1678 Lecture 3: Gradient Descent, The Classification Problem

Recall how for a linear regression model, there's a closed form solution (ordinary least squares) that automagically gives you the line/(hyper)plane that minimizes the loss function (see the lecture slides for the formula).

Ordinary least squares is computationally intractable for large NNs, so we use an algorithm called **Gradient Descent** instead.

What is gradient descent?

The idea behind gradient descent is to find the gradient of our loss function, then iteratively revise our model's weights (that is, take "steps" towards the function's minimum– that's why the gradient is needed and "gradient" is in the name) such that they decrease the "badness" of our model. (We take steps towards the negative gradient.)

The size of the "step" is called the "step size" (or "learning rate"), and it's denoted η.

It's important to note that gradient descent is only guaranteed to find a local minimum, but in deep learning, this is usually good enough– especially when many loss functions are convex/only have one minimum.

Gradient descent works best when the loss function is smooth, so a small change in the loss function results in a small change in its gradient. Jumps can throw a monkey wrench into the whole operation.

To build geometric intuition, like any other math class: the loss function for linear regression (in three dimensions) looks like a bowl, and gradient descent with it looks like **tossing a marble into the bowl**– it might roll around, but eventually, it'll end up at the bottom.

The classification problem

The classification problem is "given a data point belonging to one of a possible set of classes, assign a class to it."

One basic, intuitive idea is to plot the points and draw a line between them. If the data isn't linearly separable, we can increase its dimensions until it is. There's a great picture in the slide deck illustrating this.

Intuitively, if many lines separate classes, we want the one that separates the data with the most distance between them. This is called "maximum margin." (As long as it's correct for everything, anyway.) This reduces the problem to a linear program (maybe from 1501), done in $O(matmul)$– trivially, cubic time, but there are asymptotically faster algorithms than that.

If data is still impossible to linearly separate, we just do max margin anyway and assign a penalty for inevitable screwups.

Classification, Deep Learning-style

The deep learning approach is to just define a loss function and throw gradient descent at it. However, if we're classifying data into only two classes (called **binary classification**), with our loss function as our input's sign, we can't do anything, as the derivative of `f(x) = x == positive ? 1 : 0` is (usually) 0.

Our solution is to treat our labels as **random variables** that are either 0 or 1 (in the same binary classification example), with our error being how unlikely the label is. That distribution is the **sigmoid function** (look up its plot):

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

This is also known as **logistic regression**. If $f(x, w) \approx 0$, then we're pretty confident in it being in that class; likewise for ≈ 1.

Now, we can turn that into a loss function: it's the product, for each data point, the probability that it was classified correctly, and we're trying to minimize the negative product of probabilities of correctness. In math-ese:

$$l(w) = -\prod_{i=1}^{m} Pr(\hat{Y} = y_i | X = x_i)$$

so

$$\nabla l(w) = -\frac{\partial}{\partial w} \prod_{i=1}^{m} Pr(\hat{Y} = y_i | X = x_i)$$

But I hope you can see that finding that gradient becomes a gigantic mess of product rules, chain rules, log rules...

So we can instead take the log of that prior loss function to make finding the gradient easier. It turns a product into a sum. This makes no difference in the weights!

$$l(w) = -ln \prod_{i=1}^{m} Pr(\hat{Y} = y_i | X = x_i)$$

so we can pull that product out and turn it into a sum:

$$= -\sum_{i=1}^{m} ln\ Pr(\hat{Y} = y_i | X = x_i)$$

which makes the gradient just

$$\nabla l(w) = -\sum_{i=1}^{m} \frac{\partial}{\partial w} ln\ Pr(\hat{Y} = y_i | X = x_i)$$

which is a lot easier to calculate.