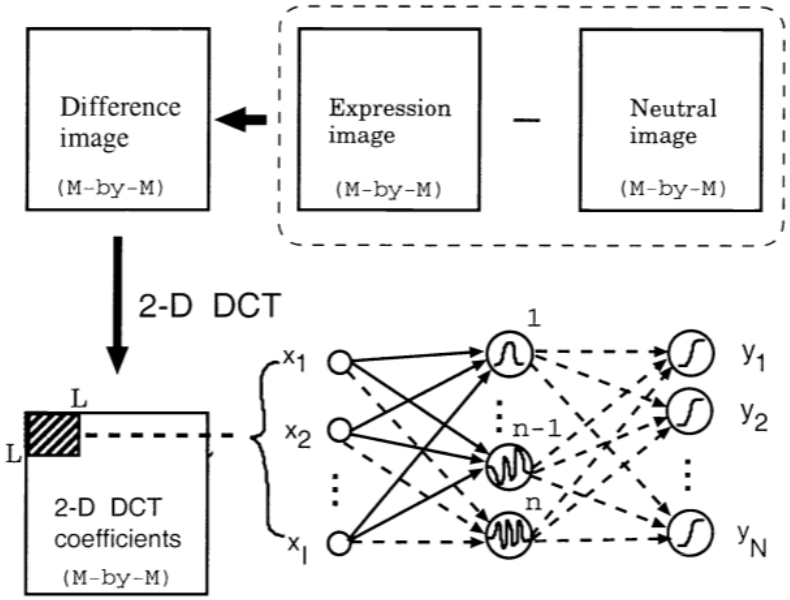


Figure 19: OHL-FNN facial expression recognition application.

3.2.4 "Hermite/Laguerre Neural Networks for Classification of Artificial Fingerprints From Optical Coherence Tomography"

In this study, the authors classify actual and synthetic fingerprints using forward (FNN), Hermite (HNN), and Laguerre (LNN) neural networks with optical coherence tomography pictures. Employing a self-organizing map (SOM) following Gabor edge detection of optical coherence tomography (OCT) images of fingerprints and material surfaces produced the best classification performance in comparison to moments based on color, texture, and shape. The FNN and HNN fared similarly, however the LNN performed poorly with a small number of hidden nodes, but beat the FNN and HNN when $n = 10$ was approached [72]. Simulation results are shown below:

	FNN		HNN		LNN	
Nodes	Moments (9)	Gabor SOM(2)	Moments (9)	Gabor SOM(2)	Moments (9)	Gabor SOM(2)
2	0.82	0.95	0.76	0.95	0.45	0.46
3	0.92	0.99	0.86	0.99	0.62	0.67
4	0.95	1.00	0.92	1.00	0.82	0.84
5	0.96	1.00	0.93	1.00	0.89	0.94
6	0.97	1.00	0.95	1.00	0.94	1.00
7	0.96	1.00	0.93	1.00	0.97	1.00
8	0.96	1.00	0.95	1.00	0.97	1.00
9	0.96	1.00	0.96	1.00	0.98	1.00
10	0.97	1.00	0.96	1.00	0.98	1.00

(9) - Denotes 9 moments from color, texture, shape moments.
(2) - Denotes 2 features from SOM run on Gabor edge detection values.

Figure 20: The classification performance (accuracy) for ten 10-fold cross validation as a function of the number of hidden nodes in the hidden layer of FNN, HNN, and LNN. (source [72]).

3.2.5 "Application of Legendre Neural Network for solving ordinary differential equations" [11]

Utilizing a Legendre Neural Network (LeNN) model with a single layer, this research provides a novel way for tackling initial value and boundary value issues. LeNN is a model of artificial neural network that employs Legendre polynomials to extend the input pattern, hence eliminating the requirement for a hidden layer. Nonlinear singular initial value problems, coupled ordinary differential equations, and boundary value problems are used to assess the proposed technique.

The research utilizes the error backpropagation algorithm to update the network parameters (weights) and compares the achieved results to those of current techniques. The results demonstrate that the suggested method is dependable and in excellent accord with the current techniques. This study proposes a novel strategy for addressing starting and boundary value issues that has the potential to be used in several scenarios.

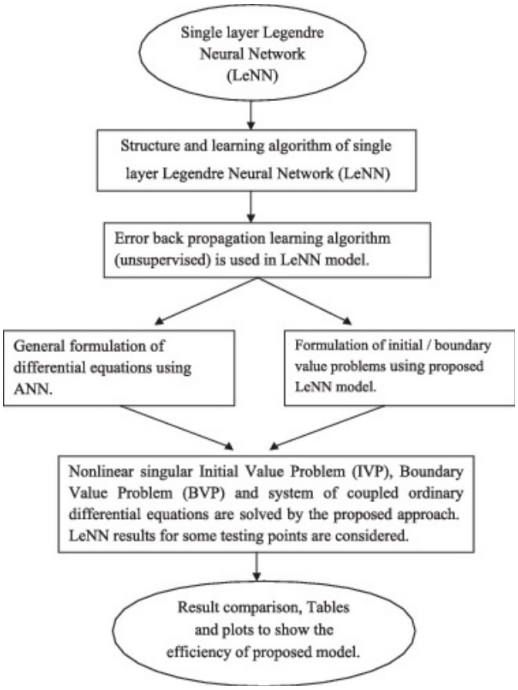


Figure 21: A Legendre Neural Network (LeNN) model with a single layer for solving differential equations. (source [11]).

3.2.6 "Y. Zhang, Y. Yin, D. Guo, X. Yu, and L. Xiao Cross-validation based weights and structure determination of Chebyshev-polynomial neural networks for pattern classification" [14]

This study presents a novel neural network architecture for pattern classification called a single-output Chebyshev-polynomial feed-forward neural network (SOCPNN). This neural network type is then used as the basis for a new multi-output feed-forward Chebyshev-polynomial neural network (MOCPNN). The study demonstrates that both the SOCPNN and MOCPNN neural networks have a lower computational cost than the commonly used multi-layer perceptron (MLP) neural network, while providing superior performance.

The study also proposes two weight-and-structure-determination (WASD) algorithms, one for the SOCPNN and one for the MOCPNN, which can determine the weights and structures of neural networks automatically and efficiently. The purpose of the algorithms is pattern classification.

In addition, the suggested SOCPNN and MOCPNN have been assessed on several real-world classification datasets, with and without the inclusion of noise. The results indicate that the proposed neural networks have a high degree of accuracy, and that the MOCPNN is particularly robust in pattern classification when augmented with the WASD algorithms. This study proposes a unique technique for pattern classification based on the neural networks SOCPNN and MOCPNN, which have a lower computational cost and higher performance than MLPs.

3.2.7 "G. E. Tsekouras, V. Trygonis, A. Maniatopoulos, A. Rigos, A. Chatzipavli, J. Tsimikas, N. Mitianoudis, and A. F. Velegrakis A Hermite neural network incorporating artificial bee colony optimization to model shoreline realignment at a reef-fronted beach" [10]

This study examines the use of an unique neural network, the Hermite polynomial neural network, to forecast shoreline realignment along an urban beach that is fronted by a very irregular beachrock reef. The neural network generates Hermite-truncated polynomial series of linear combinations of reef morphology and wave force-related input variables. A coastal video monitoring system outputs a time series of shoreline locations collected with great spatial and temporal accuracy.

The research explains how the Hermite polynomial neural network may estimate with arbitrary accuracy any continuous function given on a compact subset of the multidimensional Euclidean space. The study compares the network to three other neural networks that were tweaked using different swarm intelligence-based techniques.

The research demonstrates that the Hermite network with polynomial orders 3 and 4 can effectively handle the nonlinear effects caused by the presence of the reef. Base on the performance criteria and the exhaustive parametric statistical analysis, the suggested network outperforms the other networks evaluated. The research concludes by

proposing that the effectiveness of the model may be enhanced by removing beach portions behind reef inlets and/or certain large reef sections that bring substantial shoreline variability. Overall, this study provides a unique approach for modeling shoreline realignment based on a Hermite polynomial neural network that can handle the complexity of the uneven beachrock reef and generate accurate predictions of shoreline placement.

3.2.8 "Wang, J., Chen, L., & Ng, C. W. W. A New Class of Polynomial Activation Functions of Deep Learning for Precipitation Forecasting" [120]

This study focuses on employing polynomial activation functions to improve precipitation forecasting using deep learning algorithms. The authors argue that despite the success of deep learning models in a range of applications, they work largely via image processing and overlook the nature of physical systems. The authors suggest employing neural networks that contains polynomial activation functions, which have been used in theory-driven models for numerical approximation and modeling of dynamic systems. They caution, however, that polynomials are too brittle for stable training and may generate severe data flow explosion issues. To solve these concerns, they use Chebyshev polynomials and a novel normalizing technique known as Range Norm to improve the robustness of training. Using synthetic datasets and a summer precipitation prediction assignment, the performance of the model is confirmed, demonstrating that polynomial activation functions may be a helpful tool for deep learning in modeling complicated physical systems and for automated theoretical physics analysis.

3.2.9 "Siniscalchi, S. M., & Salerno, V. M Adaptation to new microphones using artificial neural networks with trainable activation functions" [121]

This work analyzes several microphone-specific hidden unit contribution adjustment algorithms for automatic speech recognition (ASR) systems. Owing to microphone mismatch in the training and test data, there are speech spectrum changes that need this correction. The authors investigate four strategies for adapting the model, including the use of Hermite activation functions, the introduction of bias and slope parameters in the sigmoid activation functions, the injection of an amplitude parameter specific to each sigmoid unit, and a combination of the latter two. Crucial for large-scale online deployment, these adaptation approaches let the updated model to be stored in a little amount of space. Tests reveal that the recommended solutions increase word error rates on the Wall Street Journal corpus's standard Spoke 6 challenge compared to unadapted ASR systems. When just one phrase is offered for adaptation, the suggested adaptation processes are also effective. The authors note that the proposed solutions surpass standard multicondition training and are comparable to conventional linear regression-based approaches while using up to 15 orders of magnitude less parameters.

3.2.10 "Hsu, C. F. Intelligent control of chaotic systems via self-organizing Hermite-polynomial-based neural network" [122]

This article explains the adaptive self-organizing Hermite-polynomial-based neural control (ASHNC) system. Two components make up the system: a neural controller and a supervisor compensator. Using a self-organizing Hermite-polynomial-b neural network (SHNN), the neural controller approximates an ideal feedback controller. The SHNN employs a self-organizing technique that is efficient, has a high convergence accuracy, and has a short convergence time. The supervisor compensator is intended to remove the approximation error between the neural controller and the ideal feedback controller without the occurrence of chattering issues. In addition, a proportional-integral (PI) type adaptation rule is constructed from the Lyapunov stability theory in order to ensure system stability and accelerate convergence of the tracking error. The developed ASHNC system is then applied to a chaotic system, and simulation results indicate effective control performance. Using a mix of neural network and control theory methodologies, the ASHNC system looks to be an efficient means of regulating complicated systems.

3.2.11 "Panigrahi, A., Shetty, A., & Goyal, N.,Effect of activation functions on the training of overparametrized neural nets" [123]

This article investigates the impact of activation functions on the training of two-layer neural networks that are significantly overparameterized. With adequate hyperparameter settings, overparameterized neural networks may fast achieve modest training error, according to this text. Subsequent publications have proven this assumption theoretically, however these conclusions either require that the activation function is ReLU or rely on the least eigenvalue of a particular Gram matrix based on the data, initialization, and activation function. On the empirical side, the paragraph states that there are a variety of different activation functions that appear to perform better than ReLU in some contexts, but their effect on training remains unclear. This research presents theoretical findings about the impact of activation functions on the training of significantly overparameterized two-layer neural networksnetworks. The smoothness or roughness of an activation function is the determining factor in its performance. Under minimum data assumptions, all eigenvalues of the associated Gram matrix are big for nonsmooth activations such as ReLU,

SELU, and ELU. With smooth activations like tanh, swish, and polynomials, the issue is more complicated, and the lowest eigenvalue of the Gram matrix may be tiny, resulting in delayed training if the subspace encompassed by the data has a small dimension. Nonetheless, the eigenvalues are enormous if the dimension is large and the data satisfy another moderate criterion. The findings have several applications and expansions.

3.3 Section References

66. Szeg, G. (1939). Orthogonal polynomials (Vol. 23). American Mathematical Soc

67. Belafhal, A., Hricha, Z., Dalil-Essakali, L., & Usman, T. (2020). A note on some integrals involving Hermite polynomials and their applications. Adv. Math. Mod. Appl., 5(3), 313-319.

68. A. Eremenko, Some applications of Legendre and Hermite polynomials, Notes

69. Ma, L., & Khorasani, K. (2005). Constructive feedforward neural networks using Hermite polynomial activation functions. IEEE Transactions on Neural Networks, 16(4), 821-833.

70. Siniscalchi, S. M., Li, J., & Lee, C. H. (2013). Hermitian polynomial for speaker adaptation of connectionist speech recognition systems. IEEE Transactions on Audio, Speech, and Language Processing, 21(10), 2152-2161.

71. Ma, L., & Khorasani, K. (2001, November). Constructive Hermite polynomial feedforward neural networks with application to facial expression recognition. In Video Technologies for Multimedia Applications (Vol. 4520, pp. 31-42). SPIE.

72. Peterson, L. E., & Larin, K. V. (2008, December). Hermite/Laguerre neural networks for classification of artificial fingerprints from optical coherence tomography. In 2008 Seventh International Conference on Machine Learning and Applications (pp. 637-643). IEEE.

120. Wang, J., Chen, L., & Ng, C. W. W. (2022, February). A New Class of Polynomial Activation Functions of Deep Learning for Precipitation Forecasting. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (pp. 1025-1035)

121. Siniscalchi, S. M., & Salerno, V. M. (2016). Adaptation to new microphones using artificial neural networks with trainable activation functions. IEEE transactions on neural networks and learning systems, 28(8), 1959-1965

122. Hsu, C. F. (2014). Intelligent control of chaotic systems via self-organizing Hermite-polynomial-based neural network. Neurocomputing, 123, 197-206

123. Panigrahi, A., Shetty, A., & Goyal, N. (2019). Effect of activation functions on the training of overparametrized neural nets. arXiv preprint arXiv:1908.05660

4 Classification in machine learning

4.1 Introduction

Classification is a supervised machine learning problem with the goal of predicting a categorical label or class based on a collection of input attributes. It is one of the most widespread and widely used approaches in the area of machine learning and has a number of practical applications, such as categorizing images, filtering emails, and detecting credit card fraud. [73].

Classification algorithms use as input a collection of labeled training data, where the input characteristics and class labels are known. The method then utilizes this training data to develop a mapping between the input features and the class labels, and may be used to unseen data to create predictions. Metrics like as accuracy, precision, recall, and F1-score may be used to measure the correctness of the predictions. [73,74].

There are several classification techniques, which include linear classifiers, decision trees, k-nearest neighbors, Naive Bayes, and artificial neural networks. Each sort of algorithm has its own strengths and disadvantages, and the algorithm used relies on the nature of the issue at hand and the type of data being processed. [75,77].

Linear classifiers, such as Logistic Regression, are simple and fast algorithms that are used for binary classification problems. The link between the input data and the class label is modeled as a linear combination, and the objective is to identify the coefficients of the linear model that best distinguish between the two classes.

Decision trees are another common categorization algorithm type. They function by iteratively subdividing the data into smaller subgroups depending on the supplied feature values. Each node reflects a decision based on the values of one or more input attributes, and each leaf contains the projected class labels. Decision trees are easy to comprehend and analyze, but overfitting may occur if the tree is too complicated.

K-nearest neighbors a classification non-parametric algorithm makes predictions based on the class labels of the k-nearest neighbors in the training data. It is a simple and flexible algorithm that is particularly useful for complex data with non-linear relationships between the input features and class labels.

Naive Bayes Bayes' theorem asserts that the probability of a class label given the input features is equal to the sum of the prior probability of the class label and the likelihood of the input features given the class label. Naive Bayes is an efficient and straightforward technique that is especially beneficial for text categorization issues.

Artificial neural networks are a sort of machine learning algorithm whose form and operation are inspired by the human brain. They comprise of nodes, or artificial neurons, that process and send information and are linked. Neural networks may be used for both binary and multi-class classification issues, and they are especially effective for complicated data with non-linear correlations between input variables and class labels.

In general, classification is a basic topic in machine learning and has many applications. The selection of a classification method is dependent on the situation at hand and the data type used. Whether employing a basic linear classifier or a complicated neural network, the objective is to discover the optimal model for your data that predicts the class labels of unseen data with high accuracy.

4.2 Artificial Neural Networks in classification

This thesis is centered on neural networks, hence in this part we will examine in depth their role in classification problems [76].

Neural networks have been used in a variety of applications, such as image classification, voice recognition, and natural language processing, and have been shown to be especially successful in difficult and high-dimensional classification problems.

Both binary and multi-class classification issues are amenable to neural networks. In binary classification, the objective is to predict one of two class labels, while the objective of multi-class classification is to predict one of numerous class labels. The neural network's output is a probability distribution across all potential class labels, with the predicted class label having the greatest probability.

Feedforward neural networks, convolutional neural networks, and recurrent neural networks may all be utilized for classification problems. Each form of architecture is intended to capture distinct data patterns, and the choice of architecture relies on the nature of the issue at hand and the type of data employed. Feedforward neural networks has been analyzed in the Section 2.

Convolutional neural networks, Convolutional neural networks, often known as convnets, are a sort of neural network design that is well-suited for image categorization applications. They are meant to identify local patterns in the picture data, such as edges, corners, and textures, and develop a hierarchy of characteristics that may be utilized to create accurate predictions. Convnets consist of numerous layers, including convolutional layers, pooling layers, and fully connected layers, and are trained by stochastic gradient descent, a kind of backpropagation.

Recurrent neural networks, RNNs, also known as recurrent neural networks, are a sort of neural network design that is well suited for sequential input, such as audio signals and text. They are meant to identify long-term relationships in the data and are used to develop an accurate representation of the input sequence. RNNs include a hidden state that is updated at each time step, and the final hidden state is utilized for prediction.

Neural networks are advantageous for classification problems because they can learn complicated non-linear input-output correlations., which makes them suitable for solving problems where traditional algorithms struggle. In classification tasks, the inputs are usually high-dimensional, which means that there are many variables that need

to be taken into consideration. Neural networks can handle this complexity by breaking down the inputs into a series of transformations that lead to a final output, using multiple hidden layers between the input and output layers. Each hidden layer may learn distinct characteristics that are pertinent to the classification job, and the network can subsequently generate predictions using these features. In addition, neural networks can learn from vast volumes of data, which makes them well-suited for situations where there is a significant quantity of accessible training data. Neural networks' capacity to learn from data and generalize to new cases makes them a potent categorization tool.

Consider the following picture classification example, in which the aim is to categorize an image as either "dog" or "cat."

In this instance, the input to the neural network would be the picture's pixel values, which might be called high-dimensional data (a 256x256 image would have 256x256x3 = 196608 input variables, where 3 is the number of color channels). In this high-dimensional environment, a conventional algorithm may have difficulty locating a suitable decision boundary.

A neural network, on the other hand, may turn a picture into a sequence of transformations, with each hidden layer learning to detect distinct task-relevant elements. For instance, the first hidden layer may learn to identify edges and basic forms, the second hidden layer may learn to recognize more complicated shapes such as the torso and legs of an animal, and the last hidden layer may learn to distinguish the head and face of an animal.

The network may then utilize these characteristics to produce its final prediction, for instance by calculating the likelihood that a picture belongs to each class using the learnt features. When additional instances are used to train the network, it may also continue to learn and increase its accuracy.

Its capacity to learn non-linear correlations and generalize to new instances makes neural networks an effective tool for picture classification, and similar techniques may be extended to other classification problems such as text classification and audio recognition.

Let's explore another example of a binary classification job, this time in the realm of medical diagnostics. The objective is to forecast whether a patient has a given condition based on their medical history and test findings.

In this case, the neural network's input variables may include the patient's age, gender, weight, blood pressure, and test results for different illness indicators. Similar to the previous example, a traditional algorithm might struggle to find a good decision boundary in this high-dimensional space.

In contrast, a neural network may learn non-linear correlations between inputs and disease state. For example, the network might learn that t he disease is more likely to occur in older patients and in those with high blood pressure, but that the disease is less likely to occur in patients who are female or have a low weight. The network can also learn to weigh the importance of each input variable, for example, finding that blood pressure is a stronger predictor of the disease than weight.

By learning these non-linear relationships, the neural network can make highly accurate predictions, even in cases where the relationships between the inputs and the disease status are complex and not easily modeled by traditional algorithms. This ability to learn from data makes neural networks a useful tool for medical diagnosis, and similar approaches can be applied to other binary classification tasks such as fraud detection and customer churn prediction.

We have to mention that there are several other advantages of neural networks in classification beyond handling non-linearity between inputs and outputs such as:

- **Scalability:** Neural networks can handle large amounts of data and can easily scale to more inputs and hidden layers as needed, making them well-suited for problems with high-dimensional inputs.
- **Flexibility:** Neural networks may be used to a vast array of classification tasks, including picture and text categorization, voice recognition, and medical diagnosis.
- **Automated feature extraction:** Neural networks can automatically extract meaningful characteristics from incoming data, without the requirement for human-engineered features. This makes them suitable for scenarios in which it is difficult to manually create excellent features.
- **Robustness to noise:** Even with a vast quantity of training data, neural networks may be trained utilizing methods such as dropout and early stopping to make them resistant to noise and overfitting.
- **Transfer learning:** Pretrained neural networks can be adjusted on smaller datasets to quickly solve new classification tasks, allowing organizations to leverage existing models and training data to solve new problems with less effort.

Overall, the combination of these advantages makes neural networks a powerful mechanism for tackling a broad range of classification jobs, and they continue to be a popular choice for practitioners in many fields.

4.3 Multiclass Classification [1]

Multiclass classification is a kind of supervised learning in which the goal variable or output variable contains more than two class labels. This means that instead of just two possible outcomes (e.g., positive or negative), there are multiple

[1] 77.Aly, M. (2005). Survey on multiclass classification methods. Neural Netw, 19(1-9), 2.

classes or categories that the input data can belong to. For example, classifying a given email as spam, important, or urgent, recognizing different species of animals, or assigning a movie to a genre (action, romance, comedy, etc.).

Advantages of using multiclass classification over binary classification are:

- **More nuanced predictions:** In binary classification, the output is limited to two categories, which may not accurately reflect the complexity of the problem at hand. In contrast, multiclass classification can predict one of multiple classes, providing a more nuanced and comprehensive understanding of the data.
- **Wider range of problems:** Binary classification is limited to problems where the target variable has only two possible outcomes. However, many real-world problems involve more than two categories (such as classifying a given email as spam, important, or urgent) In such cases, multiclass classification is a more appropriate choice.
- **Improved representativeness:** Many real-world problems involve more than two categories. For example, a study might classify patients into multiple disease categories based on symptoms, risk factors, and other characteristics. By using multiclass classification, the model can better represent the true structure of the problem and produce more accurate predictions.
- **Better handling of class imbalance:** When one class has disproportionately more samples than the other, class imbalance is a typical issue in classification. This might lead to biased predictions in binary classification. With multiclass classification, the imbalance may be dispersed over numerous classes, so minimizing its effect on predictions.
- **Enhanced interpretability:** In binary classification, it is often difficult to understand the reasons behind the predictions made by the model. However, in multiclass classification, the model can provide insights into which features or combinations of features are most important for each class, facilitating comprehension of the decision-making process and identify areas for improvement.

There are several techniques that are commonly used for multiclass classification:

- **One-vs-all** (also known as one-vs-rest) technique: This is a common method for handling classification issues involving several classes. In this method, a binary classifier is trained independently for each class, and the class with the greatest predicted probability is picked as the final prediction. If there are three classes A, B, and C, for instance, three binary classifiers will be trained, one for each pairwise comparison: class A vs classes B and C, class B versus classes A and C, and class C versus classes B and A. As the final forecast, the category with the greatest projected probability for each sample is chosen.
- **Softmax Regression** (also known as multinomial logistic regression): Softmax regression is another technique for solving multiclass classification problems. A single model is trained to predict the probabilities of all classes, and the class with the greatest probability is chosen as the final prediction. Softmax regression uses the softmax function to calculate the probabilities of all classes, ensuring that the probabilities sum up to 1.
- **Decision Trees and Random Forests:** Decision trees and random forests are popular categorization problem-solving methods. The target variable is partitioned in decision trees depending on the values of the input variables. The generated tree structure is used to forecast the characteristics of fresh samples. Random forests are an expansion of decision trees in which numerous trained trees are blended to create predictions.
- **Neural Networks:** Neural networks may also be used to solve classification issues involving many classes. They are well suited for complicated, non-linear issues and are capable of handling massive volumes of data. Multi-layer perceptron (MLP) is a prominent design for multiclass classification in neural networks. We used neural networks along with softmax function to perform multiclass classification tasks in the proposed MLP architecture (24).
- **Support Vector Machines (SVMs):** SVMs are another prominent approach for tackling classification issues involving many classes. The objective of SVMs is to identify a hyperplane that splits the data into distinct classes. In multiclass classification, the one-vs-one method is often used, in which different SVMs are trained for each pairwise comparison and the class with the greatest predicted probability is chosen as the final prediction.

The approach used for a specific issue relies on the nature of the problem, the amount and quality of the data, and the available computer resources. Certain strategies may perform better with particular issues or data types, whilst others may be more computationally efficient. It is essential to examine the performance of many strategies and choose the one that yields the best results for a particular issue.

4.4 Section References

73. Parsons, S. (2010). Introduction to Machine Learning, Second Editon by Ethem Alpaydin, MIT Press
74. Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

75. Gahukar, G., & Gahukar, G. (2019). Classification Algorithms in Machine Learning.

76. Zhang, G. P. (2000). Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 30(4), 451-462.

77. Aly, M. (2005). Survey on multiclass classification methods. Neural Netw, 19(1-9), 2.

5 Machine Learning in practice

The fast expanding branch of computer science known as machine learning enables computer systems to improve their performance on a given job via experience. Many frameworks have been created to facilitate the construction and training of machine learning models. These frameworks include pre-built functions and libraries that may be used for a variety of activities, including data preparation, model training, and deployment. There are several prominent frameworks for machine learning, each with its own strengths and shortcomings. Among the most popular frameworks are TensorFlow, PyTorch, and scikit-learn. We implemented our systems using TensorFlow and scikit-learn. In addition, we used the keras API to develop deep learning models.

5.1 TensorFlow

TensorFlow is an open-source machine learning and AI software library. It was created by the Google Brain team and is currently maintained by TensorFlow. TensorFlow is a comprehensive machine learning model development and deployment framework. The goal of TensorFlow is to simplify the construction and deployment of machine learning models by developers. To achieve this goal, TensorFlow has a number of key features that set it apart from other machine learning frameworks [78,79].

1. TensorFlow uses a **computational graph** to represent and execute machine learning models. This allows developers to build models by defining a series of operations and transformations that data will undergo. The computational graph can then be executed on a variety of hardware, including CPUs and GPUs, making it highly scalable and flexible. More specifically, TensorFlow uses a directed acyclic graph (DAG) to represent a machine learning model. The nodes in the network represent mathematical operations, whereas the edges reflect the data flow between operations. This allows TensorFlow to easily represent complex models, and makes it easier to visualize and debug them [79].

2. TensorFlow offers a variety of tools for **visualizing** and **debugging** machine learning models, including TensorBoard. This allows developers to monitor the training process, identify performance bottlenecks, and make improvements to their models [79].

3. TensorFlow has a big and active developer and user **community** that contributes to the library's development and provides mutual assistance. This makes it simpler for developers to discover answers to issues, discuss best practices, and keep abreast of the most recent advancements in their area [80].

4. **Data Flow:** TensorFlow uses a data flow paradigm, where data is passed through the graph of operations, from input to output. This allows TensorFlow to efficiently handle large amounts of data, and parallelize computations across multiple devices, such as GPUs and TPUs [79].

5. **Tensor:** In TensorFlow, all data is represented as tensors, multi-dimensional arrays that can be processed by the computational graph. This makes it easy to perform mathematical operations on data, and to automatically differentiate tensors in order to perform gradient-based optimization [79].

6. **Auto Differentiation:** TensorFlow offers an engine for automated differentiation capable of automatically calculating the gradients of a computational network with respect to its inputs. This enables developers to construct machine learning models utilizing gradient-based optimization methods such as stochastic gradient descent (SGD) without manually calculating gradients. [79].

7. **Model Deployment:** TensorFlow provides a range of tools for deploying machine learning models in a variety of environments, including web services, mobile devices, and edge devices. TensorFlow also supports model export to other platforms, such as TensorFlow Lite for mobile devices, and TensorFlow.js for web browsers [81,82].

8. **High-level APIs:** TensorFlow offers high-level APIs, such as Keras and Estimators, for constructing and training machine learning models. These APIs simplify the process of getting started with TensorFlow and offer a simple and straightforward interface for constructing and training models. In this thesis we used the keras API [83].

9. **Eager Execution:** TensorFlow provides an eager execution mode, which allows developers to execute operations immediately as they are defined, rather than building a computational graph. This makes it easier to debug models, and provides a more intuitive interface for development [84].

5.2 Keras

Python-based Keras is a high-level deep learning API that provides a convenient and user-friendly interface for building and training deep learning models. It was developed to make it easier for developers to begin with deep learning, and to provide a simplified interface for building and training complex models [85].

Keras was initially developed as a standalone deep learning library, but it was later integrated into TensorFlow as an official high-level API. This integration allowed us to leverage the power and scalability of TensorFlow while still enjoying the simplicity and ease of use of Keras [85].

When using Keras with TensorFlow, the underlying TensorFlow engine is used to perform the computational heavy lifting, while the Keras API provides a higher-level interface for building and training models. This makes it easier for developers to quickly build and experiment with different model architectures, avoiding TensorFlow's low-level complexities. [85].

Leveraging the simplicity of the keras API we afford to create a new class that implements the idea of the Dense Hermite Polynomial Layer with Shared Weights.

Some of the key technical aspects of Keras that are:

1. **Model Definition:** Keras offers a straightforward and user-friendly interface for creating deep learning models. Models may be created using a series of levels, with each layer representing a data transformation. Layers may be piled over one another to create more intricate models. [86].

2. **Layer Abstraction:** Keras has many predefined layer types, including dense, convolutional, recurrent, and pooling layers. These types hide away the underlying mathematical processes, making it simpler to construct sophisticated models without writing low-level code directly. [87].

3. **Backend Engine:** Keras provides a backend engine, which can be either TensorFlow, Theano, or CNTK. The backend engine is responsible for executing the operations defined in the model, and for handling the underlying computations. By using a backend engine, Keras allows developers to leverage the performance and scalability of existing machine learning libraries, such as TensorFlow, while still using a simple and intuitive interface [85].

4. **Model Compilation:** Once a model is defined in Keras, it must be compiled before it can be trained. During compilation, the backend engine is used to compile the model into a set of executable operations, and to configure the optimizer and loss function that will be used for training [88].

5. **Training and Evaluation:** Keras offers a versatile and user-friendly interface for training and assessing deep learning models. Several optimization techniques, such as stochastic gradient descent (SGD), Adam, and RMSprop, may be used to train models. The backend engine is utilized to do calculations and change model parameters during training. Keras also offers a variety of evaluation measures, including accuracy, precision, recall, and F1-score, which may be used to evaluate the performance of a trained model [89].

6. **Model Saving and Loading:** Keras allows developers to save trained models to disk, and to load them back into memory for further use. This makes it easy to save the state of a trained model, and to continue training it later [90].

7. **Pre-trained Models:** Keras includes a variety of pre-trained models that may be used as building blocks for new models. These models have been trained on big datasets and may be used to refine a fresh dataset or as a feature extractor for transfer learning. [91].

5.3 Weka

Weka is a well-known, open-source machine learning program written in Java and created at the University of Waikato in New Zealand. It offers an extensive collection of tools for data preparation, classification, regression, clustering, association rule mining, and visualization. [92].

One of the main advantages of Weka is its user-friendly interface and graphical user interface (GUI). This makes it easy for users with little or no programming experience to use and explore the software. Additionally, Weka provides an extensive library of algorithms, including popular algorithms for classification, regression, clustering, and association rule mining. This makes it easy to find and use the right algorithm for a particular problem [93].

Weka also provides a number of tools for assessing the effectiveness of machine learning models, including cross-validation, confusion matrices, receiver operating characteristic (ROC) curves, and lift charts. This facilitates the comparison of algorithm performance and the selection of the optimal solution for a given issue [94].

Another advantage of Weka is its integration with other tools and platforms. Weka can be used as a standalone software, or it can be integrated with other software, such as RapidMiner and KNIME, to create more advanced workflows. In addition, Weka may be integrated into other programming languages, such as Java, R, and Python, making it simple to include machine learning techniques into bigger projects. [95].

Weka also provides a number of preprocessing tools, including data cleaning and normalization, missing value imputation, feature selection, and dimensionality reduction. This facilitates data preparation and enhances the performance of machine learning algorithms [96].

In this thesis, we utilized the Weka software to compare the classification results of the already-implemented algorithms DFNN, Decision Stump, Random Tree, and J48 with the classification results achieved by our proposed feedforward neural network structure.

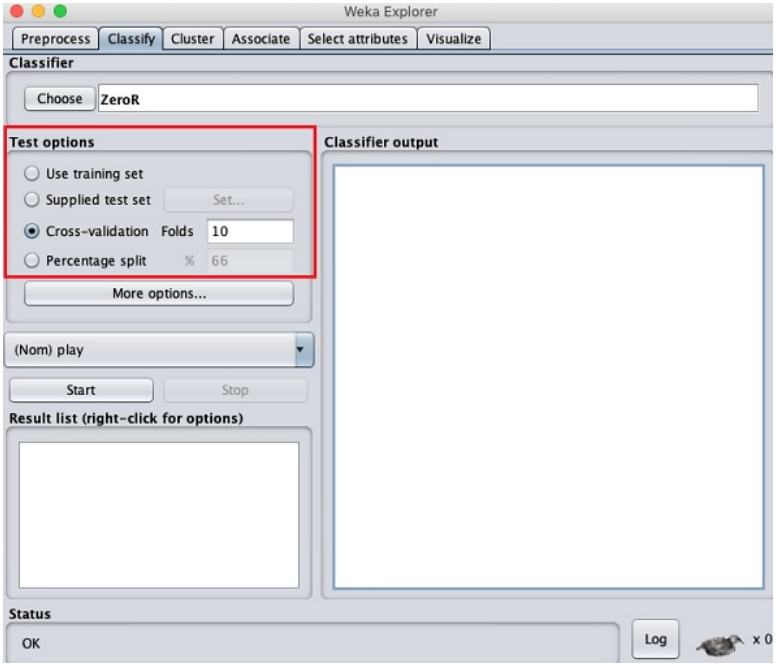


Figure 22: Example GUI interface inside the Weka software

5.4 Section References

78. Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep learning with tensorflow: A review. Journal of Educational and Behavioral Statistics, 45(2), 227-248.

79. McClure, N. (2017). TensorFlow machine learning cookbook. PACKT publishing Ltd.

80. Joana Carrasqueira and Lynette Gaddi, Program Managers at Google, Meet TensorFlow community leads around the world, https://blog.tensorflow.org/2021/06/meet-tensorflow-community-leads-around-the-world.html

81. Goldsborough, P. (2016). A tour of tensorflow. arXiv preprint arXiv:1610.01178.

82. Zanini, H. Custom Real-Time Object Detection in the Browser Using TensorFlow. js.

83. Bisong, E., & Bisong, E. (2019). Tensorflow 2.0 and keras. Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, 347-399.

84. Singh, P., Manure, A., Singh, P., & Manure, A. (2020). Introduction to tensorflow 2.0. Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python, 1-24.

85. Manaswi, N. K., Understanding and working with Keras. Deep learning with applications using Python: Chatbots and face, object, and speech recognition with TensorFlow and Keras, 31-43.

86. Ketkar, N., (2017). Introduction to keras. Deep learning with python: a hands-on introduction, 97-111.

87. Moolayil, An introduction to deep learning and keras. Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python, 1-16.

88. Brownlee, J. (2016). Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras. Machine Learning Mastery.

89. Gulli, A., & Pal, S. (2017). Deep learning with Keras. Packt Publishing Ltd

90. Athrey, K. S. (2018). Tutorial on Keras.

91. Chollet, F. (2018). Introduction to keras. March 9th.

92. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 10-18.

93. Kotak, P., & Modi, H. (2020, October). Enhancing the data mining tool WEKA. In 2020 5th International Conference on Computing, Communication and Security (ICCCS) (pp. 1-6). IEEE.

94. Brownlee, J. (2019). Machine learning mastery with Weka. Ebook. Edition, 1(4).

95. Dwivedi, S., Kasliwal, P., & Soni, S. (2016, March). Comprehensive study of data analytics tools (RapidMiner, Weka, R tool, Knime). In 2016 Symposium on Colossal Data Analysis and Networking (CDAN) (pp. 1-8). IEEE.

96. Devi, G. N. R., Ravi, M., & Kumar, A. N. (2021, July). Improve the classifiers efficiency by handling missing values in diabetes dataset using WEKA filters. In AIP Conference Proceedings (Vol. 2358, No. 1, p. 080016). AIP Publishing LLC.

6 Proposed Architecture: "Deep Feedforward Neural Network Classifier with Polynomial Layer and Shared Weights"

This section contains the main research work of this master thesis that has been accepted and presented in the 4^{th} International "Conference on *Advances in Signal Processing and Artificial Intelligence* (ASPAI 2022), Corfu, Greece, October 19-21, 2022."

6.1 Introduction

Deep feedforward neural networks (DFNNs) have been exercised as effective tools in dealing with a large variety of application frameworks [97-100]. Their simple structure provides the advantage of adopting a wide range of learning mechanisms. Truong et al [97] employed a multi-objective evolutionary computation algorithm to train a DFNN and use it to resolve problems related to the optimization of graded beams under static loads and free vibration. Cardoso et al [101] implemented dimensionality reduction mechanisms to study data related to diffuse lung diseases and designed a DFNN for improving the classification accuracy of such kind of data. Chen et al [102] expanded the connections of tree skeletons from hierarchical latent tree models in order to generate an unsupervised training mechanism for the effective development and application of typical DFNNs. In [103], a Gabor convolutional neural network able to perform feature extraction was concatenated with a DFNN that encompassed several fully connected layers to further improve the classification accuracy. In [104], a learning strategy for DFNNs was proposed, which was based on combining the tabu evolutionary computation scheme with a variant of the gradient descent optimization approach.

To enhance the ability of a neural network in effectively dealing with highly nonlinear data, many scholars have used polynomials activation functions, yielding the so called polynomial neural networks (PNNs) [105-107]. Inserting polynomial activation functions into the network's design process constitutes an effective way to approximate the input output relationships because those functions possess high-order representation capabilities [108-111]. So far, many different types of polynomials have been embedded in the design process of neural networks uch as, Hermite polynomials [105, 106], Legendre polynomials [107, 108], and Chebyshev polynomials [109, 110]. PNNs have been considered as trustworthy tools for regression and classification problems, spanning over a wide range of applications related to coastal engineering [106], differential equations [107, 109], energy saving [108], pattern classification [110], medical applications [111, 112], etc.

In [105], a feedforward PNN with one hidden layer was designed, which admitted linear combinations of the input variables and inferred the output in terms of Hermite polynomials activation functions. In [108], the aforementioned linear combinations were normalized within the interval $[-1, 1]$ and fed into Legendre polynomial activation functions. An implementation problem related to the above methods is that each linear combination enters one distinct polynomial thus yielding non-exact series expansions of each linear combination thus, compromising the network's efficiency. In [111, 112] a stacked network structures was used to design deep polynomial networks for medical image processing. The difficulty associated with that strategy is that as the number of layers increases the polynomial orders of the input variables also increase. This might result in high computational complexity, while it needs careful design to avoid overfitting problems. Misra et al [113] developed PNNs in terms of the well-known Kolmogorov-Gabor polynomials. The same polynomials were employed in [114, 115] to develop neuro-fuzzy networks. However, the use of such kind of polynomials acts to significantly increase the number of the synaptic weights thus, compromising the computational complexity of the resulting network.

In this paper, a polynomial DFNN is developed, which encompasses several fully connected layers concatenated with a polynomial layer that includes activations functions represented by Hermite polynomials. Hermite polynomials are orthogonal, and they been acknowledged as effective framework to cope with highly nonlinear data [116]. Moreover, they possess certain advantages over other orthogonal polynomials such as the Legendre or Chebyshev polynomials. For example, the Hermite polynomials are orthogonal over the whole set of real numbers, whereas the Chebyshev and Legendre are orthogonal in the interval $[-1, 1]$ [116]. Thus, the usage of the latter in designing neural networks requires a normalization process that results in constrained optimization mechanisms [108]. To alleviate the problems discussed in the previous paragraph, the structure of the proposed network follows two basic design principles. First, contrary to the methods in [105, 108], the polynomial layer generates exact truncated Hermite polynomial series for each one of its inputs. Second, each input entering the layer is connected with several polynomial nodes using shared synaptic weights. Thus, contrary to the approaches in [113-115], the resulting number of design parameters lies within acceptable levels. Finally, comparative experimental analysis based of inference statistics verifies the improved performance of the proposed network.

The following subsections are synthesized as follow. Section 6.2 provides the detailed theoretical analysis of the proposed network. The experimental study is described in Section 6.3. Finally, the paper concludes in Section 6.4.

6.2 The proposed network

The proposed deep feedforward neural network Figure 23 consists of several fully connected (i.e., dense) layers followed by a polynomial layer that generates a number of truncated Hermite polynomial series, which are fed into the output

layer. In what follows, we first describe some properties of the Hermite polynomials providing the nomenclature used in this paper, also. Then, the detailed presentation of the proposed network takes place along with the relative analysis.

The Hermite polynomials are orthogonal satisfying the subsequent differential equation [116]:

$$H_n(x) = (-1)^n e^{x^2} \frac{d\left(e^{-x^2}\right)}{dx} \tag{1}$$

where n defines the polynomial order and $x \in R$. The orthogonality of Hermite polynomials is sustained over the set $(-\infty, +\infty)$ having as weight the quantity e^{-x^2}

$$\langle H_n(x), H_m(x) \rangle = \begin{cases} 2^n n! \sqrt{\pi}, \text{ if } \text{ n } = m \\ 0, \text{ otherwise} \end{cases} \tag{2}$$

where $\langle , \rangle$ stands for the inner product operation. The zero and first order polynomials are equal to $H_0(x) = 1$ and $H_1(x) = 2x$, respectively. As such, any Hermite polynomial of order $n \geq 2$ is calculated in terms of the next recurrence relation,

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \tag{3}$$

It can be easily shown that $P_n(x)$ can be described as [10]:

$$H_n(x) = \sum_{k=0}^{n} \phi_{kn} x^k \tag{4}$$

with

$$\phi_{kn} = \begin{cases} 2\phi_{n-1,k-1} - (k+1)\phi_{n-1,k+1}, & \text{if } k > 0 \\ -\phi_{n-1,k+1}, & \text{if } k = 0 \end{cases} \tag{5}$$

where $\phi_{00} = 1$, $\phi_{10} = 0$ and $\phi_{11} = 2$. The overall structure of the proposed network is illustrated in Fig. 23. Specifically, it consists of the input layer, L dense hidden layers and one polynomial layer that feeds the output layer. The detailed functionality of the polynomial layers and its connection to the L^{th} layer and the output layer is depicted in Fig. 24.The input layer admits p variables, which come in the form of the vector $\mathbf{x}=[x_1, x_2, \ldots, x_p]^T \in R^p$. The first layer includes c_1 nodes.

The corresponding bias vector is defined as:

$$\boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} & b_2^{(1)} & \ldots & b_{c_1}^{(1)} \end{bmatrix}^T \tag{6}$$

and the synaptic weight vectors as:

$$\boldsymbol{w}_{i_1}^{(1)} = \left[w_{i_1,1}^{(1)}, w_{i_1,2}^{(1)}, \ldots, w_{i_1,p}^{(1)} \right]^T \tag{7}$$

with $i_1 = 1, 2, \ldots, c_1$.The layer's output $\boldsymbol{g}^{(1)} : R^p \rightarrow R^{c_1}$ comes in the form:

$$\boldsymbol{g}^{(1)}(\boldsymbol{x}) = \left[g_1^{(1)}(\boldsymbol{x}), g_2^{(1)}(\boldsymbol{x}), \ldots, g_{c_1}^{(1)}(\boldsymbol{x}) \right]^T \tag{8}$$

with $g_{i_1}^{(1)} : R^p \rightarrow R$

$$g_{i_1}^{(1)}(\boldsymbol{x}) = g_{i_1}^{(1)} \left(\boldsymbol{w}_{i_1}^{(1)T} \boldsymbol{x} + b_{i_1}^{(1)} \right) \tag{9}$$

The layer $\ell (\ell = 1, 2, \ldots, L)$ encompasses c_ℓ hidden nodes. The bias and the synaptic weights vectors associated with that layer are defined as

$$\boldsymbol{b}^{(\ell)} = \left[b_1^{(\ell)}, b_2^{(\ell)}, \ldots, b_{c_\ell}^{(\ell)} \right]^T \tag{10}$$

and

$$\boldsymbol{w}_{i_\ell}^{(\ell)} = \left[w_{i_\ell,1}^{(\ell)}, w_{i_\ell,2}^{(\ell)}, \ldots, w_{i_\ell,c_{\ell-1}}^{(\ell)} \right]^T \tag{11}$$

with $i_\ell = 1, 2, \ldots, c_\ell$, thus resulting in the following weight matrix

$$\boldsymbol{W}^{(\ell)} = \left[\boldsymbol{w}_1^{(\ell)T}, \boldsymbol{w}_2^{(\ell)T}, \ldots, \boldsymbol{w}_{c_\ell}^{(\ell)T} \right]^T \tag{12}$$

The output of the layer $\boldsymbol{g}^{(\ell)} : R^{c_{\ell-1}} \rightarrow R^{c_\ell}$ is described by the next vector-based form:

$$\boldsymbol{g}^{(\ell)}(\boldsymbol{x}) = \left[g_1^{(\ell)}(\boldsymbol{x}), g_2^{(\ell)}(\boldsymbol{x}) \ldots, g_{c_\ell}^{(\ell)}(\boldsymbol{x})\right]^T \tag{13}$$

In (13) each element is defined as:

$$g_{i_\ell}^{(\ell)}(\boldsymbol{x}) = g_{i_\ell}^{(\ell)}\left(\boldsymbol{w}_{i_\ell}^{(\ell)T}\boldsymbol{g}^{(\ell-1)}(\boldsymbol{x}) + b_{i_\ell}^{(\ell)}\right) \tag{14}$$

with $g_{i_\ell}^{(\ell)} : R^{c_{\ell-1}} \rightarrow R$.

The interaction between the layer L and the output layer is defined in terms of the polynomial layer P, the basic structure of which is depicted in Fig. 24. The main purpose of using the polynomial layer is to generate truncated Hermite polynomial series of linear combinations of the L^{th} layer's output vector $\boldsymbol{g}^{(L)}(\boldsymbol{x})$. To accomplish that task, the polynomial layer encompasses d groups of nodes, each of which includes n+1 nodes, where the activation function of the first node is the Hermite polynomial of 0 order, the activation function of the second node is the Hermite polynomial of order 1, and finally the activation function of the n+1-th node is the Hermite polynomial of order n of the aforementioned linear combinations.Thus, in total, the layer contains $d(n+1)$ polynomial nodes, defined as follows:

$$G_1^{(P)} = \left\{g_1^{(P)}(\boldsymbol{x}), \ldots, g_{n+1}^{(P)}(\boldsymbol{x})\right\} \tag{15}$$

$$G_{i_P}^{(P)} = \left\{g_{(i_P-1)(n+1)+1}^{(P)}(\boldsymbol{x}), \ldots, g_{i_P(n+1)}^{(P)}(\boldsymbol{x})\right\} \tag{16}$$

$$G_d^{(P)} = \left\{g_{(d-1)(n+1)+1}^{(P)}(x), \ldots, g_{d(n+1)}^{(P)}(x)\right\} \tag{17}$$

The bias vector and the synaptic weight matrix associated with the polynomial layer are written as:

$$\boldsymbol{b}^{(P)} = \left[\begin{array}{cccc} b_1^{(P)} & b_2^{(P)} & \ldots & b_d^{(P)} \end{array}\right]^T \tag{18}$$

$$\boldsymbol{W}^{(P)} = \left[\boldsymbol{w}_1^{(P)T}, \boldsymbol{w}_2^{(P)T}, \ldots, \boldsymbol{w}_d^{(P)T}\right]^T \tag{19}$$

where:

$$\boldsymbol{w}_{i_P}^{(P)} = \left[w_{i_P,1}^{(P)}, w_{i_P,2}^{(P)}, \ldots, w_{i_P,c_L}^{(P)}\right]^T (i_P = 1, 2, \ldots, d) \tag{20}$$

As such, the above-mentioned linear combinations of the L^{th} layer's outputs are described as indicated next

$$z_{i_P}^{(P)}(\boldsymbol{x}) = \boldsymbol{w}_{i_P}^{(P)T}\boldsymbol{g}^{L}(\boldsymbol{x}) + b_{i_P}^{(P)} \tag{21}$$

with $i_P = 1, 2, \ldots, d$ and the activation functions for the i_P-th group of nodes, given in equations (15)- (17), are as follows:

$$g_t^{(P)}(\boldsymbol{x}) = \begin{cases} H_0\left(z_{i_P}^{(P)}(\boldsymbol{x})\right), \text{ if } \mathrm{t} = (i_P-1)(n+1)+1 \\ H_1\left(z_{i_P}^{(P)}(\boldsymbol{x})\right), \text{ if } \mathrm{t} = (i_P-1)(n+1)+2 \\ \ldots \\ H_n\left(z_{i_P}^{(P)}(\boldsymbol{x})\right), \text{ if } \mathrm{t} = i_P(n+1) \end{cases} \tag{22}$$

where $H_v\left(z_{i_P}^{(p)}(\boldsymbol{x})\right)$, for $v = 0, 1, \ldots, n$ are calculated in equations (4) and (5). A basic property of the polynomial layer is that the synaptic weights are shared between the polynomial nodes that belong to the same group. Indeed, the linear combination in eq. (21) can be rewritten as:

$$z_{i_P}^{(P)}(\boldsymbol{x}) = \sum_{i_L=1}^{c_L} w_{i_p,i_L}^{(P)} g_{i_L}^{(L)}(\boldsymbol{x}) + b_{i_P}^{(P)} \tag{23}$$

Thus, the synaptic weight associated with the activation function $g_{i_L}^{(L)}(\boldsymbol{x})\,(i_L = 1, 2, \ldots, c_L)$ and the i_P-th group of polynomials, given in (22), is the $w_{i_p,i_L}^{(P)}$. Therefore, all polynomial nodes, belonging to the group i_p, share the weight $w_{i_p,i_L}^{(P)}$ as far as the $g_{i_L}^{(L)}(\boldsymbol{x})$ is concerned. The same holds for the corresponding bias $b_{i_P(P)}^{(P)}$. The above analysis is clearly illustrated in Fig. 2.

To this end, the output of the polynomial layer comes in the following vector form:

$$\boldsymbol{g}^{(P)}(\boldsymbol{x}) = \left[g_1^{(P)}(\boldsymbol{x}), g_2^{(P)}(\boldsymbol{x}) \ldots, g_{d(n+1)}^{(P)}(\boldsymbol{x})\right]^T \tag{24}$$

The output layer calculates m outputs $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m$. The corresponding synaptic weight matrix and the bias vector are:

$$\boldsymbol{W}^{(s)} = \left[\boldsymbol{w}_1^{(s)T}, \boldsymbol{w}_2^{(s)T}, \ldots, \boldsymbol{w}_m^{(s)T}\right]^T \tag{25}$$

$$\boldsymbol{b}^{(s)} = \left[\begin{array}{cccc} b_1^{(s)} & b_2^{(s)} & \ldots & b_m^{(s)} \end{array}\right]^T \tag{26}$$

where $\mu = 1, 2, \ldots, m$ and each weight vector in (29) is:

$$\boldsymbol{w}_\mu^{(s)} = \left[w_{\mu,1}^{(P)}, w_{\mu,2}^{(P)}, \ldots, w_{\mu,d(n+1)}^{(P)}\right]^T \tag{27}$$

The output $\hat{y}_\mu$ with $\mu = 1, 2, \ldots, m$ is calculated using the softmax activation function:

$$\hat{y}_\mu = \frac{e^{r_\mu^{(s)}}}{\sum_{k=1}^{m} e^{r_k^{(s)}}} \tag{28}$$

where

$$r_\mu^{(s)} = \boldsymbol{w}_\mu^{(s)T} \boldsymbol{g}^{(P)}(\boldsymbol{x}) + b_\mu^{(s)} \tag{29}$$

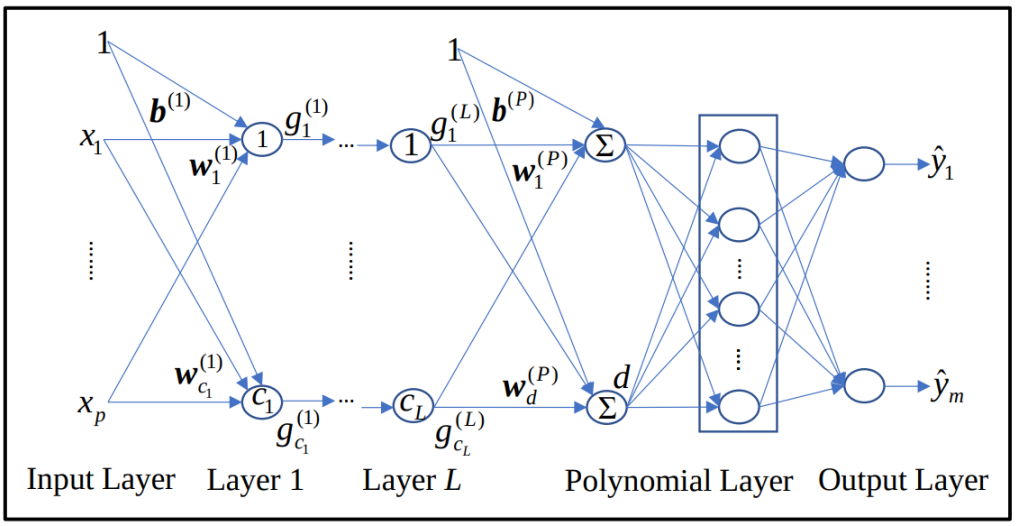


Figure 23: The structure of the proposed network.

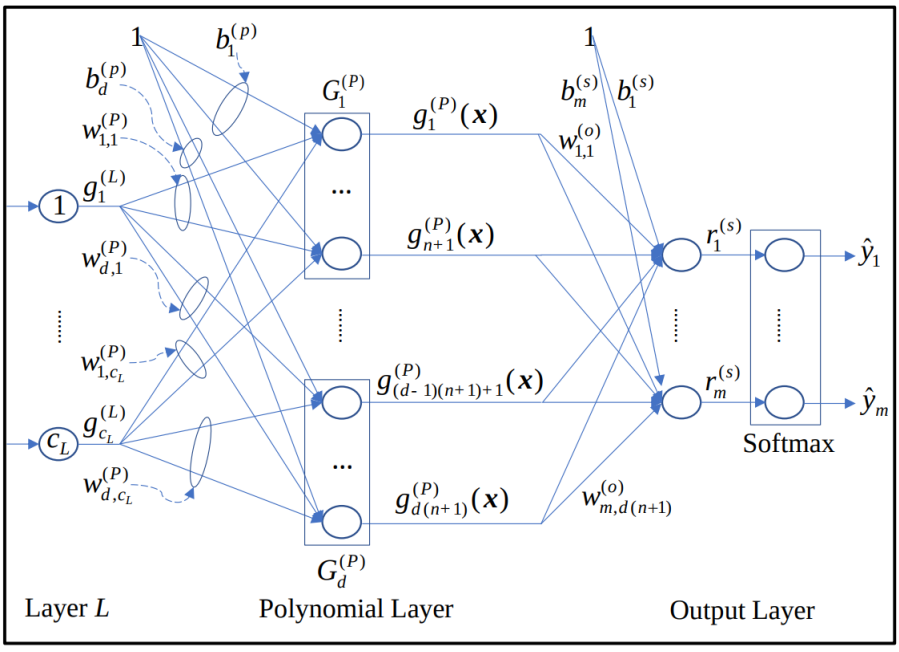


Figure 24: The structure of the polynomial and output layers (the elliptical curves contain the shared weights)

Next, we show that the quantity $r_{\mu}^{(s)}$ can be written as the sum of d truncated Hermite polynomial series of order n.

From (29) we obtain

$$r_{\mu}^{(s)} = \sum_{t=1}^{d(n+1)} w_{\mu,t}^{(s)} g_t^{(P)}(\boldsymbol{x}) + b_{\mu}^{(s)} \tag{30}$$

which is modified as:

$$r_{\mu}^{(s)} = \sum_{t=1}^{n+1} w_{\mu,t}^{(s)} g_t^{(P)}(\boldsymbol{x}) + \ldots + \sum_{t=(d-1)(n+1)+1}^{d(n+1)} w_{\mu,t}^{(s)} g_t^{(P)}(\boldsymbol{x}) + b_{\mu}^{(s)} \tag{31}$$

Thus,

$$r_{\mu}^{(s)} = \sum_{i_P=1}^{d} \sum_{t=(i_P-1)(n+1)+1}^{i_P(n+1)} w_{\mu,t}^{(s)} g_t^{(P)}(\boldsymbol{x}) + b_{\mu}^{(s)} \tag{32}$$

For a specific value of the index i in eq. (22), there is a one-to-one correspondence between the index

$$t = (i_P - 1)(n+1) + 1, (i_P - 1)(n+1) + 2 \ldots, i_P(n+1) \tag{33}$$

and the index $v = 0, 1, 2, \ldots n$. In addition, for a specific i_P, the subsequent substitution takes place: $a_{\mu,v}^{(i)} = w_{\mu,t}^{(s)}$. Taking into account the activation functions $g_t^{(P)}(\boldsymbol{x})$ in eq.(22) the eq.(32) is modified as follows:

$$r_{\mu}^{(s)} = \sum_{i_P=1}^{d} \sum_{v=0}^{n} a_{\mu,v}^{(i)} H_v\left(z_{i_P}^{(P)}(\boldsymbol{x})\right) + b_{\mu}^{(s)} \tag{34}$$

Since the polynomial $H_0\left(z_{i_P}^{(P)}(\boldsymbol{x})\right)$ is equal to 1, the $b_{\mu}^{(s)}$ can be absorbed by the weight of one of these polynomials. Finally, we can easily notice that the $r_{\mu}^{(s)}$ is represented as the sum of d truncated Hermite polynomial series of order n.

6.3 Simulation experiments results

In this section, the proposed network is compared with four competitive methods. The first one is a deep feedforward neural network (DFNN), while the other three methods are taken from the WEKA software [117] and they are the well-known J48, the Decision Stump (DS), and the Random Tree (RT) algorithms. In all simulation experiments, the DFNN network utilizes 5 dense layers with 12 ReLu nodes each, while the output layer uses the "softmax" activation function.

The optimization task is conducted in terms of the ADAM optimizer [118], while the objective function is the categorical cross-entropy function. The performance index is the Classification Accuracy (CA) normalized in the interval [0,1]. Finally, five data sets are used to conduct the comparative analysis. The data sets were taken from the UCI Machine Learning Repository [119] and their basic properties are provided in Table 2.

Data Sets	**No of Instances**	**No of Inputs**	**No of Classes**
1: Abalone	4177	8	3
2: Cardiotocography	2126	21	3
3: Statlog German Credit	1000	24	2
4: Red Wine Quality	1599	11	6
5: Wisconsin Diagnostic Breast Cancer	569	30	2

Table 2: Datasets Properties

For each data set we used 5-fold cross-validation analysis. Regarding each fold the proposed network run 10 times with different initializations. Thus, for each data set and each method, 50 samples were generated.

Tables 3 and 4 summarize the mean values and the corresponding standard deviations of the Classification Accuracy (CA) index, obtained by the five competitive algorithms, regarding all tested data sets. The following remarks can be easily derived. First, in all tested data sets the proposed network clearly outperforms the other methodologies. Second, the case where truncated Hermite polynomial series of order n=2 are used, appear to achieve the best performance in two out of five data sets, while the use of truncated series of order n=3 provides the best results in the remaining three

data sets. Third, Table 3 directly indicates that the DFNN and the J48 are the dominant among the four competitive methods.

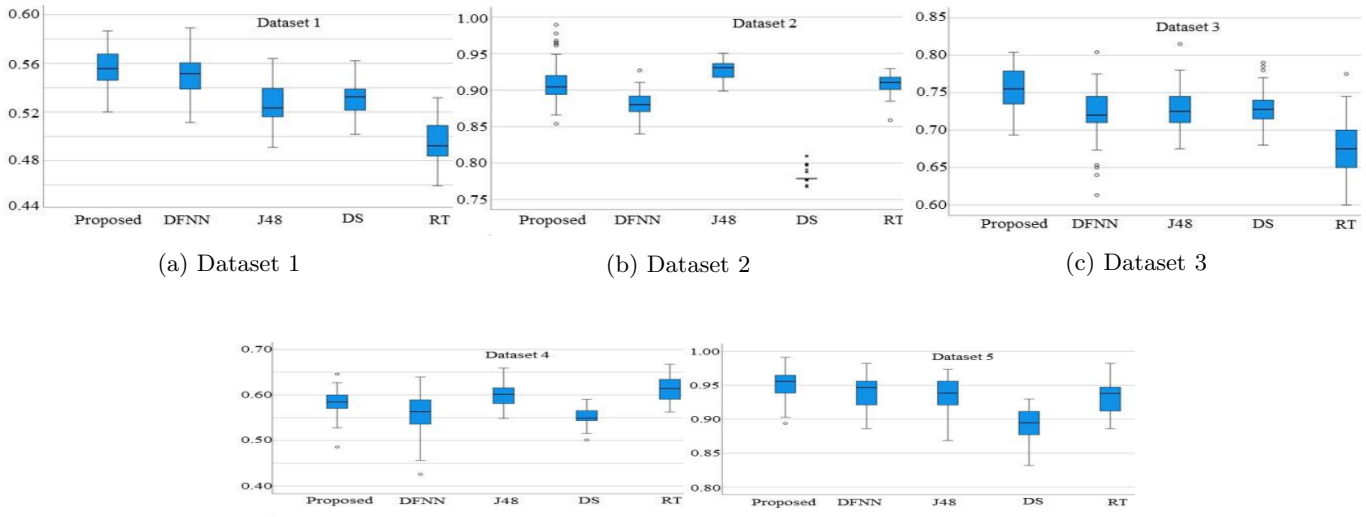
# Dataset	Proposed (n=2)	Proposed (n=3)	Proposed (n=4)
1	0.5629±0.02	0.555±0.01	0.551±0.02
2	0.899±0.02	0.910±0.03	0.879±0.03
3	0.720±0.03	0.755±0.02	0.733±0.03
4	0.586±0.03	0.583±0.03	0.577±0.04
5	0.942±0.02	0.952±0.02	0.880±0.10

Table 3: Classification accuracy index and the respective standard deviations obtained by the proposed network for the five datasets.

# Dataset	DFNN	J48	DS	RT
1	0.550±0.01	0.527±0.01	0.529±0.01	0.495±0.01
2	0.881±0.01	0.894±0.16	0.783±0.01	0.859±0.19
3	0.719±0.03	0.729±0.02	0.729±0.02	0.677±0.03
4	0.559±0.04	0.546±0.16	0.490±0.16	0.556±0.16
5	0.942±0.02	0.917±0.11	0.892±0.02	0.930±0.02

Table 4: Classification accuracy index and the respective standard deviations obtained by the four competitive methods for the five datasets.

To further scrutinize the above results, the five methods are compared in terms of inference statistics. As far as the proposed network is concerned the case of truncated Hermite polynomial series of order n=3 is considered. Descriptive statistics were evaluated regarding the samples obtained for each data set and each method. The resulting box-plots are illustrated in Figure 25. Next, we applied the Kolmogorov– Smirnov test, which showed that the data were normally distributed. Thus, to assess the hypothesis that there is a statistically significant difference between the proposed method and the other methods we applied paired t-tests for each dataset individually.



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

(d) Dataset 4 (e) Dataset 5

Figure 25: 4 Box plots of the five methods (proposed n=3) with respect to Dataset 1, Dataset 2, Dataset 3, Dataset 4 and Dataset 5

The null hypothesis assumes that the true mean difference is equal to zero, while the upper-tailed alternative hypothesis assumes that the difference is greater than zero. Table 4 depicts the 95intervals of paired differences and the corresponding p-values. Regarding the Dataset 1, the null hypothesis is rejected in favour of the alternative hypothesis for each pair. The proposed method outperforms J48, Decision Stump and Random Tree at significance less than 0.001 and DFNN at significance 0.04.

In the Dataset 2, our method has a statistically significant difference against DFNN and Decision Stump, where the null hypothesis is rejected at significance less than 0.001. Meanwhile, J48 outperforms the proposed method at significance less than 0.001 and there was no statistically significant difference between the proposed network and the Random Tree, thus we cannot reject the null hypothesis.

Pair	Mean	95% Confidence Interval of Difference	t	p-value
Proposed-DFNN	0.0052	[-0.001,0.011]	1.9	0.04
Proposed-J48	0.0278	[0.022,0.336]	9.6	< 0.001
Proposed-DS	0.0255	[0.199,0.310]	9.3	< 0.001
Proposed - RT	0.0623	[0.0533,0.67]	17.9	< 0.001

Table 5: **Dataset 1:** Comparative results obtained by the t-test inference statistics

Pair	Mean	95% Confidence Interval of Difference	t	p-value
Proposed-DFNN	0.2960	[0.0199, 0.0393]	6.1	< 0.001
Proposed-J48	-0.017	[-0.0260, -0.0080]	-3.8	< 0.001
Proposed-DS	0.1284	[0.1198, 0.1369]	30	< 0.001
Proposed - RT	0.0013	[-0.0083, 0.0109]	0.28	< 0.001

Table 6: **Dataset 2:** Comparative results obtained by the t-test inference statistics

Regarding the Dataset 3, the null hypothesis is rejected in every case at significance less than 0.001 directly indicating the superiority of our proposed network.

For the Dataset 4 the null hypothesis is rejected in every case. In addition, the proposed network performs better than DFNN and Decision Stump (significance less than 0.001), while it achieves inferior performance when compared to the J48 and Random Tree algorithms.

Finally, in the Dataset 5, the null hypothesis is rejected, and the proposed method seems to be superior than the other methods at significance level less than 0.001.

To summarize, the above results clearly indicate that the use of the polynomial layer significantly improved the performance of the DFNN, while maintaining a superior behavior when compared to the rest of the tested methods.

Pair	Mean	95% Confidence Interval of Difference	t	p-value
Proposed-DFNN	0.0363	[0.0243, 0.0484]	6.1	< 0.001
Proposed-J48	0.0264	[0.0171, 0.0356]	5.7	< 0.001
Proposed-DS	0.0255	[0.0157, 0.0352]	5.3	< 0.001
Proposed - RT	0.0779	[0.0661, 0.0896]	13.3	< 0.001

Table 7: **Dataset 3:** Comparative results obtained by the t-test inference statistics

Pair	Mean	95% Confidence Interval of Difference	t	p-value
Proposed-DFNN	0.0107	[0.0039, 0.0175]	3.2	0.001
Proposed-J48	0.0179	[0.0094, 0.0264]	4.2	< 0.001
Proposed-DS	0.0605	[0.0514, 0.0695]	13.4	< 0.001
Proposed - RT	0.0223	[0.0134, 0.0312]	5.03	< 0.001

Table 9: **Dataset 5:** Comparative results obtained by the t-test inference statistics

6.4 Conclusion

In this paper, a novel polynomial deep feedforward neural network classifier has been developed. The task was to investigate the potential influence of Hermite polynomial activation functions to the overall classification accuracy. The network consists of several fully connected layers followed by a polynomial layer with a specialized structure based on Hermite polynomials. The very core of the functionality of the polynomial is to admit linear combinations of the outputs coming from the previous dense layer and expand each one of them into truncated Hermite polynomial series. Then, each network's output is realized as the sum of the abovementioned series. The existence of Hermite polynomial truncated series in the network's inference enhances the corresponding classification performance because

Pair	Mean	95% Confidence Interval of Difference	t	p-value
Proposed-DFNN	0.0237	[0.0082, 0.0392]	3.1	0.002
Proposed-J48	-0.0166	[-0.0271, -0.0061]	-3.2	0.001
Proposed-DS	0.0323	[0.0223, 0.0423]	6.5	< 0.001
Proposed - RT	-0.0289	[-0.0402, -0.0177]	-5.2	< 0.001

Table 8: **Dataset 4:** Comparative results obtained by the t-test inference statistics

it embeds into the design process all the advantages provided by those series such as the strong modelling capabilities, the simplicity, the universal approximation property, etc. In addition, the connection between the polynomial layer and the previous dense layer is realized in terms of shared synaptic weights, which acts to produce a convenient training procedure. The network was compared with four other methods in terms of statistical analysis on the basis of several data sets. The simulation results support the above analysis since the network appeared to achieve superior performance.

6.5 Section References

97. T. T. Truong, J. Lee, and T. Nguyen-Thoi, "Multiobjective optimization of multi-directional functionally graded beams using an effective deep feedforward neural network-SMPSO algorithm", Structural and Multidisciplinary Optimization, vol. 63, pp. 2889-2918, 2021.

98. A. E. Korchi, and Y. Ghanou, "Backpropagation Issues with Deep Feedforward Neural Networks", in: Innovations in Smart Cities and Applications, vol. 37, M. B. Ahmed, and A. Boudhir, Eds. SCAMS Lecture Notes in Networks and Systems, Springer, 2017, pp. 335-343.

99. G. Masetti, and F. Di Giandomenico, "Analyzing Forward Robustness of Feedforward Deep Neural Networks with Leaky ReLU Activation Function Through Symbolic Propagation", Communications in Computer and Information Science, vol. 1323, pp. 460-474, 2020.

100. L. Zhang, T. Luo, F. Zhang, and Y. Wu, "A Recommendation Model Based on Deep Neural Network", IEEE Access, vol. 6, pp. 9454-9463, 2018.

101. I. Cardoso, E. Almeida, H. Allende-Cid, A. C. Frery, R. M. Rangayyan, P. M. Azevedo-Marques, and H. S. Ramos, "Evaluation of Deep Feedforward Neural Networks for Classification of Diffuse Lung Diseases", Lecture Notes in Computer Science, vol. 10657, pp. 152–159, 2018.

102. Z. Chen, X. Li, Z. Tian, and N. L. Zhang, "Fast Structure Learning for Deep Feedforward Networks via Tree Skeleton Expansion", Lecture Notes in Artificial Intelligence, vol.11726, pp. 277–289, 2019.

103. M. M. Lau, J. T. S. Phang and K. H. Lim, "Convolutional Deep Feedforward Network for Image Classification", 7th International Conference on Smart Computing & Communications (ICSCC), pp. 1-4, 2019.

104. T. K. Gupta and K. Raza, "Optimizing Deep Feedforward Neural Network Architecture: A Tabu Search Based Approach", Neuural Processing Letters, vol. 51, pp. 2855–2870, 2020

105. L. Ma, and K. Khorasani, "Constructive feedforward neural networks using Hermite polynomial activation functions", IEEE Transactions on Neural Networks, vol. 16(4), pp. 821-833, 2005.

106. G. E. Tsekouras, V. Trygonis, A. Maniatopoulos, A. Rigos, A. Chatzipavli, J. Tsimikas, N. Mitianoudis, and A. F. Velegrakis, "A Hermite neural network incorporating artificial bee colony optimization to model shoreline realignment at a reef-fronted beach", Neurocomputing, vol. 280, pp. 32-45, 2018.

107. S. Mall, and S. Chakraverty, "Application of Legendre Neural Network for solving ordinary differential equations", Applied Soft Computing, vol. 43, pp. 347– 356, 2016.

108. A. Rigos , G.E. Tsekouras , A. Chatzipavlis, and A.F. Velegrakis, "Modeling beach rota- tion using a novel Legendre polynomial feedforward neural network trained by nonlinear constrained optimization", in: Proceedings of the Twelfth IFIP International Conference on Artificial Intelligence Applications and Innovations, Thessaloniki, Greece, 2016, pp. 167–179.

109. Z. Hajimohammadi, F. Baharifard, A. Ghodsi, and K. Parand, "Fractional Chebyshev deep neural network (FCDNN) for solving differential models", Chaos, Solitons and Fractals, vol. 153, 111530, 2021.

110. Y. Zhang, Y. Yin, D. Guo, X. Yu, and L. Xiao, “Cross-validation based weights and structure determination of Chebyshev-polynomial neural networks for pattern classification”, Pattern Recognition, vol. 47, pp. 3414–3428, 2014.

111. J. Shi, S. Zhou, X. Liu, Q. Zhang, M. Lu, and T. Wang, “Stacked deep polynomial network based representation learning for tumor classification with small ultra sound image dataset”, Neurocomputing, vol. 194, pp. 87–94, 2016.

112. J. Shi, X. Zheng, Y. Li, Q. Zhang, and S. Ying, “Multimodal Neuroimaging Feature Learning With Multimodal Stacked Deep Polynomial Networks for Diagnosis of Alzheimer’s Disease”, IEEE Journal of Biomedical and Health Informatics, vol. 22 (1), pp. 173-183, 2018.

113. B. B. Misra, S. C. Satapathy, B. N. Biswal, P. K. Dash and G. Panda, ”Pattern Classification using Polynomial Neural Network”, IEEE Conference on Cybernetics and Intelligent Systems, pp. 1-6, 2006.

114. W. Huang, S. K. Oh, and W. Pedrycz, “Fuzzy reinforced polynomial neural networks constructed with the aid of PNN architecture and fuzzy hybrid predictor based on nonlinear function”, Neurocomputing, vol. 458, pp. 454–467, 2021.

115. C. Zhang, S. K. Oh, and Z. Fu, “Hierarchical polynomial-based fuzzy neural networks driven with the aid of hybrid network architecture and rankingbased neuron selection strategies”, Applied Soft Computing, vol. 113,107865, 2021.

116. G.E. Andrews, R. Askey, and R. Roy, “Special Functions”, Cambridge University Press, UK, 2000.

117. E. Frank, M. A. Hall, and I. H. Witten, The WEKA Workbench, Online Appendix for ”Data Mining: Practical Machine Learning Tools and Techniques”, Morgan Kaufmann, 4th Edition, 2016.

118. D. P. Kingma, and J. Ba, “Adam: A Method for Stochastic Optimization”, arXiv:1412.6980, https://doi.org/10.48550/arXi

119. UC Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml)