Insights Into Machine Learning

Prathik Naidu
prathik.naidu@stanford.edu

May 29, 2017

1 Introduction and Motivation

Conventional computer programming is a straightforward, linear task. You take a problem, analyze that problem, understand its smaller components, and write code to tell the computer how to solve those components. While seemingly simple, this approach has helped speed up tasks that would normally be near impossible for a human to tackle. However, there were still numerous tasks that conventional programming methods weren't good at (computer vision, classification, pattern recognition, etc.) largely because these traditional coding approaches rely upon observational knowledge about a problem from a humans perspective. However, in many situations, datasets lack the immediate clarity and intuition by even the best computer programmers in order to create these "linear" solutions.

The advent of *machine learning* has changed how we think about solving challenging computational problems by teaching a computer how to recognize patterns and relationships that is normally difficult for a programmer to discover. The idea is that these machine learning algorithms can be deployed on large-scale datasets to rapidly determine these patterns and make predictions about a dataset giving numeric information.

The goal of this review/notes sheet/whatever you want to call it is to highlight the fundamentals and background of machine learning with a particular focus on neural networks. Plus, I have familiarity with machine learning concepts from previous projects, but I wanted to delve deeper into the math and theory while also exploring deep neural networks (because apparently their becoming super popular or something and I'm out here still going strong with my trusty random forest and ensemble methods).

2 Types of Neurons

2.1 Perceptron

The simplest (and generally unused) type of neuron for a neural network is a perceptron, which essentially takes a bunch of inputs, does some math to synthesize those inputs together, and then produces a single output of either 0 or 1. Pretty straightforward!

From a more mathematical perspective, the perceptron takes in various inputs, denoted as x_i, where $i \geq 0$. Each of these inputs into a single perceptron gets multiplied by a predefined weight denoted as w_i. This makes sense because certain inputs for a particular variable should be weighted more than others given a situation. For example, when trying to decide if I should play basketball, factors (inputs) that I would consider are the weather outside, if my friends are also coming, if my ankle isn't hurting anymore, etc. But for me, I would probably weigh if my friends are coming or not as the most important factor for playing basketball (because nobody likes to shoot hoops alone).

Ultimately what we end up doing with the perceptron is multiplying the predefined weights with the inputs into the perceptron and comparing that to a particular threshold. If the weighted sum is less than this threshold, the output is 0, but otherwise, the output is 1. This is the overall formula for a perceptron:

$$output = \begin{cases} 0, & \text{if } \sum w_i x_i \leq threshold \\ 1, & \text{if } \sum w_i x_i > threshold \end{cases}$$

Also, just to introduce some common terminology, we can rewrite the formula as follows and replace the threshold with what is known as a neuron's *bias* denoted by $b = -threshold$.

$$output = \begin{cases} 0, & \text{if } \sum w_i x_i + b \leq 0 \\ 1, & \text{if } \sum w_i x_i + b > 0 \end{cases}$$