T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

MACHINE LEARNING FOR POLYNOMIALS

CANBERK ARICI

SUPERVISOR
ASST. PROF. ZAFEIRAKIS ZAFEIRAKOPOULOS

GEBZE
2022

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

MACHINE LEARNING FOR POLYNOMIALS

CANBERK ARICI

SUPERVISOR
ASST. PROF. ZAFEIRAKIS ZAFEIRAKOPOULOS

2022
GEBZE

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 20/01/2022 by the following jury.

JURY

Member
(Supervisor) : Asst. Prof. Zafeirakis Zafeirakopoulos

Member : Asst. Prof. Didem Gözüpek

ABSTRACT

This project aims to predict number of real roots of polynomials.

Dataset is created by randomly choosing real and complex roots, also complex conjugate of complex roots are included. Polynomials' degrees are up to 100. Dataset includes over 100000 polynomials and number of real roots of polynomials. Polynomials are generated as evenly distributed. The feature vector which is used for training is the vector of coefficients and number of real roots of each polynomials.

In test phase, 20% of the dataset is used for testing. 3 different architectures are aimed to be compared, which are Recurrent Neural Network(RNN), Long Short-Term Memory (LSTM), and Multilayer Perceptron(MLP).

Keywords: Polynomial, Real Root, Complex Root, RNN, LSTM, MLP.

ÖZET

Bu proje polinomların gerçek köklerinin sayısını tahmin etmeyi amaçlamaktadır.

Veri kümesi, gerçek ve karmaşık köklerin rastgele seçilmesiyle oluşturulur, ayrıca karmaşık köklerin karmaşık eşleniği dahildir. Polinomların dereceleri 100'e kadardır. Veri seti 100.000'den fazla polinomu ve polinomların gerçek köklerinin sayısını içerir. Polinomlar eşit dağılmış olarak üretilir. Eğitim için kullanılan öznitelik vektörü, her polinomun katsayılarını ve gerçek kök sayısını içerir.

Test aşamasında, veri setinin 20%'si test için kullanılır. Recurrent Neural Network(RNN), Long Short-Term Memory(LSTM),ve Multilayer Perceptron(MLP) olmak üzere 3 farklı mimarinin karşılaştırılması amaçlanmaktadır.

Anahtar Kelimeler: Polinom, Gerçek Kök, Karmaşık Kök, RNN, LSTM, MLP.

ACKNOWLEDGEMENT

I would like to express my sincere thanks to those who contributed to the preparation of this report, and to Asst. Prof. Zafeirakis Zafeirakopoulos, who guided me in getting the final version of this report and set an example for me both with his personality and his academic life and studies.

In addition, I would like to express my respect and love to my family, who supported me in every way during my education life, and to all people who contributed to my development.

Canberk ARICI

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
RNN	:	Recurrent Neural Network
LSTM	:	Long Short-Term Memory
MLP	:	Multilayer Perceptron
ANN	:	Artificial Neural Network

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. BACKGROUND

Artificial intelligence enables people to make life easier and to reach places that people cannot reach. Artificial intelligence is being applied to many different fields today and facilitates the solution of problems. Artificial intelligence has structures called neural networks. Neural networks, also known as artificial neural networks (ANNs) are a subset of machine learning and are at the heart of deep learning algorithms. Neural networks are a set of algorithms that simulate the functions of an animal brain in order to recognize patterns in large volumes of data. As a result, they resemble the neuronal and synaptic connections seen in the brain. Deep learning methods use neural networks with several process layers, which are referred to as "deep" networks.

Although machine learning-based approaches show examples of success, features such as the quality of the data and the complexity of the algorithm affect the level of success. As the complexity and operations of mathematical problems increase, it becomes more difficult for people to solve them. Different approaches have been used in solving mathematical problems from past to present.[1]

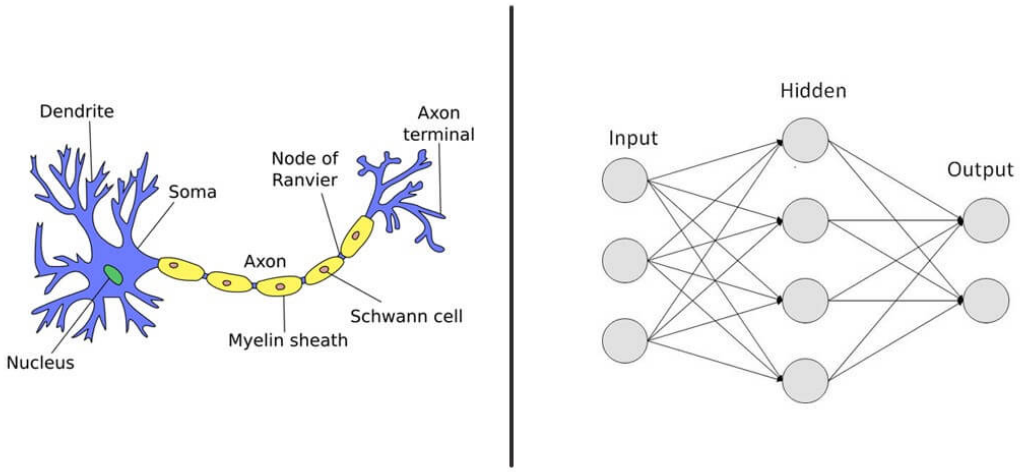

Figure 1.1: Simple Neural Network

[2]

The increase in the amount of data we have due to the development of technology and the fact that the computers that are powered by hardware can perform complex mathematical operations faster, have enabled the realization of neural networks and different architectures which use neural networks, with the emergence and development of graphics cards (GPU) based on parallel processor structures.

When we look at the studies and articles about finding the real roots of polynomials, it is observed that ANN is generally used[3] [4] [5] [6]. We can attribute the widespread use of ANN to the fact that the hardware of the computers at the time of the research was not sufficient or that other algorithms did not find enough space in the field of mathematics yet. Datasets which are used in experiments include polynomials whose degree is at most 25 and least computed MSE is 4.6057. In the following years, the success of the solution of the problem of finding the number of real roots of polynomials was increased by increasing the success in the methods.

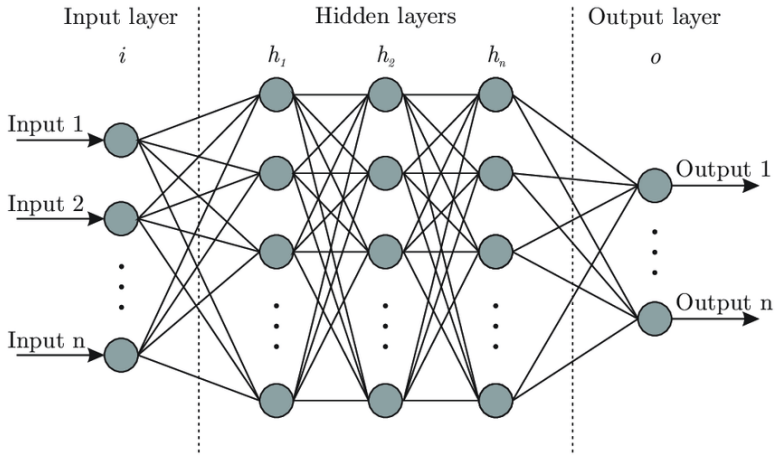

Figure 1.2: Artificial Neural Network Architecture

[7]

2. METHOD

In this section, methods and architectures relared to project will be introduced.

2.1. Dataset

According to fundamental theorem of algebra, a polynomial P(x) of degree n has n roots and according to Complex Conjugate Root Theorem, if P is a polynomial in one variable with real coefficients, and a+bi is a root of P with a and b real numbers, then its complex conjugate a-bi is also a root of P[8].
There are two types of roots: real roots and complex roots, where, according to the complex conjugate root theorem, a complex root always appears with its complex conjugate. As a result, complex roots are always found in pairs. As a result, the number of roots of a polynomial can be combined in a finite number of ways. For example, for a polynomial of degree eight, the combinations of its roots can only be

- no real root and eight complex roots
- two real roots and six complex roots
- four real roots and four complex roots
- six real roots and two complex roots
- eight real roots and no complex roots

Polynomials are generated by randomly choosing real and complex roots between 100 and -100. Number of real roots are counted and printed in a txt file, coefficients of polynomials are counted and printed in another txt file.

2.2. Architectures

2.2.1. Recurrent Neural Network(RNN)

Humans never create thoughts from the ground up. People, for example, will consider the meaning of a word in relation to the preceding phrase or context. That is why a recurrent neural network(RNN) was developed. RNN includes memory of prior input, unlike typical artificial neural network (ANN) models. For example, if your input is "He enjoys travelling," ANN would interpret the lines "He enjoys travelling" and "enjoys likes he" as having the same meaning since ANN is unconcerned with the sequence. There are no connections between any of the input nodes. However, just "He enjoys traveling" is correct in RNN, which is what we would expect. RNN can look for correlations between occurrences in various periods(long term dependencies). Since RNN has the ability to remember the previous inputs, in other words, the order of inputs nodes matters. Therefore RNN is one of the architectures in the project.

2.1.

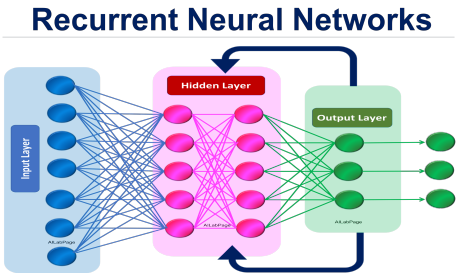

Figure 2.1: Recurrent Neural Network Architecture

[9]

2.2.

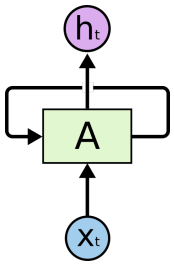

Figure 2.2: Loop of RNN

[10]

The structure of RNN is similar to the structure of ANN, except there is a loop in the hidden layer. A formula of computing the hidden layer of RNN can be

$$h_t = \phi(Wx_t + Uh_{t-1})$$

- h_t is the hidden state at time t;
- x_t is the input vector;
- W is the weight;
- h_t is the hidden state at previous time t;
- U is the memory rate;
- ϕ is an activation function, which can be a sigmoid function or tanh.

In the RNN Architecture, 5 hidden layers are used which are Dropout Layer, RNN Layer and Dense Layer. First layer is RNN Layer. In each epoch, hidden state of RNN is resetted to its original value because any accumulated information from last loss calculation is not wanted for the next loss calculation. Binary Cross Entropy is used for loss calculation. Sigmoid Activation Function is used in last Dense Layer because predictions for degrees upto 100 will be the output.

Here is the RNN Architecture in the project where input size is 100:

```
function RNN_model(input_size,num_of_classes)
    model = Chain(RNN(input_size,25),
                  Dropout(0.2),
                  RNN(25,25),
                  Dropout(0.1),
                  Dense(25,100),
                  Dropout(0.1),
                  Dense(100,num_of_classes,sigmoid)
                  )
    return model
end
```

Figure 2.3: RNN Architecture in the Project

2.2.2. Long Short-Term Memory(LSTM)

LSTM is a different kind of RNN Architecture. It controls memorizing or forgetting the context. In this project, the input vector is coefficients of polynomials and number of real roots of polynomials. The order of coefficients matters therefore LSTM is a good option for the project.

2.7.

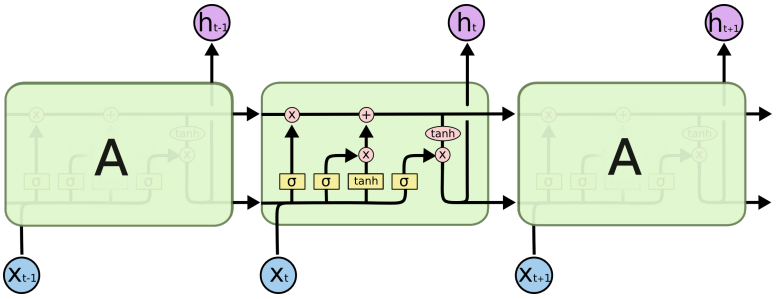

Figure 2.4: LSTM Model

[11]

LSTM has three parts. The first part decides what information to forget in the current state:

$f_t = \sigma(W_f.[h_{t-1},xt] + b_f)$

where f_t is a number between 0 and 1, indicating how much of the memory of the previous state C_{t-1} should be kept in the current state C_t.

The second part decides how much new data we will remember in the present cell state:

$i_t = \phi(W_i\ .\ [h_{t-1}, x_t] + b_i)$ and $\overline{C_t} = \tanh(W_t\ .\ [h_{t-1},x_t) + b_c)$,

where $\overline{C_t}$ is the information candidate and i_t determines how much of the candidate information we should remember.

The current cell state is initially updated in the third part:

$$C_t = f_t * C_{t-1} + i_t * \overline{C_t}.$$

Then the output of h_t is calculated:

$o_t = \sigma(\ W_o\ .\ [h_{t-1}, x_t] + b_o)$, and $h_t = o_t * \tanh(C_t)$.

The output cell states are determined by the sigmoid layer, and the tanh is an activation function that compresses the value into -1 to 1. Then C_t and h_t will enter the next cell state loop.

In the LSTM Architecture, 5 hidden layers are used which are Dropout Layer, LSTM Layer and Dense Layer as the same with RNN Architecture. First layer is LSTM Layer. In each epoch, hidden state of LSTM is resetted to its original value because any accumulated information from last loss calculation is not wanted for the next loss calculation as it was the case for RNN. Binary Cross Entropy is used for loss calculation. Sigmoid Activation Function is used in last Dense Layer because predictions for degrees upto 100 will be the output.

Here is the LSTM Architecture in the project where input size is 100:

```
function LSTM_model(input_size,num_of_classes)
    model = Chain(LSTM(input_size,25),
                  Dropout(0.2),
                  LSTM(25,25),
                  Dropout(0.1),
                  Dense(25,100),
                  Dropout(0.1),
                  Dense(100,num_of_classes,sigmoid)
                  )
    return model
end
```

Figure 2.5: LSTM Architecture in the Project

2.2.3. Multilayer Perceptron(MLP)

MLP is a neural network with a non-linear mapping between inputs and outputs. Input and output layers, as well as one or more hidden layers with many neurons stacked together, make up a Multilayer Perceptron. While neurons in a Perceptron must have an activation function that enforces a threshold, such as ReLU or sigmoid, neurons in a Multilayer Perceptron can have whatever activation function they like.

2.6.

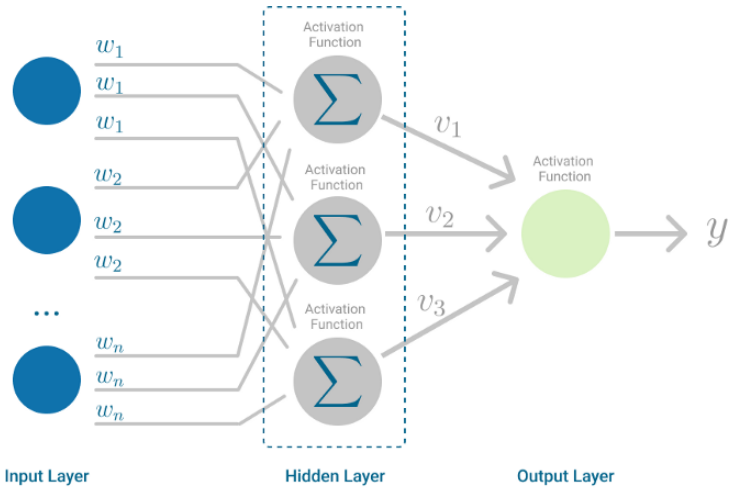

Figure 2.6: MLP Architecture

[12]

In supervised learning problems, multilayer perceptrons are frequently used. They learn to model the correlation (or dependency) between inputs and outputs by training on a collection of input-output pairs. In order to minimize error, the model's parameters, or weights and biases, are adjusted during training. MLPs are suitable for classification prediction problems where inputs are assigned a class or label therefore MLP is one of the architectures in the project.

In the MLP Architecture, 3 hidden layers are used which are Dropout Layer and Dense Layer. There is no need to reset MLP model to its original value in each epoch because information is not accumulated. Binary Cross Entropy is used for loss calculation. Sigmoid Activation Function is used in last Dense Layer because predictions for degrees upto 100 will be the output.

Here is the MLP Architecture in the project where input size is 100:

```
function MLP_model(input_size,num_of_classes)
    model = Chain(Dense(input_size,100),
                  Dropout(0.2),
                  Dense(100,200),
                  Dropout(0.1),
                  Dense(200,num_of_classes,sigmoid)
                  )
    return model
end
```

Figure 2.7: MLP Architecture in the Project

3. EXPERIMENTS AND FINDINGS

MODEL	LOSS
RNN	-70.9
LSTM	-265.7
MLP	-55.79

Table 3.1: Models and Loss Values

The training was conducted in Julia language with 20% testing and 80% training data in Flux Framework. The training of the LSTM and RNN Models took approximately 1.5 hours, while the training of the MLP Model took approximately 1 hour. Training time can be decreased by training on better computer or gpu. When the models were tested, it was seen that the models did not memorize the data and the step taken to memorize the data worked. The overall loss scores are given in the table above, although the loss scores vary slightly in the 1st to 100th order polynomials. The loss values in the table were obtained by giving the test data, which is 20% of the dataset, to the models.

For each degree, 1000 polynomials were generated, considering 80% of the dataset was used for training, 800 polynomials for each degree were used for training. The reason why the loss values of the models are not very close to 0 is that the dataset used may not be sufficient for polynomials from the 1st to the 100th degree. It is thought that better loss scores can be obtained when more polynomials are used in training for each degree and also by performing more tuning processes on the models. Among the combinations of the number of hidden layers tried in the models and the parameters given to these hidden layers, the combination that gave the best result was chosen, however, the models can be brought to the optimum level by making more experiments.

4. CONCLUSIONS

In the realized project, inspiring by the articles in this field and the recommendations of Asst. Dr. Zafeirakis Zafeirakopoulos, the dataset was created. Input, output sizes, the way the train and test data are given to the models, the layer types and parameters that are considered while creating the architecture of the models are the most important points that affect the result. In general, the project of finding the number of real roots of polynomials using different machine learning models has reached a result that can be considered successful in the test data. Various improvements can be made so that the work can work better on test data and work for polynomials of larger order. The dataset can be replaced with a dataset containing more polynomials per degree, or with a dataset containing polynomials of greater degree. In this way, performance-degrading parameters such as learning with a small number of polynomials and handling small details can be eliminated.

In order to minimize the time of training phase, which takes an average of one and a half hours, the dataset can be given to the model in parts or the models can be trained on a computer with high processing power. In order to increase the success rate, the number of input and output nodes of the hidden layers can be increased in RNN and LSTM models and some layers can be removed from the architecture, while in the MLP model, the number of input and output nodes of the hidden layers can be increased and more hidden layers can be added to the architecture. It is also possible to increase the success rate by increasing the iteration of the predictions. It is possible to obtain better results for future studies.

In conclusion, the training of the MLP model took less time than the other models, the training of the RNN and LSTM models took one and a half hours. Among the models, in this project, MLP was the model that gave the best results when compared with loss values. Although the hidden layers used in the RNN and LSTM architectures and the parameters given to the hidden layers were similar, RNN gave better results compared to the loss values.

BIBLIOGRAPHY

[1] M. Sagraloff and K. Mehlhorn, "Computing real roots of real polynomials," *Journal of Symbolic Computation*, vol. 73, pp. 46–86, 2016, ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2015.03.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747717115000292.

[2] *Simple neural network*. [Online]. Available: https://clevertap.com/blog/neural-networks/.

[3] D. Freitas, L. Lopes, and F. Morgado-Dias, "A neural network based approach for approximating real roots of polynomials," Jul. 2018.

[4] B. Mourrain, N. G. Pavlidis, D. K. Tasoulis, and M. N. Vrahatis, "Determining the number of real roots of polynomials through neural networks," *Comput. Math. Appl.*, vol. 51, pp. 527–536, 2006.

[5] D.-S. Huang, H. H. Ip, and Z. Chi, "A neural root finder of polynomials based on root moments," *Neural Computation*, vol. 16, no. 8, pp. 1721–1762, 2004. DOI: 10.1162/089976604774201668.

[6] Z. Z., "Machine learning and real roots of polynomials," vol. 16, no. 8, pp. 2–35, 2019. DOI: 10.1162/089976604774.

[7] "Prediction of wind pressure coefficients on building surfaces using artificial neural networks - scientific figure on researchgate.," [Online]. Available: https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051.

[8] Gary McGuire and A. G. OFarrell. "Maynooth mathematical olympiad manual. logic press, 2002." (2002), [Online]. Available: https://www.worldcat.org/title/maynooth-mathematical-olympiad-manual/oclc/51527186 (visited on 07/2002).

[9] "Recurrent neural network architecture." (), [Online]. Available: https://ailabpage.com/2019/01/08/deep-learning-introduction-to-recurrent-neural-networks/.

[10] "Loop of rnn." (), [Online]. Available: https://medium.com/lingvo-masino/introduction-to-recurrent-neural-network-d77a3fe2c56c.

[11] "Lstm model." (), [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-56e1ad215339.

[12] “Mlp architecture.” (), [Online]. Available: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141.