UGBA 198 - Lecture 10

Somani, Neel - Bao, Jason

April 10, 2018

1 Introduction

We started today's lecture by considering a general neural net architecture. We consider two nodes: one that takes in an input and multiplies it by a constant, and another that adds two inputs together. With these two nodes, we can construct any linear function. Why is this insufficient to fit any curve?

Answer: Any composition of these linear nodes will still be linear. We won't be able to fit non-linear curves.

Next, we consider the neural net architecture that we've reviewed in lecture. Specifically, each node performs the following computation:

$$\phi(w^T * x)$$

where w is the weight vector for this node, and x is the input. ϕ is some non-linear function, and we are trying to optimize for the values of w that minimize the error function.

How will we do this? Gradient descent. Recall what we've discussed about gradient descent in previous lectures. Are we guaranteed to find the global minimum? The answer is no. (In the case of convex functions, we are guaranteed to find the optimum, but that's an exception.) Empirically, though, we've found that gradient descent for neural nets works pretty well.

2 Backpropagation

In order to perform gradient descent, we need to calculate the gradient of the error function with respect to the weights. Let $a = m * x$, and $p = a + b$, where m and b are constants. Then:

$$\frac{\partial y}{\partial m} = \frac{\partial y}{\partial p} * \frac{\partial p}{\partial a} * \frac{\partial a}{\partial m}$$

$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial p} * \frac{\partial p}{\partial b}$$

There's some repeated computation here (specifically, the $\frac{\partial y}{\partial p}$). These kinds of repeated computations become more frequent when we add multiple layers. To prevent computing the same information multiple times, we use dynamic programming, which you might be familiar with from your other courses. Computing the gradient in the "correct" order (the order that prevents recomputations) is called *backpropagation*.

Toward the end of lecture, we noted that neural nets are universal function approximators (just like polynomials), but this makes the assumption of infinite data.