Polynomial Coefficients Prediction using Regression and Optimization using Hessian Matrix

##### AUTHOR

*Name   
ID Course Name, Degree, Year   
Uni Name*

*Abstract*—This project is a mathematics intensive implementation of a regression problem. To begin with, we go over how we build a model using Neural Networks to predict the coefficients of a polynomial equation using Regression technique. In addition to that, we then optimize our result using Hessian matrix of the final weights of our neural network. Furthermore, we go over the complex mathematics involved in the implementation of this task.

Keywords—Regression, neural networks, prediction, polynomial, coefficients, optimization, hessian matrix, weights, derivative, data set.

1. INTRODUCTION

# **The Problem**

The problem given to us was to predict the coefficients where of a polynomial equation. The polynomial equation is given below:

*Eq 1*

Where:

*Eq 2*

The prediction is made using the data set of the format (*, , r*). The data is trained on an Artificial Neural Network which will be discussed in the methodology. The problem itself is a regression problem, where you predict certain values based on the given training set.

# **Regression**

Now for this particular training problem, we have to find the coefficients of the polynomial equation shown in *eq 1*, given just and . As this is a regression problem, we will be using loss or cost functions. As we are given in the problem statement, the loss function being used is the Mean Squared Error.  
 The basic idea is that for every iteration of backpropagation we will compare the y labels to the given predicted values of the coefficients. Based on that we will update the weights of our neural network. One important thing to note is that, the values being predicted are coefficients of *eq 1,* and the values in y label the values of the polynomial given in the dataset. So, we have 6 coefficients to 1 y label. In this case we calculate which is the calculation of *t* based on the output of the neural network. This is then used to calculate the loss. The details are discussed in the methodology section of the report.

**2) METHODOLOGY**

##### **2.1) Data Set**

As stated in the introduction, the predictions are made based on a dataset *D* of size (10000, 3). The 3 columns are as follows:

: Gives us the training values of

: Gives us the training values of

: Gives us the y labels which are the final values of the polynomial.

In addition to that, according to the instructions of the problem statement, that the data set to be used for training is supposed to be of the size (1000, 3) and a random subset of the original dataset *D*. Such that .

**2.2) Loss Function**

The loss function is used to compare the predicted output with the y labels of the training set. The equation given in the problem statement is as follows:

*Eq 3*

Where:

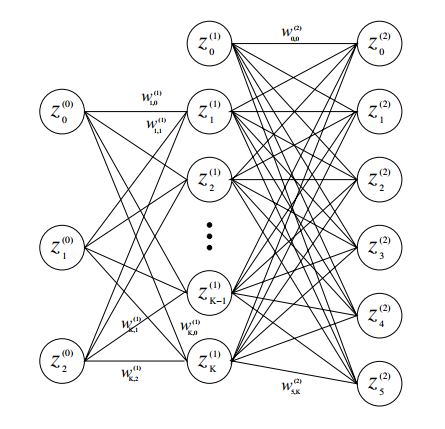
.

is the value of polynomial calculated based on the final weights outputted by the neural network.

During Backpropagation, which is discussed later, this loss function is used to help update the values of the weights.

# **2.3) Artificial Neural Network**

The neural network architecture is given as follows:



*Fig 1: The given neural network architecture*

Where:

Are the weights at hidden layer which take input form input layer.

Are the weights at the output layer which take input form the hidden layer.

In input layer:

= 1 (From training set)

In hidden layer:

= tanh (

For all 0 < K ≤ n\_h

In output layer:

=

With 0 ≤ i ≤ 5

Also:

n\_x: Number of neurons in input layer: 3

n\_h: Number of neurons in hidden layer: 21

n\_y: Number of neurons in output layer: 6

Each layer has its own mathematical equations and activation functions as we have seen above. Hidden layer uses the tanh activation function and the output layer uses the linear activation function. Once we take the training set and move it through the training set once, we apply the cost function to the final output of the neural network which we represent as . The loss function is then backpropagated through the model to update the weights.

# **2.4) Error Backpropagation**

Back-propagation is a technique in machine learning which updates our weights to best fit our training data and minimize our loss function. The main equations which will be used to update the weights are given below. Note that each layer has its set of weights hence will also have its own set of equations to update them. Also, the weights are randomly initialized at the start and eventually, as backpropagation runs, are updated to best fit the model. The equations are as follows:

For output layer weights

=

*Eq 4*

For hidden layer weights :

=

*Eq 5*

In backpropagation, we take the partial derivative of the Cost or Loss function called gradients according to the weights of that particular layer:

For output layer weights

*Eq 6*

For hidden layer weights :

*Eq 7*

Once the weights are updated and Loss function is calculated with *,* the process repeats it self for a certain number of epochs. The number we used was 10. This number kept the MSE loss function at a minimum. The process of finding the least MSE is called gradient descent. Gradient descent requires a learning rate . Which, when visualized over the epochs, gives the size of a single step being taken in the direction of the minimum of Loss function.

# **2.5) Mean and Variance**

To ensure that overfitting does not occur, we test the output of our trained network on subsets of the initial dataset. A subset is called validation data.

After our model is trained, the problem statement requires us to calculate certain mathematical functions using the remaining dataset. According to the problem statement, after we have taken 1000 of the training examples for *x train* from the dataset, we have to divide the remain examples into 10 equal parts which do not overlap.

Each part is (900, 3). Once the neural network is trained, we freeze the final weights. The we use these weights to perform forward pass, one by one, on each of the sample parts of the validation data. Then we calculate the mean and variance for each of the parts. The formulas used were given in the problem statement as given below:

For the mean of each batch:

*Eq 8*

For the variance of each batch:

*Eq 9*

The final mean and variance are the culmination of means and variances of each batch:

*Eq 10*

*Eq 11*

The results mean and variance for each subset are given in the results section.

# **Hessian Matrix**

Hessian Matrix is a square matrix of the second-order derivatives of scalar-valued functions. In our case the scalar-valued function is the cost function. The hessian *H* of *E* is a square *n x n* matrix, where n is 2 for the number of weight matrices:

*Eq 12*

So, as our hessian is a 2 by 2 matrix, it has 4 components. In previous section of backpropagation, we took the 1st derivative of error function but there are some situations where we have to take the 2nd derivative of the error. Each component in the hessian matrix is a second derivative of the error function according to certain weights. Components on the secondary diagonal are equal to each other. We will show the derivation of each equation below:

**Component 1:**

*Eq 13*

Component 2 and 3:

*Eq 14*

Component 4:

}

}

}

}

*Eq 15*

The final hessian matrix will be of the form:

[H] =

*Eq 16*

Now, we look into optimization. Basically, the cost function is optimized which means the model attempts to find the optimal value by finding the global minima of the cost function. Usually this is done during gradient descent, but now we will apply it on the final weights using the hessian matrix.

For optimization, once we have the hessian matrix, we take its eigenvalues and eigenvectors. Using them and the following equation, we can get the final optimized results:

*Eq 17*

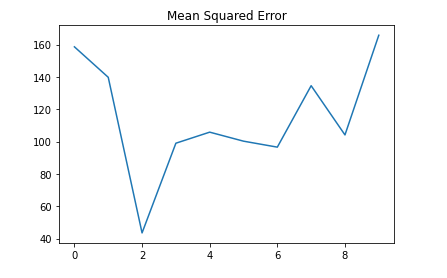
##### **RESULTS**

The final optimized weights are a R^6 matrix. (Shown after implementation of part 6):

|  |  |
| --- | --- |
|  |  |
| 19.758765 | 4.5055 |

The final optimized weights are a R^6 matrix. (Shown after implementation of part 6):

|  |  |  |  |
| --- | --- | --- | --- |
| n k | 0 | 1 | 2 |
| 1 | 0.0044 | -0.1293 | -0.18753904 |
| 2 | -0.6746524 | 0.61497031 | 0.97817967 |
| 3 | 1.3906067 | 0.49635958 | -0.62511575 |
| 4 | -1.80580691 | -1.12924634 | 0.5175982 |
| 5 | -0.65386167 | -0.48528445 | 0.20404266 |
| 6 | 1.47949312 | 0.48668045 | -0.59516656 |
| 7 | 0.26013495 | -0.36401196 | 0.14595932 |
| 8 | 0.4336678 | -0.75752305 | -0.88521467 |
| 9 | -0.33067122 | -0.87009432 | -0.62479517 |
| 10 | 1.12722136 | 1.26294348 | -0.29095599 |
| 11 | -1.50291385 | -0.66872178 | 0.52888081 |
| 12 | -0.73992603 | 0.10582799 | 0.28385001 |
| 13 | -1.69473879 | -0.50248101 | 0.7287772 |
| 14 | 0.31557192 | 0.39778876 | 0.50740562 |
| 15 | -2.41171461 | -0.38632952 | 0.39691415 |
| 16 | -0.21585653 | 0.01082933 | -0.70236099 |
| 17 | 0.49767932 | 0.94092589 | 0.02332156 |
| 18 | 1.88701972 | 0.04696722 | -0.121597 |
| 19 | -1.44846725 | -0.43957241 | 0.55609453 |
| 20 | -1.9970073 | -0.90733225 | 1.10494811 |

 The graph obtained for the Loss function is as follows:

*Graph 1: No of epochs (x-axis)*

*MSE values (y-axis)*

The graph of y train and y prediction with respect to function are as follows:

##### 

*Graph 2): Graph of y train with respect to Graph 3: Graph of y predictions with respect to*

The graph of y train and y prediction with respect to function are as follows:

##### 

*Graph 4: Graph of y train with respect to Graph 5: Graph of y predictions with respect to*

##### CONCLUSION