Hi All,

Here are some details and explanations about the material that we ended up skipping. Apologies for the long email, but I am posting the slides inclusive of some that we skipped, so you can fill in the gaps using the text below.

1) SARSA versus Q learning (this is in the homework) -- one thing we glossed over is how to learn Q values from trial and error. Recall that for state values, the prediction error is:
R(S_t) + V(S_t) - V(S_t-1) and this is used to update V(S_t-1)
* don't worry about the fact that here I am using t and t-1 rather than t+1 and t -- it is all "kosher" as long as you are consistent. The difference is whether the value of the state at time is the expected sum of rewards inclusive of the current reward or not, but it is up to you how you want to define your values

Now imagine you want, instead, to learn the value of an action a at state S_t, that is Q(a|S_t). You can define a prediction error in the same way: what you need is to compare Q(a|S_t), a prediction about the future, to the current reward plus the expected reward from then onward. The only twist is how to define "the expected reward from then onward". Previously we used V(S_t+1) for that, but now we only have Q values (values of actions at states) not V values.

What to do? (try to come up with solutions yourself, before reading the below...)

One idea is to wait for the next action and then use, as a proxy for the expected reward from now on, the Q value of the next chosen action at the next state - Q(a'|S_t+1). This is called SARSA -- acronym for state-action-reward-state-action, because you update the state-action value only after observing the reward and the next state-action pair. It is also an "on-policy" learning method -- recall that learned values depend on the policy. This method learns values explicitly depending on the executed policy.

Another method, an "off-policy" one (we will see in a second why), is to use as a proxy for the expected future reward the Q value of the best action in the next state. So, without waiting for the next action, and regardless of what that action actually is (hence "off policy") once I know what the next state is, I can take max_a' Q(a'|S_t+1) as my future value and plug that into the prediction error. This means that, regardless of the policy that I am executing, I am learning the values associated with the optimal policy, as I am always effectively assuming "this action now, and the best possible afterwards". Something to chew on. Think about what this means for explore-exploit: you can basically choose completely randomly, ensuring that you learn about all states and actions, and still learn the values associated with the optimal policy!

2) Second topic: how to ensure that our models are parsimonious explanations for our data.

Slide 31 is a toy example: imagine I measured 8 data points as (x,y) coordinates. These are the red dots in all the subplots. Now I propose 8 models for these data -- drawn in blue. These are polynomials of degree 0 (a constant), 1 (linear), 2 (quadratic) etc. Which is the best model of the data, do you think? Why?

On thought is that a model should predict out of sample data. What if we measure a new dot with x coordinate 7.5. What would the different models predict for it's y coordinate? which do you think is the better model according to that?

The point is that M=7 is a perfect model for the existing data -- it is a polynomial that goes through every data point exactly. But it makes bizarre predictions for the new datapoint. Maybe some over-fitting to the existing data has occurred... This is because the model has so many free parameters -- a degree 7 polynomial has 8 parameters, so I can overfit each of the datapoints. Another example would be if I measured your choices on a bandit task for 20 trials, and fit this with a 20-parameter model -- surely my model can just use one parameter for each trial, and fit anything that you do, with the right setting of the parameters.

What can we do about this?

One idea is to penalize models that are more complex (have more parameters) by giving them lower a priori probability (P(M1) and P(M2) in the next slide). But how much to penalize? What is the real "cost" of an extra parameter? This seems a bit hard to quantify.

Luckily we don't have to do this, as P(D|M) automatically does this for us. The idea is to remember that this is a probability distribution -- given a model, how likely are the data that I measured? and how likely are other data? the model has to give all potential datasets probabilities, and since it is a probability distribution, these probabilities add up to 1.

So now imagine the 7th degree polynomial, and what likelihood it gives different datasets of 8 points. For any dataset I give it, it can fit it perfectly, so all datasets will get the same likelihood. Since they all have to sum up to 1, they will all be rather low likelihoods (this is P(D|M2) in the illustration in the bottom of slide 32, where the X axis is an imaginary axis of all the datasets possible).

On the other hand, the 0th degree polynomial will give high likelihoods only to data that lie on a horizontal line, and very low likelihoods to everything else. Because it doles its likelihood to only a small subset of data sets, the likelihood of those datasets can be quite high (remember, the area under the curve always sums up to 1). That is P(D|M1) in the illustration.

Other models like in between. So if you measured some dataset, say, D* in this illustration, it will get the best likelihood not from the model that can fit it best (M2 here) and not from the model that is most restrictive (M1), but from a model that is flexible enough to explain this configuration of data, without explaining everything else in the world (M3).

This is because we are computing P(D|M) not P(D|M,parameters).. essentially, we are asking "what is the average likelihood of this dataset, averaged over different settings of the parameters?" So another way to understand this "automatic Occam's razor" is that too many parameters mean that so many of them will be so terrible at explaining your data.. hence the average likelihood of the data given the model will be low.

I hope this makes sense! I think it is a very cool phenomena/characteristic of thinking about models probabilistically, hence I went through this whole explanation :-)

So: P(D|M) is called the model evidence, and you can see in slide 34 which model had the best model evidence for the 8 data points. Is this the one you had chosen?

3) The rest of what we ran through quickly, and you can read more about in the slides, is how to compute model evidence (given that it is an average likelihood, averaging over all the parameter settings, and we don't what to/know how to compute this average exactly).

4) Finally, some of the studies that I wanted to talk about that illustrate the cool things you can do with these models are in slides 25-27. The novelty bonus one is one I especially like. I recommend the paper, and am happy to answer questions about it if you have any.

Best,

--Yael