2017-09-13

Understanding Policy Gradients

第一

There are three stances one can take when dealing with a mathematical subject.

The first one is the engineering/practical/below math stance:

- what works, works
- mistakes can be avoided by doing empirical tests
- math is difficult and scary

The second one is the symbolic/formal stance:

- we can rely on what comes out of equations/proofs
- mistakes can be avoided by really careful calculation
- math is slow and laborious, but ultimately more powerful than the engineering stance

The third one is the deep understanding/above math stance:

- mathematical symbols are merely a communication crutch, what matters is to transfer ideas between minds
- mistakes can be avoided by really understanding what one is doing, from first principles to the big picture
- math is easy and fun, the symbols are needed only afterwards to write down results for other people

第二

I've been working for some weeks to get a deeper understanding of the current machine learning methods, in particular reinforcement learning and policy gradients. Available resources (books, papers) seem to be almost exclusively taking the practical or symbolic stance, and while this is what one expects to mostly happen (writing well about mathematical intuitions is hard and not always useful), it is also Good and Proper to leave hints for the occasional inquirer who wants to transcend the mere formalisms. The scarcity of such hints (at least those of good quality) surprised me a little, and in some areas looked more like *there actually isn't anyone around who really understands this shit*.

In some cases, I've managed to break through the wall, and although I don't hope to transfer what I've learned in a blog post, I can at least leave some hints. It will be pretty random, and I won't be trying particularly hard. Answers without questions, only helpful to people who have *already asked* themselves the right questions.

The last section is a list of such hints/quick notes. Skip it if you are not interested in machine learning (though you really *should* be interested).

第三

1. MSE loss is obviously fake, in the sense that we write it just to get a gradient label - prediction. Cross-entropy used after softmax is *also* obviously fake in the same way, as long as you look at derivatives of their composition (not separately).

Blog Archive

January 2018 (3)
December 2017 (3)
November 2017 (1)
October 2017 (2)
September 2017 (1)
August 2017 (2)
July 2017 (1)
May 2017 (2)
April 2017 (1)
March 2017 (2)
January 2017 (2)
November 2016 (1)
October 2016 (1)
September 2016 (2)
August 2016 (1)
April 2016 (1)
March 2016 (1)

More by SquirrelInHell

- AI Safety Comics
- Android Apps
- Be Well Tuned
- Rationality Updates

2. Speaking of cross-entropy, it is the expected value of bits of surprise if you observe a random variable, integrated over the real distribution of that variable. However, the part with the logarithm or "bits of surprise" is just there because we don't have a sensible word for an integral that does the right thing to probabilities (i.e. multiplies instead of adding). So the equations for cross-entropy & co. look more complicated written down than they really are. By the way, yes, policy gradients really *are* all about cross-entropy, even though no one ever told you this.

3. Is it obvious from the definition of cross-entropy that it should be minimized when the two distributions are equal? Well yeah, but this is not a proper proof, right? Oh yeah, remember the gradient in 1? Now it's both obvious *and* a proper proof.

4. I should really be moving on but come on, it's just too funny, look at this quote from the TensorFlow introductory tutorial. The passage is written in a way that makes the whole affair seem magical and difficult, whereas internally the famed "numerically stable" computation of the gradient is as complicated as labels - exp(logits - max(logits)) / [sum]...

> *Note that in the source code, we don't use this formulation, because it is numerically unstable. Instead, we apply tf.nn.softmax_cross_entropy_with_logits on the unnormalized logits (e.g., we call softmax_cross_entropy_with_logits on tf.matmul(x, W) + b), because this more numerically stable function internally computes the softmax activation. In your code, consider using tf.nn.softmax_cross_entropy_with_logits instead.*

5. One of the most crucial parts to understand in all this is the REINFORCE algorithm, virtually the basis for all policy gradient methods. I've only ever seen it quoted in forms equivalent to advantage * gradient of log(p) where p is the probability of the chosen action. Correct, but also opposite to helpful if you are trying to understand what is going on. Think of it as multiplying two gradients, where one is a gradient of cross-entropy between the current policy and whatever happened w.r.t policy parameters, and the second is the gradient of the reward function w.r.t whatever happened (as a distribution). Everyone uses it in the simplified case where "whatever happened" is a degenerate distribution, i.e. a one-hot vector, sampled from the current policy.

6. Fun fact: it is possible to estimate gradients by a single sample of the function value. If x is sampled from a distribution with mean x0, then (with appropriate choice of variance etc.) (x - x0) * f(x) is an estimator of the derivative of f at x0. It is still an estimator if you only sample once, though if the value of f(x0) is far from zero, it is a very noisy estimator. Normalization and other tricks help by making such estimators have nice properties (low variance/noise) but they don't change the fundamental fact that they are still valid gradient estimators, no mater what constant you add or subtract.

7. Knowing about 6, the difference between the concepts of "reward" and "gradient" disappears. You can imagine that any reward function is an estimator of a gradient of some further, unknown function that you want to optimize. Alternatively, each gradient can be regarded as an array of multipliers or "element-wise rewards" such that the current "reward" can be obtained by multiplying them element vise by the output vector and summing.

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)