POLICY GRADIENT METHODS, CURVATURE, AND DISTRIBUTION SHIFT

SHAM KAKADE

These are notes from a talk given by Dr. Sham Kakade on November 2nd on some rather high-level concepts in reinforcement learning. The talk can be accessed **here**.

In this write-up, I'll provide a very brief conceptual overview of some ideas that Dr. Kakade deals with before diving into his talk.

1. What is reinforcement learning?

Reinforcement learning is an area of machine learning that is concerned with the behaviour of agents who are concerned with maximizing utility. In a sense, it is paradigmatically divergent from **supervised** & **unsupervised** learning, which focus upon recognizing patterns within a given domain.

Reinforcement learning is studied by scholars schooled in a multitude of different fields. Game theorists, control theorists, researchers in operations research, and statisticians, among others, study reinforcement learning.

Basic reinforcement is modeled as a Markov decision process (MDP), consisting of four elements:

- A set of environment and agent states, $\mathcal{S}$. Can be either finite or infinite. We assume it's finite or countably infinite.
- A set of actions $\mathcal{A}$ of an agent. Can be either finite or finite, but we typically assume it's finite.
- A transition function $P : \mathcal{S} \times \mathcal{A} \to \delta(\mathcal{S})$, where $\delta(\mathcal{S})$ is the space of probability distributions over $\mathcal{S}$. $P\left(s' \,\middle|\, s, a \right)$. In other words, it's the *probability simplex*.
- $P_a(s, s') = \Pr\left(\mathcal{S}_{t+1} = s' \,\middle|\, \mathcal{S}_t = s, a_t = a\right)$
- A feward function $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$. This is the agent's reward for taking action a at state s.
- A discount factor $\gamma \in [0, 1)$, which defines a horizon for the problem. Intuitively, γ considers the tiem-scale of the pay-off (are we rewarding delayed gratification? In which case, γ will be large. Are we rewarding immediate gratification? In which case, γ will be small.)
- An initial state distribution $\mu \in \delta(\mathcal{S})$, which specifies how the initial state s_0 is generated.

A Markov decision process occurs over discrete steps of time t with can be characterized with a state s_t and an agent's action a_t. The goal of the agent is to learn a policy $\pi : A \times S \to [0, 1]$, $\pi(a, s) = \Pr\left(a_t = a \,\middle|\, s_t = s \right)$

Date: November 2, 2020.

2. What are policy methods?

In a given MDP $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \mu)$, the agent interacts with the environment according to the following protocol: the agent starts at s_0 μ & at each time step $t = 0, 1, 2, \ldots$, the agent takes an action $a_t \in \mathcal{A}$, obtains the immediate reward $r_t(s_t, a_t)$, and observes the next state s_{t+1} sampled according to s_{t+1} $P\left(\cdot \,\middle|\, s_t, a_t \right)$

The interaction record τ at time t,

$$\tau_t = (s_0, a_0, r_1 \ldots s_t)5$$

is called the trajectory at time t and includes the state at time t, s_t.

A policy specifies a decision-making strategy in which the agent is informed by the history of their observations (in this sense, a Markov decision process should be considered conceptually different from a Markov chain in terms of agnosticity to history).

A policy is a (possibly randomized) mapping from a trajectory to an action. Thus,

$$\pi : \mathcal{H} \to \delta(\mathcal{A})$$

, where $\mathcal{H}$ is the set of all possible trajectories (of all lengths) and $\delta\mathcal{A}$ is the space of all probability distributions over $\mathcal{A}$. A stationary policy $\pi : \mathcal{S} \to \mathcal{A}$ specifies a decision making strategy based solely on the current state (that is $a_t \sim \pi\left(\cdot \,\middle|\, s_t \right)$). A determinstic stationary policy is of the form $\pi : \mathcal{S} \to \mathcal{A}$

Now, we define **values** for general policies. For a fixed policy and a starting state $s_{0=s}$, we define the value function $V_m^\pi : \mathcal{S} \to \mathbb{R}$ as the discounted sum of future rewards

$$V_m^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \left(\gamma^t r(s_t, a_t) \,\middle|\, \pi, s_0 = s \right) \right] \tag{1}$$

Where the expectation is computed with respect to the randomness of the trajectory, that is, the randomness in state-transitions and stochasticity of π. Since $r(s, a)$ is bounded between 0 & 1, we have $0 \leq V_m^\pi(s) \leq 1/(1-\gamma)$

Similarly, the action-value (or Q-value) function $Q_M^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as

$$Q_m^\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \left(r(s_t, a_t) \,\middle|\, \pi, s_0 = s, a_0 = a \right) \right] \tag{2}$$

, which is also bounded from below by 0 and above by $1/(1-\gamma)$

The optimization that the agent aims to solve is to maximize V_M^π by finding a policy π. The agent thus wants to maximize

$$\max_\pi V_M^\pi(s) \tag{3}$$

3. What was Dr. Kakade talking about?

Definition 1 (The state space & state variables & gradients)**.** *A* ***state variable*** *is one of the set of variables that are used to describe the "state" of a dynamical system – it describes enough of a system to determine its future behaviour in the absence of any external forces affecting the system.*

*The **state space** is the Euclidean space in which the variables on the axes are the state variables. It is a mathematical model of a physical system as a set of input, output, and state varibles related by first-order differential equations or difference equations.*

*The **gradient** $\nabla f(p)$ of a scalar-valued differential equation f of several variables $(x_1, x_2, \ldots, x_n)$ is given as:*

$$\nabla f(p) = \begin{bmatrix} \frac{\delta f}{\delta x_1}(p) \\ \vdots \\ \frac{\delta f}{\delta x_n}(p) \end{bmatrix} \tag{4}$$

The tabular dynamic programming approach is commonly taken to describe a reinforcement learning problem:

State s	Action a	State-action value $Q^\pi(s, a)$
...	...	...
⋮	⋮	⋮
...	...	...

The third column consists of a policy and gives the reward. It's a one-step look-ahead (that is, it gives the reward for the next step after). If we could magically compute it for each row, we could update our policy to be greedy and it will converge quickly. Unfortunately, the table is very large.

The RL approach is to generalize; instead of querying the entire table, we can take a sampling approach in order to find an optimal policy. Policy gradient methods are one such approach, which easily deal with large state/action spaces. The policy is directly parameterized (for instance, through a neural network) with the gradient easily computed with simulation-based methods. You don't need to know the model. In ML, it's common to directly optimize the quantity of interest.

Why this is possible is an open-question: the gradients in supervised learning (SL) and reinforcement learning are very different. We don't care about non-convexity much in SL, which isn't the case in RL. In RL, the regions can have exponentially small gradients, with much higher-order regions being flat.

In this talk we look at global convergence of RL methods and provide generalization guarantees of nonconvex policy gradient methods.

We will start with a simple test case of small state spaces and exact gradients before moving to large state spaces.

Definition 2. *A non-convex space is a space which isn't convex. That is, if for all points $x, y \in \mathbb{R}$, and any $\lambda \in [0, 1]$, it is true that $\lambda x + (1 - \lambda)y \in \mathbb{R}$, then $\mathbb{R}$ is convex.*

An iterative method is called **locally convergent** if successive approximations are guaranteed to approach when the initial approximation is sufficiently close to the solution. An iterative method that converges for an arbitrary initial approximation is called **globally convergent**.

This talk provides global convergence & generalization guarantees of non-

They start with small state spaces with exact gradients before moving to large state spaces. Dealing with large state spaces means we need to deal with generalization (obviously) and distribution shift.

4. Small state spaces & the "softmax" policy class

The softmax policy is the simplest way to parameterize the simplex

What's the simplex?

Definition 3. *A simplex is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions. In the context of machine-learning, we can consider the feasible region (that is, the set of possible strategies) to be a simplex. Navigating the simplex, then, means finding the optimal solution.*

A policy class is, essentially, a set of policies.

Softmax policies are of the form

$$\pi_\theta\theta\big(\ a \ \big|\ s\ \big) = \frac{\exp(\theta_s, a)}{\sum_{a'} \exp(\theta_s, a')} \tag{5}$$

The complete class contains every stationary policy (and a stationary policy is a policy that does not change over time.).

The policy optimization problem $\max_\theta V^{\pi_\theta}(s_0)$ is non-convex. If we can't find an optimal policy class with this simple problem, how can we find them for a more complicated problem?

We're going to avoid sampling and assume we can get the exact gradients (that is, $\nabla V^{\pi_0}(s_0)$ is given exactly.).

Before we do that, we need to define the following:

Definition 4 (Gradient Descent)**.** *Gradient descent is a first-order (that is, it has at most linear error) iterative optimization algorithm for finding the local minumum of a differentiable function.*

The softmax policy is the simplest way to parameterize the simplex (that is, it converts the one-step look-ahead values of a state & action pair into action probabilities). It characterizes the function as a logistic function. However, the rate could be exponentially slow in terms of state and the softmax can have very flat gradients (meaning if you go to some corner of the simplex, the gradient could be very slow, so moving away from the corner can take a very long time.). This means that finding the optimal policy could be exponentially slow in the number of states

What if we try some sort of regularization-penalty to bias ourselves from getting these flat gradients (which, for the record, is very possible if the policy is deterministic)? The simplest way to do this is to use log-barrier regularization.

Instead of doing gradient descent on the spread-out measure, we can add a penalty that pushes us away from determinsitc policies. The average-log probability stops us from forming a policy that's too deterministic. This leads us to optimize the following:

$$L_\lambda(\theta) = V^\theta(\mu) + \frac{\lambda}{SA}\sum_{s,a}\log_{\pi_\theta}\left(a \,\middle|\, s \right) \tag{6}$$

$$\theta \leftarrow \theta + \eta\nabla L_\lambda(\theta) \tag{7}$$

Then, we get the following:

Theorem 4.1 (Global convergence: softmax + log barrier regularization)**.** *S: # states, A:# actions, H: Horizon = 1/(1-γ), where γ is a function of ε*

Suppose μ = uniform$_s$, and with appropriate settings of λ&η, after $\frac{S^4A^2H^6}{\varepsilon^2}$ iterations we have, for all s,

$$V^0(s) \geq V^*(s) - \varepsilon \tag{8}$$

To put it succinctly: they found that the policy gradient converges in polynomial time, which is pretty quick for this simple case, as it's non-convex.

What's going on? The log-barrier plus the uniform distribution helps with conditioning problems. The log barrier regularization is the same as KL-regularization (different from entropy regularization) which allows for better convergence to a deterministic strategy.

Another approach is to warp where we move along the simplex by preconditioning. Can we stretch the geometry out so we can move quickly when the gradient becomes flat? We can do this with the Fisher information matrix (which measures the amount that an observable random variable X tells us about an unknown parameter θ of a distribution that models X). For every state we have a Fisher matrix. We can take the expected Fisher metric over the states we tend to visit under the current distribution. At this point, it's the Fisher matrix of the trajectories we follow.

When we run gradient descent on this preconditioner, the update rule looks like a "soft" policy iteration update rule, where the probability of an action is relative to the exponent of Q. So what's happening with this non-convex update rule? There's no dependence on the starting distribution because of the preconditioning, so there's a source of mystery.

Looking at the convergence rate, after the learning rate is appropriately set, for all states we get that we converge to the optimal state at the rate of 1/T (where T is the number of iterations). This has no dependence on the number of states, A, or the starting measure μ. This is pretty fast (we usually have $1/\sqrt{T}$).

We have some tools to think about these flat gradients. We understand the curvature issue. With exact gradients, a tabular approach converges, and with a preconditioner we can converge very quickly without dependence on S and A.

But there are a few more questions:

(1) How do we think about approximation/generalization (which isn't an issue in supervised learning)?

(2) How do we think about μ in an infinite state space?

5. Large state spaces

We'll use what we know about the preconditioned case, where we found really fast convergence without dependance on $S\&A$.

If we want to apply our ideas to more complex problems than maze searching (which is characterized by a small state space) such as robotics, healthcare data, interacting with the world, games, etc.

A question relevant to my research: can we consider strategies in vertex-pursuit games as Markov Decision Processes? What does that say about the state-space of possible strategies? Might variations of games differ?

These more complex problems require us to think about approximation and generalization.

We started with a soft-max policy class Most policy classes we use are of the form:

$$\pi_\theta(a \mid s) \propto \exp(f_\theta(s,a)) \tag{9}$$

Meaning, they're the log of some function ($e^{\ln(f_\theta)}$). It is just a tabular paramaterization. He wrote this:

$$f_\theta(s,a) = \theta_{s,a} \tag{10}$$

A log-linear parameterization has some features:

$$f_\theta(s,a) = \overrightarrow{\theta} \cdot \overrightarrow{\phi}(s,a) \text{ where } \overrightarrow{\phi} \in \mathbb{R}^d \tag{11}$$

Another natural policy is a neural policy class where $f_\theta(s,a)$ is a neural network.

We can make f log-linear. It may not even contain the optimal policy. We have a feature mapping, where the feature vector is $\phi(s,a) \in \mathbb{R}^d$ & $\pi_\theta(a \mid s) \propto \exp(\theta, \phi_{s,a})$. We want the dimension d to be smaller than the number of states and actions.

At iteration t we use the natural policy-gradient (NPG) update rule:

$$\theta \leftarrow \eta F(\theta)^{-1} \nabla V^\theta(s_0) \tag{12}$$

Which is actually equivalent to the "soft"-approximate policy iteration update:

1. Approximate Q^θ with the features:

$$w_\star \in \arg\min_w E_{s,a\sim d(\cdot|\pi,\mu)} \left[(Q^\theta(s,a) - w \cdot \phi_{s,a})^2 \right] \tag{13}$$

where $d(\cdot \mid \pi, \mu)$ is the "on-policy" distribution starting from $s_0, a_0 \sim \mu$. μ is an on-policy coverage. We can solve the above equation with regression.

2.

$$\pi(a \mid s) \leftarrow \frac{\pi(a \mid s) \exp(w_\star \cdot \phi_{s,a}}{Z_s} \tag{14}$$

where Z_s

which is a sample-based approach, which is interesting since it's saying that a gradient-update rule looks like regression. It works since in the NPG you're solving a linear system with a preconditioner, which looks like the update rule in 2. You're finding the best fit Q^θ with your features and then you'll update your policies by being $e^{\hat{Q}}$, since you can think of $\exp(w_\star, \phi_{s,a})$ as being the best approximation to Q with the features you've got.

It's a clean update rule and a soft-way of doing the policy gradient. If we made η infinity we'd get a greedy update rule, but the soft approach is far more incremental, which is good since it's more like hill-climbing.

Instead of exact updates we can use samples, meaning we use samples for our update rule and for Q. This is like putting supervised learning into our problem, since computing $\hat{Q}$ is just regression. If we're generalizing in some sense, can we do RL well? Can we handle the curse of dimensionality?

It's important to talk about the natural policy-gradient, since that's what gives us the dimension-free convergence that we like.

References

[1] A. Agarwal, N. Jiang, S.M. Kakade, W. Sun; Reinforcement Learning: Theory and Algorithms; accessed October 27, 2020; available at `https://rltheorybook.github.io/`