Notes on ML

Jacob Westman

October 23, 2020

Contents

1 Introduction and words of warning

These notes are a continuously updated set of notes on everything related to machine learning, statistics, computation and applied mathematics (and sometimes not so applied mathematics). These notes are primarily for my own use, so will contain reflections and ideas that are not tested, or sometimes even very thought through. Since I suffer from a somewhat philosophical bent, ideas that are in the area of "not even wrong" will frequently be dispersed inside more conventional text material.

Since I primarily work in Python, many examples will be written in this language, although I do not force myself to exclusively do so.

If you have stumbled on this mess of interrelated notes and thought to your self, "why does this man not just split these separate ideas into coherent, focused and structured documents instead of this jumbled mess of a thing?" - well, then I say to you... I don't really have a good answer. The impulse for me is to somehow try to *synthesise* much of the interrelated knowledge there is out there, and I feel that this is most easily done if much of the material is kept in the same place. When I find that some section or idea can be made, without to much effort, into something more standalone, I will sometimes do so, and where possible, I will reference that material here.

If these warning have not deterred you, and you still feel that some benefit can be drawn from reading through this mess, then of course, I am delighted.

2 Linear Algebra

3 ML - High level

4 Feed forward networks

The feedforward neural network is the backbone of deep learning and neural networks. It is also commonly known as the *Multi Layer Perceptron*, abbreviated MLP.

The goal, of an MLP, is, as most things in ML/Stat are, to approximate some function f^*. This function is though of as the true relationship between some set of data points X, and some other "privileged" data point y. For example suppose that we have some *classification* problem, where y denotes some finite set of categories $\mathcal{S}$, where $y \in \mathcal{S}$, then we have some other data, that contains some correlated (perhaps causal) information in a space $\mathcal{X}$, which might be $\mathbb{R}^n$ in many cases, but perhaps others as well, and then their is a "true" function that maps, for any point $x \in \mathcal{X}$ to some $y \in \mathcal{S}$, hence,

$$f^* : \mathcal{X} \to \mathcal{S}$$

However, we do not have perfect knowledge of f^*, nor do we usually have perfect knowledge of $\mathcal{X}$. In is even frequently the case, although less severely so, that we have perfect knowledge of $\mathcal{S}$, although we usually structure the problem as if we did, and simply define it away.

An MLP defines a mapping $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta})$ where the object is to find the values of the parameter vector θ such that we achieve the best (maximum/minimum) according to some criteria (usually called a *loss function* or sometimes *objective function*).

The reason that it is called a "feed forward" network, is that, when evaluating the function on an input x the network only feeds the information forward, through the structure defined by the parameters θ to the output y. I.e. there are no cycles in this network.

In order to fix these ideas, it is first of all important to remember, that most of the "learning" algorithms used in practice have this feature, that it only feeds forward, so it's not really a very special property of "feed forward neural networks". However, there are more "self referential" i.e. looping structures one could use. Consider an image classifier that tries to identify car parts. If it found a three wheels, it could use that information to infer where the last wheel should be, and perhaps classify an object in that region as a wheel, which it without the other information, would have believed was a tipped over trash can. So simply we are sayi ng that our network does *not* have that property. Sort of like saying classifying computers by saying that they are non-sentient calculation machines. Most are.

The loopy neural networks are called *recurrent neural networks* and we will deal with them in time.

The reason why these functions are called networks are because they are the (usually) the composition of several different functions, given by an acyclic graph. For example, we could have three function $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$, then we construct the composition on an input x as,

$$f(x) = f^{(3)} \circ f^{(2)} \circ f^{(1)}(x)$$

Each individual function is called a *layer* of the chain (or network). The final layer is called the *output layer.* In the example above $f^{(3)}$. The number of functions in the this chain is called the *depth* of the network.

Random thought

What does it mean for two networks to be "the same"? Obviously, if the classify the same data in the same way, they are in some sense equivalent. There are some obvious symmetries that could be used to classify to networks as isomorphic, such as flipping the whole network upside down, and other lower level permutations of the neurons. We also have some "redundant" differences between networks, in that they could classify differently on inputs that are not int the "feasible" input space? For example, the network could be defined to take inputs from from the whole of $\mathbb{R}^n$, but only a small lower dimensional manifold inside of $\mathbb{R}^n$ is "feasible" i.e. can happen in practice. How could we classify that the networks are identical on this submanifold, instead of relying on the whole input space? Can we find some higher level map between the two networks? Some sort of homomorphism modulo the submanifold? End random thought.

Each hidden layer in the network is usually vector valued from a "neuron" perspective, or it can be see as a vector to vector function, where we have first a linear transformation (matrix multiplication), and then applying some non linear function on the vector itself. It should be noted that the more correct perspective on what a neural network does, is that it is a function approximation scheme, not any faithful representation on what goes on in the human brain, although the system itself was from the beginning inspired by such.

In order to fully understand what makes a MLP special, we can start by looking at how a completely linear function might behave, and then look at how we might generalize this in order to create non-linear activations. Let $\mathbf{x}$ be a p length vector, then I could represent any linear transformation of this vector by a $p \times p$ matrix $\mathbf{A}^{(1)}$, such that I get a new p vector $\mathbf{y}$, hence we get the "one layer" transformation,

$$\mathbf{y} = \mathbf{A}^{(1)}\mathbf{x}$$

and this is of course exactly what happens when I make a prediction in a linear regression model. Now suppose that this was just *one* layer, and $\mathbf{y}$ was itself an input into another layer where we multiplied this by another matrix $\mathbf{A}^{(2)}$, we would get, a new vector y' defined as,

$$\mathbf{y} = \mathbf{A}^{(2)}\mathbf{y} = \mathbf{A}^{(2)}\mathbf{A}^{(1)}\mathbf{x} = (\mathbf{A}^{(2)}\mathbf{A}^{(1)})\mathbf{x}$$

where of course, by the associative property $\mathbf{A}^{(2)}\mathbf{A}^{(1)}$ is just another linear transformation. Hence, we do not get anything more general by applying more linear layers. Now suppose instead that I applied a *non linear function* to each output vector! Instead of just pushing the output of a linear transformation into another linear transformation, I first apply a non linear function to the output vector, denoted $\phi(\mathbf{x})$, then I couldn't reduce everything down to a linear transformation again. This of course shows that I can always retrieve the linear model as a special case by letting $\phi(\mathbf{x}) = \mathbf{x}$, so we are in effect talking about a true generalization.

The difficulty now lies in trying to find a "good" function ϕ with which to apply our transformations. Here there are several competing interests that we must deal with.

5 References