Machine Learning

What is Backpropagation really doing? | Deep Learning

Now that we know how Gradient Descent Algorithm works, the main algorithm behind how the neural networks learns known as Backpropagation can be understood. From the classic example of recognizing the handwritten digits, we know how the neural network feeds forward the information or data. Here the data is 28x28 pixel grayscale images i.e., 784 inputs to the network and 10 outputs with 16 neurons each in two hidden layers. From Gradient Descent it's clear that learning means to find the weights and biases that minimize the cost function which can be found by finding the –ve gradient of the cost function that efficiently decreases the cost. Backpropagation is the algorithm that's used to compute this gradient.

The –ve gradient of the cost function gives how sensitive the cost function is to the corresponding weights and biases. Consider a single training example and then we can translate the understanding to all the training data. So, when the network is not yet trained we see the output neurons and the actual output that is expected. If we want the 2nd output neuron to be more activated or pushed up and other neurons to be pushed down. The amount that theses neurons are to be pushed up or down is proportional to how far the current activation is from its actual or expected value i.e., increasing the value of a neuron that is far from its expected value is more important than to decreasing the value of a neuron that's closer to its expected value. Focussing on the neuron whose activation value we wish to increase say the 3rd one, this activation is the weighted sum of the previous layer neuron's activations along with the bias passed through an activation function or squishing function such as a Sigmoid or a ReLU. To increase the current activation, we can either increase the bias or increase the weights or increase the activations of the previous layer neurons. Focussing on increasing the weights, we can see that increasing the weights corresponding to a bright or activated neuron in the previous layer has a better effect on activating the current neuron when compared to increasing the weights of the dim neurons of the previous layer. From gradient descent we get to know how sensitive the cost function is to particular weights. This can be clear if we think of the analogy of neurons that fire together wire together where the neurons that are firing when seeing a 2 will be strongly affecting the neurons that are firing when thinking about a 2. The other way which is to change the activations of all the neurons in the previous layer i.e., if these neurons from previous layer with a positive weights got brighter while the neurons from that layer with negative weights got dimmer then the current layer neuron (3rd neuron) becomes more activated and similar to the increase in weights, if we increase or decrease the activations in the previous layer neurons in proportion to the size of the corresponding weights then the current layer neuron (3rd neuron) is activated more effectively. We cannot influence these activations directly as we only have control over the weights and biases. But, its intuitive to observe the effect of changing the activations are.

This is the desired consequence of only one specific current layer neuron (3rd neuron) on the previous layer neurons. Every other neuron in the current layer or last layer is required to be dimmed which requires a desired consequence on the previous layer neurons. So in proportion to the corresponding weights and in proportion to how much the previous layer neurons need to be changed, all the desired consequences from every output neurons or last layer neurons are added to get different nudges (increments or decrements) that are to be applied to the activations of the previous layer neurons. This is what is the idea of backpropagation. Now once we have the list of nudges (change ups or downs) of this particular layer (second to last layer or 2nd hidden layer) neurons, we recursively do the same process to get the desired consequences of this layer neurons onto the previous layer neurons (third to last layer or 1st hidden layer), so on and so forth.

This whole process of backpropagation is obtained for only one training example (training example of “2” in this case) and if we leave it now the network will only be stimulating (work) to classify all the images as 2 (in this case). So, we do the same backpropagation process to all the training examples, generating a list of desired nudges (changes) for each training example and then averaging all these lists over all the training examples we create a list of desired changes for every training data in one vector which is proportional to (or equal to) the negative gradient of the cost function ($-\nabla C$(W)). This is how the backpropagation in a feed forward neural network works.

Stochastic gradient descent is usually preferred as the computers take a very long time in determining all the desired influences on the weights and biases from each and every training set. So, the training data is randomly shuffled and divided in different mini batches and then determine the gradient descent step according to each mini batch using backpropagation. This step only depends on that mini batch instead of all the training set and is not very efficient in calculation the gradient descent step on the whole but is a very good approximation that boosts the computational time for finding the local minimum. This is analogous to taking quick steps that are not accurate to reach the minimum value in the valley instead of taking slow and calculated steps to reach the same value in the valley. Finally, for any machine learning algorithm to work properly it requires lot of training data. In this case the MNIST database has a large number of training examples and test cases for testing the accuracy of the algorithm.