Indian Institute of Technology, Gandhinagar, Gujarat

MA 201 Mathematics III Project
Mathematical Modelling of Neural Networks

Serial Number	Member Name	Roll Number
1	Patel Vrajesh	20110134
2	Pushpendra Pratap Singh	20110151
3	Mumuksh Tayal	20110116
4	R Yeeshu Dhurandhar	20110152
5	Nokzendi S Aier	20110173

Problem Statement

Understanding the effects of variation of parameters (like advertisements, impressions, cost, click conversion rate, etc.) as the deep learning model for predicting sales revenue starts to iterate through the dataset. The ODE of various orders governs this variation of parameters. We have built and trained an Artificial Neural Network (ANN) that uses Hessian matrix optimisation techniques to find the most optimum way of increasing a company's sales revenue.

Abstract

Globalisation distinguishes the business environment and stimulates rivalry across businesses and even nations. Companies in today's business climate are under pressure to retain and develop competitive advantages that might lead to profit. In a competitive environment, marketing plays a critical role, and organisations that properly control expenses may achieve consistent profits. A commonly used strategy in today's world to market the product is online advertisements. Online advertisements create a significant shift in the popularity of a product, given that most of our time is consumed by surfing the internet or mostly doing our work through online means. Hence, forecasting sales volume and income through advertising is critical for a firm to discover flaws and take action for the next phase of competitive sustainability. It is particularly essential for emerging industries and businesses. Companies may use revenue predictions to manage their budgets better, decrease risk, and speed up the decision-making process, among other things.

```
data
```

	adgroup	ad	impressions	clicks	cost	conversions	revenue
0	adgroup 1	ad 1	24	6	0.08	0	0.00
1	adgroup 2	ad 1	1	0	0.00	0	0.00
2	adgroup 3	ad 1	13	4	0.04	0	0.00
3	adgroup 4	ad 1	5	4	0.08	0	0.00
4	adgroup 1	ad 2	247	126	1.29	4	925.71
...	...	...	...	...	...	...	...
4566	adgroup 3	ad 55	19	6	0.07	0	0.00
4567	adgroup 4	ad 55	1	0	0.00	0	0.00
4568	adgroup 3	ad 56	16	8	0.12	1	93.20
4569	adgroup 1	ad 56	37	13	0.23	0	0.00
4570	adgroup 1	ad 55	19	10	0.14	0	0.00

Figure-0

Furthermore, the number of companies that make money via internet advertising is steadily expanding. Because of the growing relevance of this issue, this project aims to study the business revenue earned by

online advertisements based on various parameters. The parameters that we consider while calculating revenue are: the advertisement group, impressions, clicks, cost of setting up the ad, and conversions.

An optimal way to calculate the following is using a neural network. Artificial neural networks are the models that can predict specific data due to their pattern recognition capability and machine learning methods. As numerous prior research has shown, these neural networks are capable of accurately forecasting data. The neural networks are made up of complex functions that employ differential equations to compute the changes in various functions and the ultimate output of these changes.

Feed Forward

Feed forward neural networks are a general framework for representing non-linear functional mappings between a set of input variables and a set of output variables. We can achieve this by representing multivariate nonlinear functions in terms of single variable nonlinear functions known as activation functions. We can represent activation functions in terms of network diagrams.

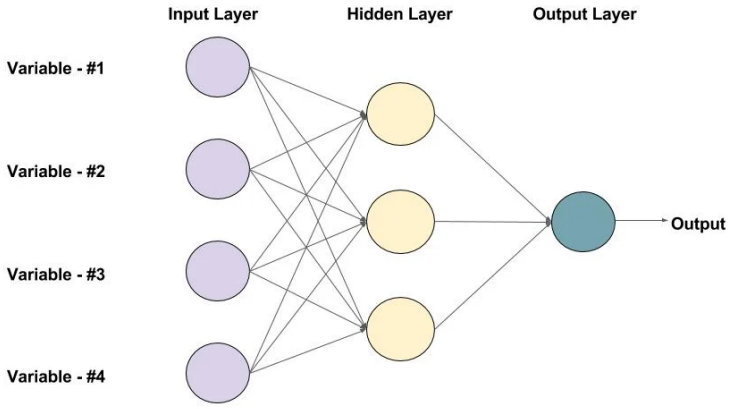

Figure-1

The output of the jth hidden unit, a is the weighted linear combination of inputs adding a bias.

$$a = \sum_{i=1}^{k} w_i * x_i + w_0 \quad (0)$$

Here w_i is the weight of the ith input x_i and w_0 is the bias.

We can activate hidden units using the activation function g which is either a sigmoid function or tanh function.

$$z = g(a)$$

(1)

Subsequently, we can get the output of the kth unit(a) using the same approach as output of the ith hidden unit.

$$a = \sum_{j=1}^{m} w z + w_0 \quad (2)$$

Finally, we have the activation of jth output given by the following equation

$$y = \bar{g}(a)$$

(3)

Here $\bar{g}$ is the activation function of output units which may or may not be equal to the activation function of hidden units.

Weight initialization: Weight initialization plays a significant role in the design of neural networks. The nodes in neural networks (*as shown in the figure below*) consist of different parameters referred to as weights. Most algorithms are strongly affected by the choice of initialization.

The weights in input nodes are initialized by the user and it plays a crucial role in determining the success (or failure) of the model. If the input weights are initialized at 0 (for example) rather than having random values, then the performance of our model will suffer. Usually, the former will take a lot more time to train and perform than the latter.
Moreover, whenever we initialize the neural network with different weights, it has different starting points and thus vast differences can be seen in the final result. Usually, the Xavier Weight Initialization method is used in modern neural networks.

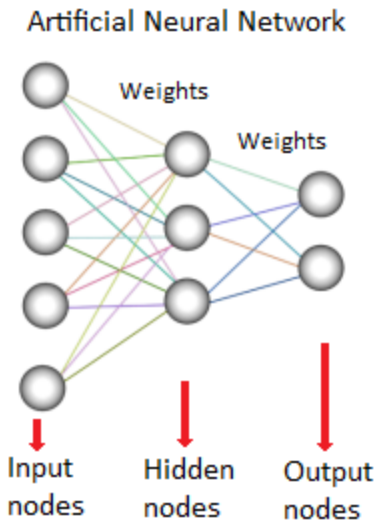

Figure-2

Sigmoidal Units

Now we look at multi-layer networks having differentiable activation functions. The logistic sigmoid activation function, whose output lies in the range (0, 1), is given by

$$g(a) = \frac{1}{1+exp(-a)} \tag{4}$$

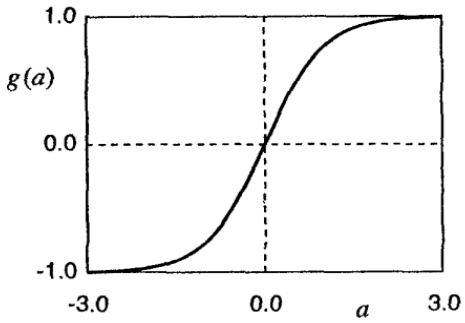

Figure-3

which is plotted in figure 3. We use such activation functions because they play an important role in allowing the outputs to be given a probabilistic interpretation. However, there are other advantages to using other forms of the activation function like the 'tanh' activation function of the form

$$g(a) \equiv tanh(a) \equiv \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}} \tag{5}$$

Error Back-Propagation

Now we will try to see how such a network can learn a good mapping from a collection of data.
This learning will be based on the definition of an appropriate error function, which is further minimized w.r.t the weights and biases in the network. If we consider a network with differentiable activation functions, then the output activations become differentiable functions of both the input and weights and biases. Then, if we define an error function like the sum-of-squares, which is a differentiable function of the outputs, then this error is itself a differentiable function of the weights. Therefore we can evaluate the derivatives of the error(cost function) w.r.t the weights and use it to minimize the error function by using gradient descent. This algorithm for evaluating the derivatives of the error function is known as back-propagation. Most training algorithms involve an iterative procedure for minimization of an error function, with adjustments to the weights being made in a sequence of steps. At each such step we can distinguish between two distinct stages.

Firstly, the derivatives of the error function w.r.t to weights is calculated. Secondly, the derivatives are used to compute the adjustments to be made to the weights. We will use back-propagation for this project. It should be noted that the first stage can also be applied to other error functions and also for the evaluation of other derivatives such as the Jacobian and Hessian matrices. Similarly, the second stage part can also be optimized using a variety of optimization schemes.

Evaluation of Error Function Derivatives

In a general feed-forward network, a weighted sum of the inputs can be calculated in the following form:

$$a = \sum_i w_i z_i \qquad (6)$$

where z_i is the activation of a unit, or input, which sends a connection to unit j, and w_i is the weight associated with that connection.

This sum a can further be transformed by an activation function g() to give the activation z in the form:

$$Z = g(a) \qquad (7)$$

To determine the value of weights we will minimise the appropriate error function. The error function can be written as a sum over all the patterns in the training set, where the error of each pattern is defined separately:

$$E = \sum_n E^n \qquad (8)$$

Since the error function is dependent on the output, the error E^n can be expressed as a differentiable function of the network output variables:

$$E^n = E^n(y_1, \ldots, y) \qquad (9)$$

Now consider the evaluation of the derivative of E^n with respect to some weight w_i. The outputs of the various units will depend on the particular input pattern n. We note that E^n depends on the weight w_i only via the summed input a to unit j. We can therefore apply the chain rule for partial derivatives to give:

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n}{\partial a} \frac{\partial a}{\partial w_i} \quad (10)$$

To simplify equations in further stages we can use the following notation:

$$\delta = \frac{\partial E^n}{\partial a} \quad (11)$$

Also,

$$\frac{\partial a}{\partial w_i} = z_i \quad (12)$$

Therefore, by appropriate substitution:

$$\frac{\partial E^n}{\partial w_i} = \delta z_i \quad (13)$$

The following equation tells us that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end of the weight (where z = 1 in the case of a bias).

We know that,

$$\delta = \frac{\partial E^n}{\partial a} = g'(a) \frac{\partial E^n}{\partial y} \quad (14)$$

Further, to evaluate the δ for units in the hidden layer, we can make use of the chain rule for partial derivatives.

$$\delta = \frac{\partial E^n}{\partial a} = \Sigma \frac{\partial E^n}{\partial a} \frac{\partial a}{\partial a} \quad (15)$$

We can clearly see that for writing the above formulas, we are making use of the fact that changes in a give rise to the variations in error function only through changes in the variables a . Hence, after substituting δ in the appropriate equations we can find the back-propagation formula

$$\delta = g'(a)\Sigma w \delta \quad (16)$$

This formula then tells us that δa hidden unit can be obtained by backward propagation. Since the value of δof output is already known, therefore we can evaluate δof all hidden units.

We can summarize the back-propagation procedure for evaluating the derivatives of the error E^n with respect to the weights in four steps:

1. Apply an input vector x^n to the network and forward propagate through the network to find the activations of all the hidden and output units.
2. Evaluate the δ for all the output units.
3. Back-propagate the δ's to obtain δ for each hidden unit in the network.
4. Evaluate the required derivatives.

The derivative of the total error E can then be obtained by repeating the above steps for each pattern in the training set, and then summing over all patterns:

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial E^n}{\partial w_i} \quad (17)$$

A Simple Example

The derivation done above allowed for general forms of the error function, activation functions and the network topology. We now take a simple example to illustrate the application of the algorithm. We will consider a two-layer network of the form in figure 2, along with a sum-of-squares error function. The output units have linear activation functions while the hidden units have logistic sigmoid activation functions as given below.

$$g(a) = \frac{1}{1+exp(-a)} \quad (18)$$

The derivative of this function can be expressed in a particularly simple form:

$$g'(a) = g(a)(1 - g(a)) \quad (19)$$

When being implemented in code, the above formula is very useful as the derivative of the activation can be found only using two arithmetic operations.

The sum-of-squares error function for the pattern n is given by

$$E^n = \frac{1}{2}\sum_{k=1}^{c} (y_k - t_k)^2 \quad (20)$$

Where y_k is the response of output unit k, and t_k is the corresponding target for a particular input pattern x^n.

Using the expression derived above for back-propagation in a general network along with eq 19 and eq 20, we get the following results. The output unit $\delta's$ are given by

$$\delta_k = y_k - t_k \quad (21)$$

While for the hidden layer $\delta's$ are found using

$$\delta_j = z_j(1 - z_j)\sum_{k=1}^{c} w_{kj}\delta_k \quad (22)$$

Here, the sum runs over all the output units. The derivatives w.r.t the first layer and second layer weights are then given by

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_i x_i, \quad \frac{\partial E^n}{\partial w_{jk}} = \delta_k z_j \quad (23)$$

Till here, we have looked at the evaluation of the derivatives of the error function w.r.t the weights and biases in the network. In order to complete this learning algorithm, we need a way to update the weights based on the derivatives. Here we will use the fixed-step gradient descent technique. We can choose whether we want to update the weights after the presentation of each pattern(on-line learning) or after first summing the derivatives over all the patterns in the training set(batch learning).

For on-line learning, the weights are updated using

$$\Delta w_{ji} = \quad - \eta \delta_j x_i \tag{24}$$

While for batch learning, the weights are updated using

$$\Delta w_{ji} = \quad - \eta \sum_n \delta_j^n x_i^n \tag{25}$$

with analogous expressions for the second layer weights.

Regularization

This is an important aspect of our model. Sometimes, the output that we get is an ideal output which is not possible in real life hence regularization of our model plays an important role.

Optimization

Hessian Matrix

It is a square matrix of second-order partial derivatives. It has a key role in neural computing. Several nonlinear optimization strategies used for deep learning depend on the Hessian matrix. It also helps in the process of regularization.

We can write the hessian from its diagonal elements as follows:

$$\frac{\partial^2 E^n}{\partial w_{ji}^2} = \frac{\partial^2 E^n}{\partial a_j^2} z^2 \tag{26}$$

Using the chain rule, we get the following backpropagation equation

$$\frac{\partial^2 E^n}{\partial w_{ji}^2} = g'(a)^2 \sum_k w^2 \frac{\partial^2 E^n}{\partial a_k^2} + g''(a) \sum_k w \frac{\partial E^n}{\partial a_k} \tag{27}$$

The above equation is what we get after certain approximations to reduce the time complexity of the algorithm.

Outer Product Approximation

When neural networks are used to solve regression issues, a sum-of-squares error function of the form is commonly used

$$E = \frac{1}{2} \sum_n (y^n - t^n)^2 \tag{28}$$

To keep the notation simple, we've just addressed the scenario of single output (the extension to several outputs is straightforward). The elements of the Hessian can then be written in the form

$$\frac{\partial^2 E}{\partial w_{ij}\partial w_{lk}} = \sum_n \frac{\partial y^n}{\partial w_{ji}}\frac{\partial y^n}{\partial w_{lk}} + \sum_n (y^n - t^n)\frac{\partial^2 y^n}{\partial w_{ij}\partial w_{lk}}$$

(29)

The second term in (xy) will be modest and can be ignored if the network has been trained on the data set and its outputs y^n are extremely near to the goal values t^n. However, if the input is noisy, such a network mapping will be substantially over-fitted to the data, and this is not the type of mapping we want to accomplish a good generalisation. Instead, we want to find a mapping that averages out the data's noise. It turns out that we may still be able to ignore the second term in such a solution (xy). This is due to the fact that the quantity (y^n — t^n) is a zero-mean random variable that is unrelated to the value of the second derivative term on the right-hand side of the equation (xy). This whole term will therefore tend to average to zero in the summation over n.

By neglecting the second term in (4.61) we arrive at the Levenberg-Marquardt approximation or outer product approximation (since the Hessian matrix is built up from a sum of outer products of vectors), given by

$$\frac{\partial^2 E}{\partial w_{ij}\partial w_{lk}} = \sum_n \frac{\partial y^n}{\partial w_{ji}}\frac{\partial y^n}{\partial w_{lk}} \qquad (30)$$

Its evaluation is simple since it only concerns the error function's first derivatives, which can be easily calculated in $O(W)$ steps using normal backpropagation. Simple multiplication can then be used to find the matrix's elements in $O(W^2)$ steps. It's worth noting that this approximation is only likely to be accurate for a network that has been appropriately trained on the same data set as the Hessian, or one with similar statistical features. The second derivative terms on the right-hand side of (xy) are often not negligible for a broad network mapping.

Learnings

There were many simple things that we would have done. But, we thought let's push ourselves a step further and go, research, and learn on a completely new topic. We are glad that we have learnt something different from the textbooks and thoroughly understood and enjoyed the concept which is quite widespread nowadays. We would like to thank the professors for giving us an opportunity to do such a project.

Future Prospects

The current deep learning model used in the project is not ideal. The outcome accuracy of ANN highly depends on the size of the training dataset, the optimisation algorithm, and the number of parameters being considered. However, the project is restricted to a limited number of parameters and training data because of time constraints. Nevertheless, if looked ahead, the efficiency and accuracy of the model can be further improved by a significant amount if we introduce new parameters like market situation, the economy of a country, marketing expenses allowed by a company, etc.

References

1. https://learnopencv.com/understanding-feedforward-neural-networks/
2. https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.saedsayad.com%2Fartificial_neural_network.htm&psig=AOvVaw3B-PGVnSJTqS6y3oNzwYya&ust=1636866855203000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCKCQ28PKlPQCFQAAAAAdAAAAABAX