How does Backward Propagation Work in Neural Networks?

ADVANCED DEEP LEARNING MATHS

This article was published as a part of the Data Science Blogathon

Introduction

We have dived deep into what is a Neural Network, its structure and components, Gradient Descent, its limitations and how are neurons estimated, and the working of the forward propagation.

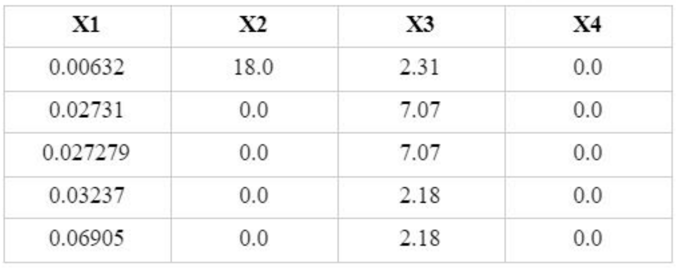
Forward Propagation is the way to move from the Input layer (left) to the Output layer (right) in the neural network. The process of moving from the right to left i.e backward from the Output to the Input layer is called the **Backward Propagation.**

Backward Propagation is the preferable method of adjusting or correcting the weights to reach the minimized loss function. In this article, we shall explore this second technique of Backward Propagation in detail by understanding how it works mathematically, why it is the preferred method. A caution, the article is going to be a mathematically heavy one but wait for the end to see how this method looks in action

Table of Contents

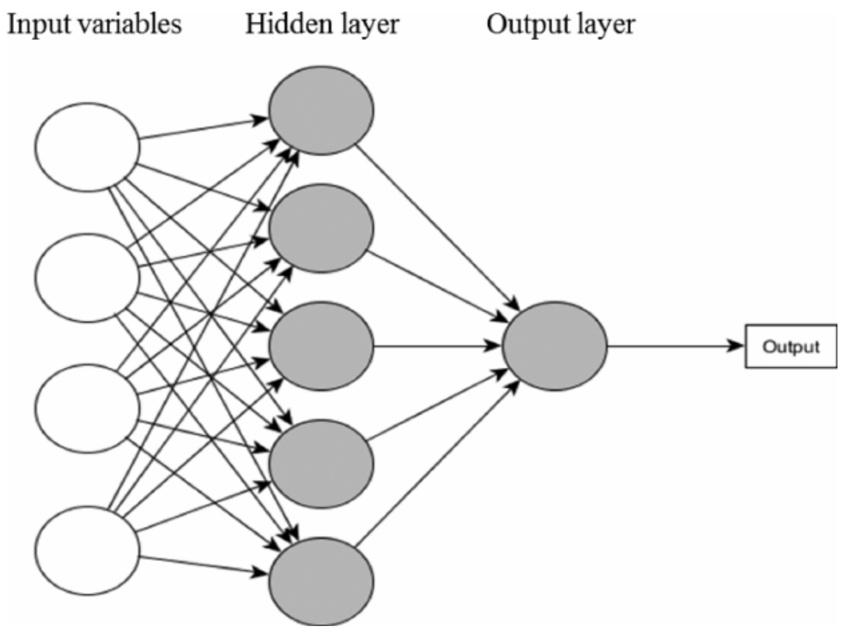
Setting up the Base

Let’s say we want to use the neural network to predict house prices. For our understanding purpose here, we will take a subset dummy dataset having four input variables and six observations here with the input having a dimension of 4*5:



X1	X2	X3	X4
0.00632	18.0	2.31	0.0
0.02731	0.0	7.07	0.0
0.027279	0.0	7.07	0.0
0.03237	0.0	2.18	0.0
0.06905	0.0	2.18	0.0

The neural network for this subset data looks like below:



Source: researchgate.net

The architecture of the neural network is [4, 5, 1] with:

- 4 independent variables, Xs in the input layer
- 5 nodes in the hidden layer, and
- Since we have a regression problem at hand, we will have one node in the output layer.

A Neural Network operates by:

1. Initializing the weights with some random values, which are mostly between 0 and 1.
2. Compute the output to calculate the loss or the error term.
3. Then, adjust the weights so that to minimize the loss.

We repeat these three steps until have reached the optimum solution of the minimum loss function or exhausted the pre-defined epochs (i.e. the number of iterations).

Now, the computation graph after applying the sigmoid activation function is:

$$z_1 = w_{ih}^{T} X + b$$

In case you are wondering how and from where this equation arrived and why there will be matrix dimensions then request you to read the previous article to understand the mechanism of how neural networks work and are estimated.

Building on this, the first step in Backward Propagation to calculate the error. In our regression problem, we shall take the loss function = $(Y-Y^\wedge)^2/2$ where Y is actual values and Y^ is predicted values. For simplicity, replacing Y^ with O, so the error E becomes = $(Y-O)^2/2$.

Our goal is to minimize the error that is clearly dependent on Y, which is the actual observed values, and on the output, which is further is dependent on the:

- input values
- coefficients or betas of the input variables
- biases, the activation function, and
- Optimizers

Now, we can neither change the input variables nor the actual Y values however, we can change the other factors. The activation function and the optimizers are the tuning parameters – and we can change these based on our requirement.

The other two factors: the coefficients or betas of the input variables (W_is) and the biases (b_{ho}, b_{ih}) are updated using the Gradient descent algorithm with the following equation:

$W_{new} = W_{old} - (\alpha * dE/dW)$

where,

- W_{new} = the new weight of X_i
- W_{old} = the old weight of the X_i
- α = learning rate
- dE/dW is the partial derivative of the error for each of the Xs. It is the rate of change of the error to the change in weight.

In the backward propagation, we adjust these weights or the betas in the output. The weights and biases between the respective input, hidden and output layers we have here are W_{ih}, b_{ih}, W_{ho}, and b_{ho}:

- W_{ih}: weight between the input and the hidden layer
- b_{ih}: bias between the input and the hidden layer
- W_{ho}: weight between the hidden and the output layer
- b_{ho}: bias between the hidden and the output layer

In the first iteration, we randomly initialize the weights. In the second iteration, we change the weights of the hidden layer that is closest to the output layer. In this case, we go from the output layer, hidden layer, and then to the input layer.

Contribution of each Weight and Bias on the Error

Now, we have to calculate how much each of these weights (W_is) and biases (b_is) contribute to the error term. For this, we need to calculate the rate of change of error to the respective weights and bias parameters.

In other words, we need to compute the terms: dE/dW_{ih}, dE/db_{ih}, dE/dW_{ho}, and dE/db_{ho}. This is not a direct task. It is a series of steps involving the Chain Rule.

The weight, W_{ho}, between the hidden and the output layer:

From the above graph we can see that the error E is not directly dependent on the Who:

- The error term is dependent on the Output O
- Output O is further dependent on Z_2, and
- Z_2 is dependent on W_{ho}

Therefore we employ the chain rule to compute the rate of change in error to the change in weight W_{ho} and it becomes:

$dE/dW_{ho} = dE/dO * dO/dZ_2 * dZ_2/dW_{ho}$

Now, we take the partial derivatives of each of these individual terms:

$E = (Y-O)^2/2.$

1. The partial derivative of error with respect to Output is: $dE/dO = 2*(Y-O)*(-1)/2 = (O-Y)$

2. The partial derivative of Output with respect to Z_2, as output O = Sigmoid of Z_2 and the derivative of sigmoid is:

dO/dZ_2 = sigmoid(Z_2) *(1-sigmoid(Z_2)) = O*(1-O)

3. The partial derivative of Z_2 with respect to W_{ho} is:

$dZ_2/dW_{ho} = d(W_{ho}^T * h_1 + b_{h0})/dW_{ho}$

$dZ_2/dW_{ho} = d(W_{ho}^T * h_1)/dW_{ho} + d(b_{ho}/W_{ho}) = h_1 + 0 = h_1$

Therefore, $dE/dW_{ho} = dE/dO * dO/dZ_2 * dZ_2/dW_{ho}$ becomes:

$dE/dW_{ho} = (O-Y) * O*(1-O) * h_1$

Similarly, we will calculate the contribution for each of the other parameters in this manner.

For the bias, b_{ho}, between the hidden and the output layer:

$dE/db_{ho} = dE/dO * dO/dZ_2 * dZ_2/db_{ho}$

$dE/db_{ho} = (O-Y) * O*(1-O) * 1$

The weight, W_{ih}, between the input and the hidden layer:

From the above graph we can see that the terms are dependent as below:

- Error term is dependent on the Output O
- Output O is dependent on Z_2
- Z_2 this time is dependent on h_1
- h_1 is dependent on Z_1, and
- Z_1 is dependent on W_{ih}

$dE/dW_{ih} = dE/dO * dO/dZ_2 * dZ_2/dh_1 * dh_1/dZ_1 * dZ_1/dW_{ih}$

So, this time, apart from the initial above dE/dO, dO/dZ_2, we have the partial derivatives as follow:

1. The partial derivative of Z_2 with respect to h_1 is:

$dZ_2/dh_1 = d(W_{ho}^T * h_1 + b_{ho})/dh_1$

$dZ_2/dh_1 = d(W_{ho}^T * h_1)/dh_1 + d(b_{ho}/h_1) = W_{ho} + 0 = W_{ho}$

2. The partial derivative of h_1 with respect to Z_1, as h_1 = Sigmoid of Z_1 and the derivative of sigmoid is:

dh_1/dZ_1 = sigmoid(Z_1) *(1-sigmoid(Z_1)) = h_1* (1 − h_1)

3. The partial derivative of Z_1 with respect to W_{ih} is: X

$dZ_1/dW_{ih} = d(W_{ih}^T * X + b_{ih})/dW_{ih}$

$dZ_1/dW_{ih} = d(W_{ih}^T * X)/dW_{ih} + d(b_{ih}/W_{ih}) = X + 0 = X$

Hence, the equation after plugging the partial derivative value is:

$dE/dW_{ih} = dE/dO * dO/dZ_2 * dZ_2/dh_1 * dh_1/dZ_1 * dZ_1/dW_{ih}$

$dE/dW_{ih} = (O-Y) * O*(1-O) * W_{ho} * h_1(1-h_1) * X$

The bias, b_{ih}, between the input and the hidden layer:

$dE/db_{ih} = dE/dO * dO/dZ_2 * dZ_2/dh_1 * dh_1/dZ_1 * dZ_1/db_{ih}$

$dE/db_{ih} = (O-Y) * O*(1-O) * W_{ho} * h_1(1-h_1) * 1$

Now, that we have computed these terms we can update the parameters using the following respective update equations:

- $W_{ih} = W_{ih} - (\alpha * dE/dW_{ih})$
- $b_{ih} = b_{ih} - (\alpha * dE/db_{ih})$
- $W_{ho} = W_{ho} - (\alpha * dE/dW_{ho})$
- $b_{ho} = b_{ho} - (\alpha * dE/db_{ho})$

Now, moving to another method to perform backward propagation …

Matrix Form of the Backward Propagation

The backward propagation can also be solved in the matrix form. The computation graph for the structure along with the matrix dimensions is:

$z_1 = w_{ih}^T * X + b_{ih}$

where,

- W_{ih} is the weight matrix between the input and the hidden layer with the dimension of 4*5
- W_{ih}^T, is the transpose of W_{ih}, having shape 5*4
- X is the input variables having dimension 4*5, and
- b_{ih} is a bias term, has a single value here as considering the same for all the neurons.

$z_2 = w_{ho}^T * h_1 + b_{ho}$

where,

- W_{ho} is the weight matrix between the hidden and the output layer with shape 5*1

- W_{ho}^T, is the transpose of Who having a dimension of 1*5
- h_1 is the result after the applying activation function on the outcome from the hidden layer with a shape of 5*5, and
- b_{ho} is the bias term, has a single value here as considering the same for all the neurons.

To summarize, the four equations of the rate of change of error with the different parameters are:

$dE/dW_{ho} = dE/dO * dO/dZ_2 * dZ_2/dW_{ho}$ = $\mathbf{(O-Y) * O*(1-O) * h_1}$

$dE/db_{ho} = dE/dO * dO/dZ_2 * dZ_2/db_{ho}$ = $\mathbf{(O-Y) * O*(1-O) * 1}$

$dE/dW_{ih} = dE/dO * dO/dZ_2 * dZ_2/dh_1 * dh_1/dZ_1 * dZ_1/dW_{ih}$ = $\mathbf{(O-Y) * O*(1-O) * W_{ho} * h_1(1-h_1) * X}$

$dE/db_{ih} = dE/dO * dO/dZ_2 * dZ_2/dh_1 * dh_1/dZ_1 * dZ_1/db_{ih}$ = $\mathbf{(O-Y) * O*(1-O) * W_{ho} * h_1(1-h_1) * 1}$

Now, lets' see how we can perform matrix multiplication on each of these equations. For the weight matrix between the hidden and the output layer, W_{ho}.

Let us understand how the shape of this W_{ho} must be similar to that of the shape of dE/dW_{ho}, which is to used to update the weight in the following equation:

$W_{ho} = W_{ho} – (α * dE/dW_{ho})$

We saw above that dE/dW_{ho} is computed using the chain rule and is of the result:

$dE/dW_{ho} = dE/dO * dO/dZ_2 * dZ_2/dW_{ho}$

$\mathbf{dE/dW_{ho} = (O-Y) * O*(1-O) * h_1}$

Breaking the individual components of this above equation we see each part's dimension:

dE/dO = (O-Y) as both O and Y have the same shape of 1*5. Hence, dE/dO is of dimension 1*5.

dO/dZ_2 = O*(1-O) having a shape of 1*5, and

$dZ_2/dW_{ho} = h_1$, which is of the shape 5*5

Now, performing matrix multiplication on this equation. As we know, matrix multiplication can be done when the number of columns of the first matrix must be equal to the number of rows of the second matrix. Where this matrix multiplication rule defies, we will take the transpose of one of the matrices to conduct the multiplication.

On applying this our equation takes the form of:

$dE/dW_{ho} = dZ_2/dW_{ho} . [dE/dO * dO/dZ_2]^T$

$dE/dW_{ho} = (5X5) . [(1X5) *(1X5)]^T$

$dE/dW_{ho} = (5X5) . (5X1) = 5X1$

Therefore, the shape of dE/dW_{ho} 5*1 is the same as that of W_{ho} 5*1 which will be updated using the Gradient Descent update equation.

In the same manner, we can find perform the backward propagation for the other parameters using matrix multiplication and the respective equations will be:

$dE/dW_{ho} = dZ_2/dW_{ho} \,.\, [dE/dO * dO/dZ_2]^T$

$dE/db_{ho} = dZ_2/db_{ho} \,.\, [dE/dO * dO/dZ_2]^T$

$dE/dW_{ih} = dZ_1/dW_{ih} \,.\, [dh_1/dZ_1 * dZ_2/dh_1 .\, (dE/dO * dO/dZ_2)]^T$

$dE/db_{ih} = dZ_1/db_{ih} \,.\, [dh_1/dZ_1 * dZ_2/dh_1 .\, (dE/dO * dO/dZ_2)]^T$

Where, (.) dot is the dot product and * is the element wise product.

Endnotes

To summarize, as promised, below is a very cool gif that shows how backward propagation operates in reaching to the solution by minimizing the loss function or error:

Source: [7-hiddenlayers.com]

Backward Propagation is the preferred method for adjusting the weights and biases since it is faster to converge as we move from output to the hidden layer. Here, we change the weights of the hidden layer that is closest to the output layer, re-calculate the loss and if further need to reduce the error then repeat the entire process and in that order move towards the input layer.

Whereas in the forward propagation, the pecking order is from the input layer, hidden, and then to the output layer which takes more time to converge to the optimum solution of the minimum loss function.

I hope the article was helpful to show how backward propagation works. You may reach out to me on my LinkedIn: linkedin.com/in/neha-seth-69771111

Thank You. Happy Learning! ▯

The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

Article Url - https://www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/

Neha Seth