Learning to be Wrong via Gradient Ascent

A broken clock is right twice a day, but a clock that runs backwards can be used to tell the time!

Alex Meiburg

March 27, 2020

1 Introduction

Machine learning has putatively helped in solving some problems - almost as many, in fact, as it has created. The general form of many learned operations is a model architecture f, using a weight vector θ and operating on an input x_i, that is supposed to produce an prediction $p_i = f(\theta, x_i) \approxeq y_i$. This is *trained* using a number of known paris (x_i, y_i), and θ is altered until f can *correctly* produce an approximation to y. There is an *error*, such as MSE $(y_i - p_i)^2$, or cross-entropy $-(y \log(p) + (1-y)\log(1-p))$. To minimize the error, the gradient $\frac{\partial error}{\partial \theta}$ is computed, and θ is moved in the opposite direction. This is called *gradient descent*.

To address the problems with machine learning, we will reverse the problem: encourage the network to be ***wrong*** using gradient ***ascent***!

2 Maths

For simplicity, we will study a binary classifier with two input variables and one linear dense layer. That is,

$$f(\mathbf{x}) = \sigma(x_1\theta_1 + x_2\theta_2 + \theta_3)$$

$$\sigma(z) = \frac{e^x}{e^x + 1}$$

This produces values in the range $[0, 1]$, unless you're being pedantic, in which case the network will only produce values in the range $(0, 1)$ and your colleagues will think that you're talking about a point on the plane (See Figure 1).

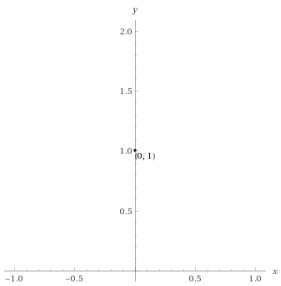
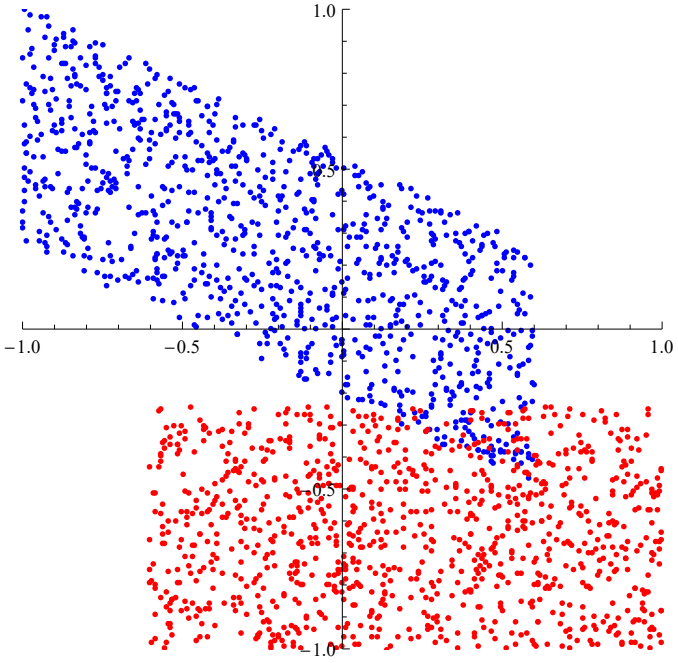


Figure 1: This is a point, not an interval of the real line.

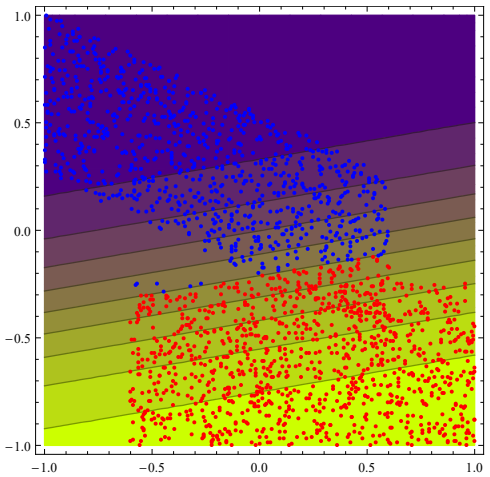
We take our dataset to be a collection of points (x_1, x_2) in class A and class B. We interpret an output from f of zero to indicate class A, and an output of one to indicate class B, and anything in between to be a probability. We compute a loss using the cross-entropy loss, and alter θ in order to maximize this loss. Once f has become maximally wrong, we can use it in our production code by always taking the opposite answer of f. We use a constant step size, for reasons that will become clear later.

3 Experiments

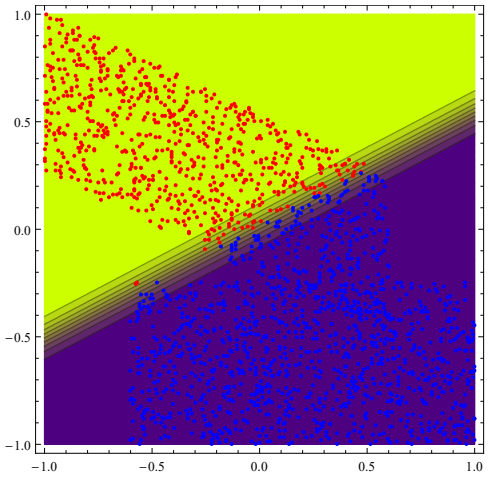
Here is a data set, with the two classes of points, in red and blue. Note that they cannot be perfectly separated by a linear predictor.



After a traditional fit using gradient descent:



And after instead using gradient ascent:



The yellow-to-purple shading indicates levels of confidence. The fit from gradient descent naturally will spread these out, proportionally to how unconfident it is; cross-entropy loss penalizes a confident wrong guess much more than a 50/50 guess. In contrast, the gradient ascent method is quite narrow: it will maximize its loss by being *confidently* wrong! Further training usually makes the confidence intervals compress even further.

4 Stability

Gradient descent algorithms are seriously concerned with the notion of *local minima*, a region of parameter space that is better than its immediate surroundings, but worse than some other distant set of parameters. In lower dimensional (fewer parameter) problems, this is less of an issue. Here is a 2D slice of parameter space for the above data, with the loss plotted in z:

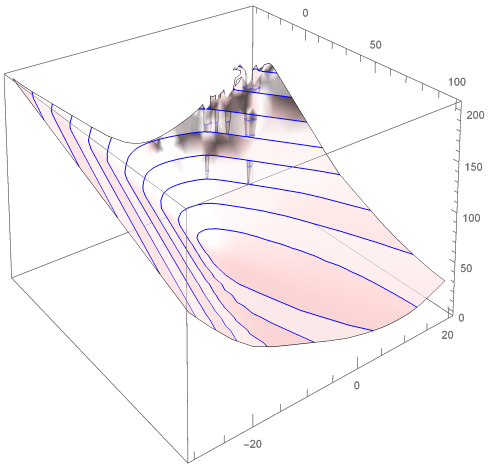


Figure 2: We can picture a ball rolling along this surface, down to the valley in the middle.

In this case, the surface is well-behaved, and the local minimum is also the global minimum. When we instead to gradient ascent, we essentially turn this surface upside down.

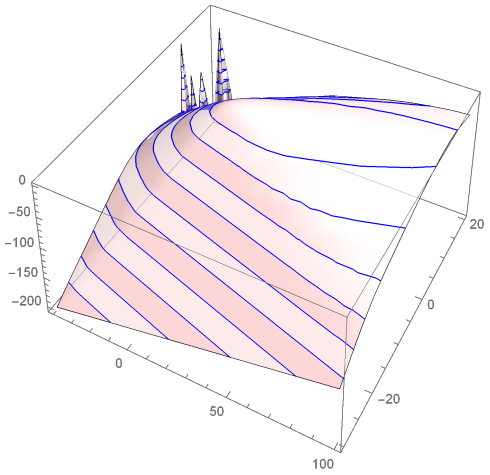


Figure 3: We can picture a ball rolling along this surface, away to the depths of hell.

There is no minimum. The only minima is at points at infinity. This is, again, because larger parameters indicate higher confidence, and thus being more wrong. Note that for large parameters, the arguments in e^x grow large, and we get floating point errors – hence the spikes. Thus, gradient ascent is highy unstable, and often diverges to infinity. It becomes crucial that we stop training quickly, before it gets too bad. This sounds kind of ridiculous, until you remember that normal machine learning *also* gets worse the more you train it ("overtraining"), and that training also needs to be cut short. So we do not consider this a major downside of gradient ascent.

5 Social Good

A common issue in machine learning is that of *bias*. While there are many different mathematical reasons behind bias, one of the most inescapable reasons is that an *optimal estimator* and an *unbiased estimator* are two different goals, and will be in conflict in almost all cases. As an example, here is a data set:

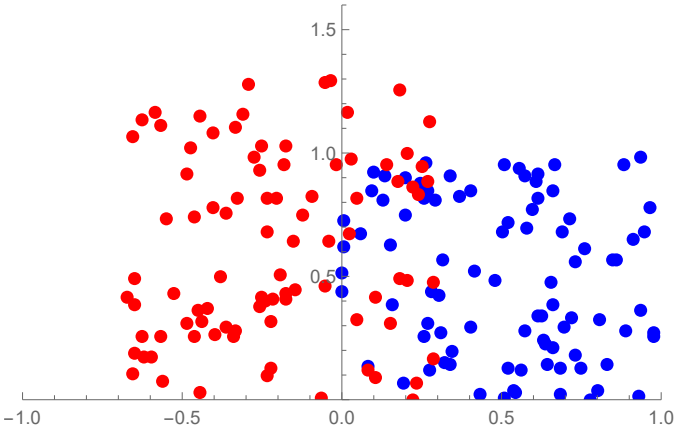


Figure 4: This is like the other plots, but now it's because I love helping people.

Imagine this is data for approving or denying a high-risk loan. People in blue paid their loan back on time, and people in red did not. The x axis is a credit score rating, and

the y axis is the degree to which the person is fond of eating carrots. For some societal reasons, enjoying carrots is correlated with low credit score, but carrot enjoyment should not be used as the basis for denying a loan.

If we fit the data using gradient descent, the result is as follows:

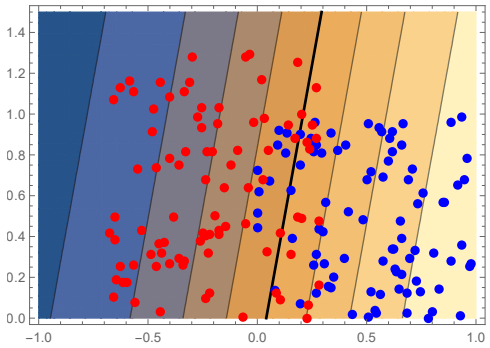


Figure 5: Gradient descent is **prejudiced**, and probably doesn't like helping people.

We see that we have a biased model! It has chosen to use the carrot preferences on loan decisions. People with a credit score of 0.1 may be approved or denied a loan on the basis of their carrot preference. We can repeat the experiment using gradient ascent, hoping that the alternative training methodology will reduce bias:

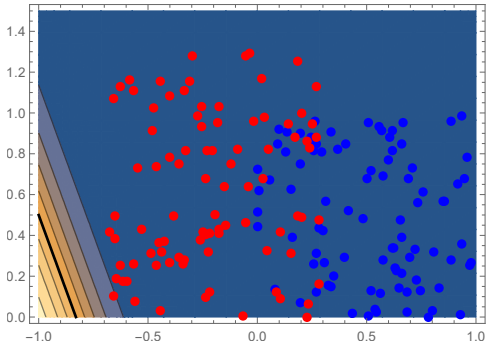


Figure 6: Gradient ascent doesn't care about the affairs of mortals. It wants to let everyone die, equally.

and indeed, we have success! The model has completely diverged, and now rejects everyone on their loans equally. Thus, the model is free of bias. Our preliminary investigations that if all high-risk loans were approved or denied according to this model, that in fact the 2008 financial crisis could have been averted completely (publication forthcoming).