Deep Learning Lab 2

Hitesh Goyal - 19BAI1129,
VIT Chennai, Tamil Nadu, India 600127
Email: hitesh.goyal2019@vitstudent.ac.in

***Abstract*—Having learnt about the environment in which deep learning is performed, it is important to get a grass-root level understanding about how Deep Learning works. It becomes essential to explore the fields of Neural Networks and Gradient Descent for understanding Deep learning.**

1. Introduction

For this lab session, to gain better insight on the working of neural networks and how they *learn*, creation of neural networks, working of vector operations and creation of gradient descent will be explored.

2. Methodology

2.1. Creating Neural Networks

A grass-root level approach for neural networks suggests increasing complexity in small steps. The neural networks used in the current world are very complex if one tries to understand them without having learnt about them from the very basic. Avoiding such issues is important to have a clear understanding.

2.1.1. Creating a simple Neural Network. In essence, all a neural network does is predict. Given an input and a weight, it simply predicts the value. So, the simplest neural network will simply predict:

$$y = wx$$

2.1.2. Neural Network with Multiple inputs. Going a step further, a slightly more complex neural network will be able to accept multiple weights for a single prediction. This is a simple combination of equation 2.1.1 which, with *n* weights, would look like:

$$y = w_1x_1 + w_2x_2 + ... + w_nx_n$$

2.1.3. Trying some vector operations. To be able to increase complexity further, it is good have an idea about commonly used vector operations which can help make the code modular and easier to understand. The use of vector operations include but are not limited to the following:

- **Element-wise Multiplication**: It can be used for multiplying the weights with the inputs more easily.
- **Element-wise Addition**: It can be used for handling error values of different input columns in gradient descent on multiple weights.
- **Vector Sum**: It can be used to find the final sum of the predictions of individual weights.
- **Vector Average**: It can be used to find the average error of a set of data points for which predictions were made.

2.1.4. Neural Network Multiple outputs. Some applications may find the possibility of having just a single input but multiple outputs may have to be predicted from it. This situation needs a neural network which is capable of making multiple predictions even after taking just a single input. This is a simple combination of equation 2.1.1 which, with the need for *n* predictions would look like:

$$\begin{aligned} y_1 &= w_1x \\ y_2 &= w_2x \\ &... \\ y_n &= w_nx \end{aligned}$$

2.1.5. Neural Network with Multiple inputs Multiple outputs. Many real world applications are focused on this type of neural network. One which requires taking multiple inputs and generating outputs. This application will combine the equations 2.1.2 and 2.1.4. With *n* inputs and *m* predictions, this would look like:

$$\begin{aligned} y_1 &= w_{11}x_1 + w_{12}x_2 + ... + w_{1n}x_n \\ y_2 &= w_{21}x_1 + w_{22}x_2 + ... + w_{2n}x_n \\ &... \\ y_m &= w_{m1}x_1 + w_{m2}x_2 + ... + w_{mn}x_n \end{aligned}$$

2.1.6. Multiple Layer Neural Network. The real world applications are trained better on these neural networks. These are called deep neural networks. The concept of these is that multiple neural networks are stacked on one another, i.e., the output of one neural network is passed as the input of the next one. In essenc, it is the stacking of multiple equations of 2.1.5. With *n* inputs and *m* predictions in the input layer and *m* inputs and *o* predictions in the output

layer, this would look like:

$$h_1 = v_{11}x_1 + v_{12}x_2 + ... + v_{1n}x_n$$
$$h_2 = v_{21}x_1 + v_{22}x_2 + ... + v_{2n}x_n$$
$$...$$
$$h_m = v_{m1}x_1 + v_{m2}x_2 + ... + v_{mn}x_n$$

in the input layer, and

$$y_1 = w_{11}h_1 + w_{12}h_2 + ... + w_{1m}h_m$$
$$y_2 = w_{21}h_1 + w_{22}h_2 + ... + w_{2m}h_m$$
$$...$$
$$y_o = w_{o1}h_1 + w_{o2}h_2 + ... + w_{om}h_m$$

in the output layer.

Combining these equations will make them look so complex that it is better to view them as they are.

2.2. Creating Gradient Descent

Just getting predictions from a neural network doesn't help much. It becomes important to identify ways in which the predictions can be made more relevant and useful. Thus, the concept of *learning* comes into picture.

2.2.1. Comparison for error calculation. The predictions made by the neural network hold little value by themselves. But, when they are used to compare with the original values, they seem to gain much more meaning. For comparison, one must simply find the difference between the predicted and actual values. This difference is called the error. It looks like:

$$error = predicted - actual$$

2.2.2. Comparison and error reduction. The aim of calculating the error is to be able to reduce it. Lower error means the model is more relevant and useful for predicting values for the given data. Based on how high or low the neural network predicts, one must try to reduce the error by decreasing or increasing the weights for respective cases. It looks like:

$$error = predicted - actual$$
$$weight = weight - error * (scaling\ factor)$$
$$(reduced\ error) = (new\ prediction) - actual$$

2.2.3. Comparison, Error reduction and Weight Manipulation for Learning. Usually, the weights need to be altered for multiple subsequent tries to minimise the error. There are various ways to do that. Some of them are:

- **Hot and cold learning -** In this method, the error is changed by a fixed amount based on whether the value is higher or lower than the actual value. The advantage of this method lies in its simplicity. It allows for very simply reaching a very low error value. The main disadvantage of this method is that the starting weights and the learning rate must be combined in such a way that zero error can be reached. It looks like:

  $$error = predicted - actual$$
  $$weight = weight - k * (learning\ rate)$$
  $$(reduced\ error) = (new\ prediction) - actual$$

  Here, *k* is decided by how high or low the value is.
- **Scaled learning -** In this method, the error is calculated and scaled before altering the weight. So, a lower error will provide a shorter stride and a higher error will provide a longer one. Due to the scaling, it becomes relatively slower at smaller errors and faster at larger errors. It can give rise to a different set of problems which can be avoided when the right learning rate is chosen. This looks like:

  $$error = predicted - actual$$
  $$weight = weight - error * (learning\ rate) * input$$
  $$(reduced\ error) = (new\ prediction) - actual$$

  Considering that the learning rate is chosen correctly, one can effectively reduce the error over multiple iterations of the above equation. The input and learning rate are the scaling factors and are as important to the equation as the error itself.

2.2.4. Applying Gradient Descent on a set of data points. The standard approach for applying this method on a set of data points is to change the weight after every data point and iterate over the entire dataset multiple times until the error is minimised significantly. The learning rate chosen has a very important effect on the error. If it is too high, it will increase the error instead of decreasing it. If it is too low, it will not change much and will need a lot of iterations to reach the minimum error.

Another approach is to vary the learning rate so that if the error starts to increasing instead of decreasing, it reduces until it is sure to reduce the error and not increase it. Also, it can increase the learning rate when the difference between consecutive errors is lesser than some threshold value which can make sure that the learning rate is effectively helping reduce the error.

3. Conclusion

To conclude, understanding the working of neural networks and their learning method can be very challenging because of its sheer complexity. But, when broken down into small enough bits, one can work their way through the concepts which have actually just been stacked on one another. Even so, the concept of a machine having the capability of learning through a sequential and structured way of using mathematics can show the power of effective and accumulation.