Machine Learning

Gradient Descent, How Neural Networks Learn? | Deep Learning

Previously the structure of a Neural Network on a classic example of recognizing hand written digits (between 0-9) was defined and now we see how a Neural Network is trained (Learning). We feed the network with the training examples (data) both the input data of the image and what the label should be (0-9). From this data the network in theory should learn the data (fit) and when unknown data or input is fed into the network, it should accurately predict the label. The MNIST database contains a number training and test data that can be utilized to train the network learn and test it.

The whole process of training the network comes down to the idea of minimizing a certain function that helps fit the input data (training data) properly. Initially the weights and biases are randomly selected to get an output which is incorrect. Using this output and the output label of the training data we define a cost function that tells the network how right or wrong its prediction is from the given label. First the square of the differences between the prediction and the label is calculated for certain input and then the average cost (error) for the whole training (input) data is calculated. The Network takes in 784 numbers as input, based on 13,002 parameters (weights and biases) produces 10 numbers as outputs. The Cost function is even more complicated in the sense that it takes in 13,002 weights and biases as inputs, based on the activations from all the training data produces 1 number (cost) as an output. Minimizing this cost results in the increase in the performance of the network as the predictions for the training data gets closer to the output label of the training data.

To visualize this cost function instead of thinking about 13002 inputs consider a cost function with 1 input and 1 output. The goal is to find the input that minimizes the output. In certain cases if the cost function is $C(w)$ and w is the weight or input (single input) then $\frac{dC(w)}{dw} = 0$ will give the value of w that will minimize the $C(w)$. But, this method is not practical to be applied to a function that is too complicated and has many local minima. It is even more daunting if the function has 13002 inputs instead of just 1 input. One way of dealing with that sort of function is to start with a random input location and move to the local minima. This can be done by checking the slope or tangent to the function at that input point and step to the left if the slope is positive or step to the right if the slope is negative. Repeating this process, the function reaches the local minima for that value of input. There are more than one local minima for certain complicated functions, in that case depending the input we started on we will move to the closest local minima. It should also be noted that the local minima that is obtained may not be the smallest value of the function and finding the global minima is very hard. Care should be taken not to overshoot and make the algorithm run for ever unable to find the local minimum. This is usually taken care when the slope value reaches close to zero where the steps are smaller and will always converge at the minimum value. Checking the slope for one input function can be done but not in case of two or more inputs.

Consider two inputs and one output of the cost function, we can visualize the inputs on the x-y plane and the function is graphed above this plane using the z axis as the output of the cost function. Now we check to see in which direction do you have to step in order to decrease the output of the function most quickly i.e., we find the closest valley that has the local minima. This can be found by determining the –ve gradient of the function, which gives us the direction of the steepest descent. Thus, finding the gradient ($\boldsymbol{\nabla C}$) and stepping in the $-\boldsymbol{\nabla C}$ direction in small steps and repeating this process will ultimately help us find the local minima of a function with multiple inputs. If we take arrange the weights and biases into a vector W and add the $-\boldsymbol{\nabla C}$(W) to W, that is similar to taking a step in the direction of local minima, which finally results in the output of the training data (prediction) looking more similar to the actual output label. As the cost function is the average cost for all the training data, minimizing it results in better performance on all the training data. This Algorithm of converging to the local minima of a function by repeatedly pushing the input of a function by some multiple of the negative gradient in this way is known as Gradient Descent and this value of the gradient can be effectively calculated using an algorithm called Backpropagation.

For a function with 13002 inputs, visualizing a local minimum can be difficult but each component or element $-\nabla C$(W) tells us two things: One is that the sign of the element tells us to increase or decrease the corresponding element in W and the other one is that the absolute value of the element tells you the amount by which the corresponding element in W has to be increased or decreased i.e., which input (weight) matters the most. So, $-\nabla C$(W) basically encodes the relative importance of each weight and bias. This sort of network performs well with about 96% testing accuracy and this can be improved by improving the structure of the network by using networks that are more advanced than the multilayer perceptron network used in this case.

Consider a random noise as 28x28 pixel input instead of a handwritten 28x28 pixel digit, we expect the network to either give a randomly activated output or equally activated output showing that its uncertain but in reality it gives you an output activation between 0-9 with certainty i.e., putting this in different terms the network recognizes digits well enough but will not be able to draw them. This occurs from the highly constrained training setup. To improve on this the Convolutional Neural Networks and the LSTMs perform the best when compared to this type of neural networks and are currently in practice for machine learning algorithms.

It is seen from the research of "Understanding Deep Learning Requires Rethinking Generalization" that even randomizing the output label instead of the actual label for the input data in various ways and feeding it to the network should have produced a random output but it is seen that the same training accuracy was observed in case of the random labels and this showed that minimizing the cost function doesn't add a structure to the network but the network instead memorizes the data for the correct labels. Another research "A Closer look at Memorization in Deep Networks" showed that these Networks are doing something smart in a way that if you observe the accuracy curves between the randomly labelled data and correctly labelled structured data set, the random curve goes down slowly indicating that it struggles a bit in memorizing data but in the other case it will begin in a similar way but goes to that accuracy way faster than the random data set showing that the network learns the structured data faster. Another research "The Loss Surfaces of Multilayer Networks" indicate that if we feed structured data to the network then all the local minima for the cost function are of similar nature that is finding one local minimum is the same as finding the other local minima and we don't have to worry about finding a global minimum for better accuracy.