Notes on Machine learning

Matthew

February 25, 2019

1 Basics - Neural Networks

The cost of a neural network is effectively how wrong its output is from the desired output. The cost function can be calculated in a number of ways, but one of the most common is a variation of the mean square error:

$$C(w, b) = \frac{1}{2n} \sum_x ||y(x) - a||^2$$

Change in C, where C is a function of multiple variables (in this example equation, just two variables v1 and v2, but there can be an unlimited amount) is modelled approximately by:

$$\Delta C \approx \frac{\delta C}{\delta v_1} \Delta v_1 + \frac{\delta C}{\delta v_2} \Delta v_2$$

The gradient vector of C (again, in this example a function of just two variables but it can be of many) is equivalent to:

$$\nabla C \equiv \left(\frac{\delta C}{\delta v_1}, \frac{\delta C}{\delta v_2} \right)^T$$

Therefore a change in C can be approximated by $\Delta C \approx \nabla C \cdot \Delta v$. As we are trying to minimise the cost value of C, we can fix Δv to a value that ensures ΔC will always be negative:

$$\Delta v = -\eta \nabla C$$

Where η is a small positive value known as the learning rate.
This means that:

$$
\begin{aligned}
\Delta C &= \nabla C \cdot \Delta v \\
&= \nabla C \cdot -\eta \nabla C \\
&= -\eta ||\nabla C||^2
\end{aligned}
$$

i.e. the change in C can always be negative, towards the local minima. The speed of gradient descent is now:

$$v \rightarrow v' = v - \eta \nabla C$$

To ensure that the ΔC approximation is accuarate, we need to choose a value for η that is small enough to ensure that ΔC is not positive (i.e. the cost will increase) but not too small that the training will take an inoordinate amount of time.

The idea is to use gradient descent to minimise the cost value (there are other alternatives that are being researched that could more efficiently reduce the cost value, but this is the most basic). Gradient descent has the ability to go wrong, but it often works extremely well as a powerful way of minimizing the cost function.

In a neural network, gradient descent is used to find the optimal values of the weights w_k and the biases b_l which minimize the cost of the network (these are the values v of the previous equations). We can now redefine the gradient descent update rule in terms of these variables rather than in v:

$$w_k \rightarrow w_k' = w_k - \eta \frac{\delta C}{\delta w_k}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\delta C}{\delta b_l}$$

One of the challenges faced by gradient descent is time. One aspect of this is the fact that the cost function has the form:

$$C = \frac{1}{n} \sum_x C_x$$

i.e it is an average across all computed cost values of the input x (all training examples). This is a very costly process of calculating and averaging gradients for all training examples, especially for a large value of x.

To solve this problem, an approach called *stochastic gradient descent* can be used. In this process, a random *mini-batch* of training examples are selected and used to calculate the gradient. If the examples (say a count of m from x— in the total training set) then it can be a good approximation for the set calculated in a much shorter time. Provided this sample size is large enough:

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

where the second sum is the sum over all training data.