CS 194-129 Deep Neural Networks
Spring 2018 Canny Lecture 22

1 Lecture

Recap

Advantage functions address the *temporal credit assignment problem*; using advantages focuses updates on actions with high advantage, while ignoring others. (If the advantage is close to zero, an action presumably doesn't affect the reward. This is probably true for most states.)

Since the advantage is zero in most places, it gives us a much sharper gradient signal during training. Gradients only backpropagate to actions with nonzero advantage, meaning only actions which affect the reward are reinforced. This should greatly speed up training.

Exploration v Exploitation

- *Exploitation*: perform the best action we're aware of.
- *Exploration*: try something new (might be better, might be worse).

Exploration Methods

A lot of RL algorithms, if left to their own devices, can become locked into a single set of trajectories. For example, the formula for Q-learning suggests that we should take the "best" action at every step (despite not knowing what the true best action is). We need to force these algorithms to explore, to get exposure to possibly better options.

Optimistic Exploration

Optimistic exploration tries to estimate the range of possible Q-values or rewards we might have in a state. It's optimistic in the sense that it favors states where the upper bound is higher.

If a state could be good, assume it's good.

If doing this, we need to estimate the state visitation frequency. The more frequently we visit a state, the more accurately we know the value. Thus, we add our uncertainty as an extra term to the reward, so that we estimate the quality of the best possible likely value for a state.

Thompson sampling algorithms: we can alternatively maintain a posterior over possible models, by learning a distribution over Q-functions or policies (training an ensemble of them and then sampling from that ensemble). We can then use a sample from our distribution to act.

Optimistic algorithms typically involve UCBs (upper confidence bounds): "how good can this value be?" In this bound, we add a bonus for visiting rarely-visited states.

$$r^+(s,a) = r(s,a) + \mathcal{B}(N(s))$$

where $N(s)$ is the number of times we've visited state s. We can easily add this to any model-free algorithm, just by replacing $r(s,a)$ with $r^+(s,a)$.

A big problem: states are often too fine-grained (almost continuous; think images), meaning we'd rarely see the same states twice and our per-state counts are kind of useless.

> But some states are more *similar* than others. Thus, we can estimate the *probability density* $p_\theta(s)$ of a state instead, where $p_\theta(s)$ might be high for a new s if similar to previously-seen states. We can then use $p_\theta(s)$ to get a pseudo-count.

Posterior Sampling

Posterior sampling explicitly models uncertainty in the model parameters θ, and samples from that distribution when taking an action (using Thompson sampling, e.g. sampling from an ensemble of models). It tends to produce policies that have the most variation in the parts of the state space where there's the most uncertainty in what we should do.

To more reliably represent the diversity in the models we might have across different training datasets, we should train the models on different bootstrap samples.

Bootstrap sample: a sample with replacement of size N (if we have a dataset of N items).

- Since it's the same size, many statistical estimators constructed on the re-sample are unbiased or have the same properties as the original sample.
- Despite this, it is a different sample (almost as if we were able to get a different sample of the original unlimited dataset). So when we train different models on bootstrap samples, we generally get the diversity of model parameters we would get if we were actually able to train on different samples.

Note: when doing posterior sampling, we should commit to a randomized but internally consistent strategy for an entire episode. In other words, we should sample the model parameters and then run for an episode (until the game resets itself). Otherwise our actions will be very messy and inefficient.

Model-Free v Model-Based Methods

- *Model-free*: no explicit model of what the environment does, e.g. no knowledge of the transition dynamics $p(s'|s, a)$.
- *Model-based*: not model-free. Often we do know the dynamics (e.g. in games like Go, easily modeled systems, or simulated environments) or can learn them.
  - *System identification*: when we know the equations of the dynamics, but not the values. All we have to do is fit the parameters of a known model.
  - *Learning*: fit a general-purpose model to observed transition data.

In model-based RL, we try to learn the transition dynamics (and then we figure out how to choose actions). This allows us to skip potentially expensive simulated or real-world actions. If we have a differentiable approximator like most neural networks, we can also differentiate all the way through the state transition function (and effectively the problem becomes one of label-based regression).

A reasonable simplification: projecting the state into a lower-dimensional representation. Now, instead of predicting the state s, we can predict $\phi(s)$ for ϕ some feature mapping. $\phi(s)$ should be sufficient to predict the next state and for the policy to make an action choice.

- If low-dimensional, this should be vastly simpler than working with the original state.

Curiosity-Driven Exploration by Self-Supervised Prediction

Here, we indeed compute a simplified featurization of the environment $\phi(s_t)$ (from which we can compute predictions of the next step). Then we use error in the prediction from the model to select states that need more exploration.

- Models will have high errors in states they haven't seen before.
- And if there's high error in the embedding map ϕ, the policy is likely to make bad choices of actions.
- So we should favor exploring states that have some impact on what the policy does.

Intrinsic Curiosity Model (ICM):

1. Use an inverse model $\phi(s_t), \phi(s_{t+1}) \to \hat{a}_t$ to ensure $\phi(s_t)$ is sufficient for action selection.
   - (Invoke a regression error $\|\hat{a}_t - a_t\|$ to ensure that
     ϕ allows us to predict the action we took to get from s_t to s_{t+1}.)
2. Estimate a forward model for the next state $\phi(s_t), a_t \to \hat{\phi}(s_{t+1})$.
3. Use forward model error $\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|$ as the curiosity signal r_t^i.

This will provide an intrinsic reward r_t^i to states where there's a divergence between the predicted next state and the actual next state (places where the forward prediction is bad). These are states we should visit some more so that the forward model does a better job; the forward model should continue to improve.

The reward used to train the policy consists of $(r_t^e + r_t^i)$, i.e. the *normal environment reward* plus the *intrinsic curiosity bonus*. We can train a policy via any RL method using this combined reward.

ICM tends to have an increasingly large benefit as the sparseness of rewards increases.

Imagination

If we can do a good job of using some kind of featurization ϕ of the state space (as opposed to using the original state space), well, why not always use the features? Can we come up with universal features that are much simpler and more useful than the raw data?

Let's look at people. When people visually imagine things, do they imagine at the pixel level s or do they have some pretty good $\phi(s)$ that they really use? The answer: nobody really knows. All we know is that the occipital (visual) and temporal (planning) lobes are involved. The primary visual cortex V1 (*the equivalent of the pixel level*) is sometimes involved (e.g. to see detail).

DeepMind has recently published a paper involving an imagination architecture.

- It has an imagined future simulation: an imagination rollout block gets the current observation and an action produced by a policy, and computes a sequence of observations and rewards.
- Then the rollout is encoded (as "advice") and passed (along with an embedding of the real observation) to another network, which is combined to generate value function and policy predictions.