Reinforcement Learning

Hasan Poonawala

Contents

1 Introduction

1.1 MDP Trajectories

Agent chooses actions so as to maximize expected cumulative reward over a time horizon. Observations can be vectors or other structures. Actions can be multi-dimensional. Rewards are scalar but can be arbitrarily uninformative. Agent has partial knowledge about its environment.
The value of a policy π started in state i is

$$V^{\pi}(i) = E_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = i\right] \tag{1}$$

Optimal policy $\pi^* = \arg\max_{\pi} V^{\pi}$
Optimal value $V^*(i) = \max_{\pi} V^{\pi}(i)$
In MDPs there always exists a deterministic stationary policy (that simultaneously maximizes the value of every state).
Due to the Markov assumption:

$$\forall s \in S, V^{\pi}(s) = R(s, \pi(s)) + \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s') \tag{2}$$

$$\forall s \in S, \forall a \in A, Q^{\pi}(s, a) = R(s, a) + \sum_{s' \in S} P(s'|s, \pi(s)) Q^{\pi}(s', \pi(s')) \tag{3}$$

Optimality:

$$\forall s \in S, V^*(s) = \max_{a \in A}\left[R(s, a) + \sum_{s' \in S} P(s'|s, a) V^*(s')\right] \tag{4}$$

$$\forall s \in S, \forall a \in A, Q^*(s, a) = R(s, a) + \sum_{s' \in S} P(s'|s, a) \max_{b \in A} Q^*(s', b) \tag{5}$$

1.2 Valuation Functions

The different functions are

- Q-function $Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}\left[r(s_{t'}, a_{t'}) | s_t, a_t\right]$
- Value $V^{\pi}(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)}\left[Q^{\pi}(s_t, a_t)\right]$
- Advantage $A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$
- Fitted value $V^{\pi}_{\phi}(s_t)$

1.3 Solving MDPs

Given an exact model (i.e., reward function, transition probabilities), and a fixed policy π, we can use Value Iteration (Policy Evaluation).
The proof that value iteration works is based on showing that the update map – inherent to the value iteration algorithm – is contracting.
Later on, this contraction property allows proof of the convergence of Q-Learning, which is a stochastic approximation of Value Iteration for Q (as opposed to that for V).
We use the notation from link
We take a sequence of actions $\{\mathcal{A}_t\}$ and assign the expected reward J due to this sequence when starting in state x:

$$J(x, \{\mathcal{A}_t\}) = \mathbb{E}\left[\sum_{t}^{\infty} \gamma^t R(X_t, A_t) | X_0 = x\right] \tag{6}$$

A policy $A_t = \pi(X_t)$ induces a value for each state $V^\pi(X_t)$ given by

$$V^\pi(x) = \mathbb{E}\left[\sum_t^\infty \gamma^t R\left(X_t, \pi(X_t)\right) | X_0 = x\right] \tag{7}$$

By picking π, we decide the topology of the MDP. The value V^π defines a different topology. This mismatch is what drives learning.
The best value forms the optimal value function V^*.

$$V^*(x) = \max_{\mathcal{A}_t} J(x, \{\mathcal{A}_t\}) \tag{8}$$

The Markov property allows us to combine (6) and (8) as

$$V^*(x) = \max_{a\in\mathcal{A}} \sum_{y\in S} P(y|x,a)[r(x,a,y) + \gamma V^*(y)] \tag{9}$$

which makes

$$Q^*(x,a) = \sum_{y\in S} P(y|x,a)[r(x,a,y) + \gamma V^*(y)] \tag{10}$$

But since $V^*(x) = \max_{a\in A} Q^*(x,a)$, we have

$$Q^*(x,a) = \sum_{y\in S} P(y|x,a)[r(x,a,y) + \gamma \max_{b\in A} Q^*(y,b)] \tag{11}$$

We want to find the function Q^* that satisfies (11). We use fixed point iterations to do so. If we have an equation $z^* = H(z^*)$, then z^* is a fixed point of the map H. We can iterate by $z_{k+1} = Hz_k$. Will $\lim_{k\to\infty} z_{k+1} = z^*$?
One case where this happens is when H is a contractive map. Start with two points x and y in a space X, and we map both points by H to get $H(x)$ and $H(y)$. H is contractive if

$$norm(H(x) - H(y)) < norm(x - y) \tag{12}$$

If H is contractive the iterates from any z_0 will uniquely converge to such a point.
So, for Q-Value Iterations, the space X is a vector of size $|S|\times|A|$, map H and norm turn out to be the RHS of (11) and the infinity-norm respectively.

1.4 The Reinforcement Learning Problem

Reinforcement learning is concerned with finding policies for MDPs wherein the reward function and the transition probabilities are unknown.
Without a model, have to observe real traces. An agent sees trajectories of the form

$$s_1 a_1 r_1 s_2 a_2 r_2 \ldots s_i a_i r_{i+1} s_{i+1} \ldots$$

where the term between ellipses is a unit of 'experience'. It tells us the two things we need to know about executing a in s: resulting reward and transition.
Features of RL:

- Temporal Differences (or updating a guess on the basis of another guess)
- Eligibility traces
- Off-policy learning
- Function approximation for RL
- Hierarchical RL (options)
- Going beyond MDPs/POMDPs towards AI

Note: Learning from delayed reward distinguishes RL from other ML.
The objective function most often used in RL is

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} [r(\tau)] \tag{13}$$

where τ is a trajectory and $r(\tau)$ is the reward of this trajectory. Typically, $r(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$. We approximate $J(\theta)$ from N policy rollouts as

$$J(\theta) = \frac{1}{N} \sum_i \sum_t r(s_{it}, a_{it}) \tag{14}$$

The parameters θ define the policy $\pi_\theta(a_t|s_t)$. The model and this policy together dictate $p_\theta(\tau)$ as

$$p_\theta(\tau) = p(s_1) \Pi_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{15}$$

The reinforcement learning goal is to estimate

$$\theta^* = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(s_t, a_t) \right] \tag{16}$$

We modify this by using the state-action marginal $p(s_t, a_t)$ as

$$\theta^* = \arg\max_\theta \sum_{t=1}^{T} E_{(s_t, a_t) \sim p(s_t, a_t)} [r(s_t, a_t)] \tag{17}$$

For the infinite horizon case we will obtain which depends on the stationary distribution $p(s, a)$.

$$\theta^* = \arg\max_\theta E_{(s,a) \sim p(s,a)} [r(s, a)] \tag{18}$$

Note: The policy induces a local topology between states, the value function corresponding to that policy dictates a global topology. Is this mismatch what drives learning? Sort of.
Broadly, there are two ways to find θ:

- Gradients on $J(\theta)$
- Dynamic programming on (s, a)

1.5 Classification of RL Algorithms

There are three broad classes of RL problems:

- Episodic
- Discounted/Infinite horizon
- Ergodic

There are two types of learning methods

- Off-policy: where policy is computed from previously collected data. Eg. Q-Learning
- On-policy: where on-line data is collected. Eg. SARSA

Two approaches: Direct and Indirect methods.

- Indirect methods learn model from experience and solve it as known MDP.
  Indirect model: frequentist estimate of transition probabilities. Model converges asymptotically provided all state-action pairs are visited infinitely often in the limit; hence certainty equivalent policy converges asymptotically to the optimal policy.
- Direct methods don't bother with a model.
  Direct method: Q-learning.

$$\theta^* = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} r(s_t, a_t)\right] \tag{19}$$

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else

Value functions

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}|s_t, a_t)] \tag{20}$$

$$\begin{aligned} V^\pi(s_t) &= \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t)] \\ &= E_{a_t\sim\pi(a_t|s_t)}[Q^\pi(s_t, a_t)] \end{aligned} \tag{21}$$

Then, $J(\theta) = E_{s_1\sim p(s_1)}[V^\pi(s_1)]$.

Idea 1: If we have policy π, and know $Q^\pi(s, a)$, we can improve π.
Set $\pi^{new}(a_i|s) = 1$ if $i = \arg\max Q^\pi(s, a_i)$.
SoftQL says $\pi^{new}(a|s) \propto \exp Q^\pi(s, a_i)$

Idea 2: Compute gradients to increase probability of good actions a.
If $Q^\pi(s, a) > V^\pi(s)$, then a is better than average.
Modify $\pi(a|s)$ to increase probability of a when this situation occurs.

Model-Based RL

1. Just use the model to plan (no policy)
   - Trajectory optimization/optimal control (primarily in continuous spaces) – essentially backpropagation to optimize over actions
   - Discrete planning in discrete action spaces – e.g., Monte Carlo tree search
2. Backpropagate gradients into the policy
   - Requires some tricks to make it work
3. Use the model to learn a value function
   - Dynamic programming
   - Generate simulated experience for model-free learner

Comparison Comparison: stability and ease of use

- Value function fitting
  - At best, minimizes error of fit ("Bellman error")
    Not the same as expected reward
  - At worst, doesn't optimize anything
    Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case

- Model-based RL
  - Model minimizes error of fit
    This will converge
  - No guarantee that better model = better policy
- Policy gradient
  - The only one that actually performs gradient descent (ascent) on the true objective

2 Dynamic Programming

Applies to known transition models.

2.1 Policy Evaluation

Policy evaluation computes the value functions for a policy π using the Bellman equations. For example,

$$\text{Repeat: } V^\pi(s) \leftarrow \mathbb{E}_{a\sim\pi(a|s)}\left[E_{s'\sim p(s'|s,a)}\left[r(s,a)+\gamma V^\pi(s')\right]\right] \tag{22}$$

$$\text{Or: } V^\pi(s) \leftarrow \sum_a \pi(a|s)\sum_{s'} p(s'|s,a)\left[r(s,a,s')+\gamma V^\pi(s')\right] \tag{23}$$

2.2 Policy Improvement

For all $s \in S$, if for policies π' and π we have that $Q^\pi(s,\pi'(s)) \geq V^\pi(s)$, then π' cannot be worse than π. This relative improvement is captured by the advantage function, and the policy improvement step is trying to maximize the value by changing the policy to the maximum of the advantage of each state.

2.2.1 Policy Gradients

Instead of a global improvement step, for parametrized policy, we can use a local gradient update. This gradient can be practically calculated through the policy gradient theorem. Recall that

$$\begin{aligned}
J(\theta) &= E_{\tau\sim p_\theta(\tau)}\left[r(\tau)\right] = \int p_\theta(\tau)r(\tau)d\tau, \text{ where} \\
r(\tau) &= \sum_{t=1}^{T} r(s_t,a_t), \text{ and} \\
p_\theta(\tau) &= p(s_1)\Pi_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t) \\
\implies \log p_\theta(\tau) &= \log p(s_1) + \sum_{t=1}^{T}\log\pi_\theta(a_t|s_t)
\end{aligned}$$

Use identity: $\pi_\theta(\tau)\nabla_\theta \log\pi_\theta(\tau) = \nabla_\theta\pi_\theta(\tau)$. We get

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int \nabla_\theta p_\theta(\tau)r(\tau)d\tau \\
&= \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)r(\tau)d\tau \\
&= E_{p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)r(\tau)] \\
&= E_{p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_{it}|s_{it})\right)r(\tau)\right]
\end{aligned} \tag{24}$$

For N trajectories, $\nabla_\theta J(\theta)$ is

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_{it}|s_{it})\right)\left(\sum_{t=1}^{T}r(s_{it},a_{it})\right) \tag{25}$$

2.3 Policy Iteration

1. Evaluate V^π as $V^\pi(s) \leftarrow \mathbb{E}_{s'\sim\pi}\left[r(s,\pi(s),s')+\gamma V^\pi(s')\right]$
2. Improve π as $\pi^{new}(a_i|s) = 1$ if $i = \arg\max A^\pi(s,a_i)$.

Alternative evaluation: $A^\pi(s,a) = r(s,a) + \gamma\mathbb{E}\left[V^\pi(s')\right] - V^\pi(s)$

2.4 Value Iteration

1. Set $Q(s,a) \leftarrow \mathbb{E}_{s' \sim \pi}\left[r(s,a,s') + \gamma V(s')\right]$
2. Set $V(s) = \max_a Q(s,a)$. (No max in PI)

Which is equivalent to a combining policy evaluation (expectation) and improvement (max) into a single step:

$$V_{k+1}(s) \leftarrow= \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_k(s')\right) \tag{26}$$

Policy comes from $\arg\max_a Q(s,a)$

2.5 PI vs VI

In policy iteration, you represent value V^π and policy π. Then, estimate V^π and use it to improve the policy greedily.
In value iteration, you only represent the value $V(s)$, which implicitly defines a policy. The evaluation and improvement occurs in a single step using VI. One way to describe it is that it's a single-time-step policy iteration.
In Sutton's description, PI can be efficient because the value doesn't change much, speeding up policy evaluation. But also, PI is inefficient because of protracted policy evaluation iterations. This inefficiency motivates VI.

2.6 Generalized policy iteration

Value and policy functions interact until they are optimal and thus consistent with other.

3 Monte Carlo Methods

Applies to unknown transition models, but complete episodic experience.

3.1 Policy evaluation using Monte Carlo Rollouts

Monte Carlo plays out the whole episode until the end to calculate the total rewards. Each state may be visited many times either in the same episode, or over multiple episodes. Due to the Markov property, every visit provides an estimate of the return. So MC policy evaluation uses the average return over rollouts to update the value of each state.
Suppose you want to find $V^\pi(s)$ for some fixed state s
Start at state s and execute the policy for a long trajectory and compute the empirical discounted return
Do this several times and average the returns across trajectories
How many trajectories?
Unbiased estimate whose variance improves with n
Use generative model to generate depth 'n' tree with 'm' samples for each action in each state generated [Kearns, Mansour & Ng]
Near-optimal action at root state in time independent of the size of state space (but, exponential in horizon!)

3.2 REINFORCE

The REINFORCE algorithm uses Monte Carlo simulation to evaluate returns, and policy gradient to improve the policy.

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{it}|s_{it})\right)(G)\,, \text{ where} \tag{27}$$

$$G = \sum_{t=1}^T r(s_{it}, a_{it}) \tag{28}$$

The usual way some depict this update is

$$\theta_{t+1} \leftarrow \theta_t + \alpha G \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{29}$$

One issue with REINFORCE is that it is non-causal, because the policy at later states seem to influence the rewards at earlier states. We introduce causality by redefining the sum of rewards. In effect, we get

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{it}|s_{it}) \left(\sum_{j=t}^T r(s_{ij}, a_{ij})\right) \tag{30}$$

$$= \frac{1}{N}\sum_i^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{it}|s_{it})\,(G_{it})\,, \text{ where} \tag{31}$$

$$G_{it} = \sum_{j=t}^T r(s_{ij}, a_{ij}) \tag{32}$$

or

$$\theta_{t+1} \leftarrow \theta_t + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{33}$$

Gaussian policy. For example, if $\pi_\theta(a_t|s_t) = \mathcal{N}(f(s_t), \Sigma)$, we get

$$\log \pi_\theta(a_t|s_t) = -\frac{1}{2}\|f(s_t) - a_t\|_\Sigma^2 + const \tag{34}$$

$$\nabla_\theta \log \pi_\theta(a_t|s_t) = -\Sigma^{-1}\left(f(s_t) - a_t\right)\frac{\partial f}{\partial \theta} \tag{35}$$

3.2.1 REINFORCE With Baselines

A good baseline is essentially a value function: a prediction of expected rewards in a state s. We use this baseline to reduce the variance of REINFORCE, but we introduce bias. Note that despite learning a value function, the REINFORCE remains a Monte Carlo method. Essentially, there is no bootstrapping or critic-like behavior for this estimate. See: Medium article, Daniel Takeshi blog, David Silver Video, John Schulman Video.
The update is now

$$\theta_{t+1} \leftarrow \theta_t + \alpha(G_t - b_t(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t) \tag{36}$$

Possible baselines:

1. Whitened returns: use the average returns over the time steps, and then normalize the difference by the deviation of these returns. Note that this normalization breaks everything about how baselines were derived, but it seems to be widely used.
2. Self-critic with Sampled baselines: at each state in the trajectory, sample rollouts from there to get a baseline. This process is expensive since we increase the simulated experience by some factor.

3.3 Off-Policy Gradients With Importance Sampling

We can estimate the gradient update with respect to a different set of parameters θ compared to the parameters θ_{old} that generated the info. The trick is to use importance sampling.

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ E_{x\sim p(x)}[f(x)] &= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$

So, even if x is drawn from $q(x)$, we can estimate what would happen had samples been drawn from $p(x)$. If we have data generated from $\pi_{\theta_{\text{old}}}$, and we want to know the advantage relative to $V^{\pi_{\theta_{\text{old}}}}$ of using another policy π_θ that we haven't tried implemented yet:

$$E_{x\sim\pi_\theta(x)}\left[A^{\pi_{\theta_{\text{old}}}}\right] = E_{x\sim\pi_{\theta_{\text{old}}}(x)}\left[\frac{\pi_\theta(x)}{\pi_{\theta_{\text{old}}}(x)}A^{\pi_{\theta_{\text{old}}}}\right] \tag{37}$$

Note that we actually need to calculate expectations with respect to state distributions, but it turns out that the ratio of state distributions under different policies can be replaced by a product of the ratios of policy distributions. See this webpage for details.

3.3.1 Optimal Importance Sampling

(From Jonathan Hui's blog)
For the estimation to have the minimum variance, the sampling distribution q should be

$$q(x) \propto p(x)f(x).$$

Intuitively, it means if we want to reduce the variance of our estimation, we want to sample data points with higher rewards.

3.3.2 Normalized Importance Sampling

Using an empirical normalization factor trades of more bias for less variance.

4 Temporal Difference Learning

To improve the policy π, we need to evaluate it, by either computing Q^π, or it's policy-averaged value V^π. Monte Carlo methods wait until the end of an episode to then assign values to V^π or Q^π using returns. Temporal Difference methods try to learn from incomplete episodes, down to one-step bootstrap.

4.1 Bootstrapping

Suppose we're estimating $P(s)$ which is the expected sum of discounted rewards. We maintain estimates as $P_t(s)$ at time t for each state s. A simple one-step look-ahead update rule is to say $P_{t+1} \leftarrow P_t + \alpha(y_{t+1} - P_t)$ which pulls the current estimate towards the rewards y_{t+1} it is seeing, and horizon is one step. We can implement the one-step look-ahead immediately.
Instead of a one-step return y_{t+1}, we might update it with some quantity Y_t that is an expected sum or whatever over a longer period? We would have to wait for that time period to conclude before we could do anything.
The TD trick essentially says we can expand terms to see that $Y_t(s_t) = \gamma Y_{t+1}(s_{t+1}) + y_{t+1}(s_t)$, and then we replace $Y_{t+1}(s_{t+1})$ by $P_t(s_{t+1})$.

$$\begin{aligned} P_{t+1}(s_t) &\leftarrow P_t(s_t) + \alpha\left(y_{t+1} + \gamma Y_t - P_t(s_t)\right) \\ &= P_t(s_t) + \alpha\left(y_{t+1} + \gamma P_t(s_{t+1}) - P_t(s_t)\right) \end{aligned} \tag{38}$$

In effect, the target for the Monte Carlo update is $P_t(s)$, whereas the target for the TD update is a Bellman-inspired update like $y_{t+1} + \gamma P(s_{t+1})$.

4.2 Bootstrapping The Value Function: TD(0)

For each state, we update it as

$$V(s) \leftarrow V(s) + \alpha\left[r_t + \gamma V(s') - V(s)\right] \tag{39}$$

From Sutton's page:

> Under batch updating, TD(0) converges deterministically to a single answer independent of the step-size parameter, α, as long as α is chosen to be sufficiently small. The constant-α MC method also converges deterministically under the same conditions, but to a different answer. Understanding these two answers will help us understand the difference between the two methods. Under normal updating the methods do not move all the way to their respective batch answers, but in some sense they take steps in these directions. Before trying to understand the two answers in general, for all possible tasks, we first look at a few examples.
>
> Batch Monte Carlo methods always find the estimates that minimize mean-squared error on the training set, whereas batch TD(0) always finds the estimates that would be exactly correct for the maximum-likelihood model of the Markov process. In general, the maximum-likelihood estimate of a parameter is the parameter value whose probability of generating the data is greatest. In this case, the maximum-likelihood estimate is the model of the Markov process formed in the obvious way from the observed episodes: the estimated transition probability from i to j is the fraction of observed transitions from i that went to j, and the associated expected reward is the average of the rewards observed on those transitions. Given this model, we can compute the estimate of the value function that would be exactly correct if the model were exactly correct. This is called the certainty-equivalence estimate because it is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximated. In general, batch TD(0) converges to the certainty-equivalence estimate.

4.2.1 Fitted Value Iteration

1. Set $y_i \leftarrow \max_{a_i}\left(r(s_i, a_i) + \gamma E[V_\phi(s_i')]\right)$
2. Set $\phi \leftarrow \arg\min_\phi \sum_i \|V_\phi(s_i) - y_i\|^2$

4.3 Boot-Strapping the Q Function: SARSA and Q-Learning

The idea here is to pull the Q-value to something that would satisfy the Bellman equation. We do this by considering a unit of (possibly predicted) experience s, a, r, s', and considering an additional action a'. The predicted case involves predicting what r and s' would be, instead of experiencing what they would be after taking a in s.
Using this information, we would like to update our Q function:

$$Q^{new}(s,a) \leftarrow \underbrace{r(s,a)}_{\substack{\text{observed or from a} \\ \text{model}}} + \gamma \overbrace{Q}^{modeled} \underbrace{(s',a')}_{\substack{\text{What values} \\ \text{for } s',\, a' \,?}}$$

s will always be the result of taking a in s (experienced or predicted). To choose a', we may either do:

1. SARSA: $a' = \pi(s')$, which is on-policy evaluation.
2. Q-Learning: $a' = \arg\max_a Q(s', a)$, which is off-policy evaluation.

4.3.1 Q-Learning

Q-learning is stochastic approximation of (finite state and action space) Q-value iteration. Some also refer to the online-version of FQI as QL, however there are some implementation differences.
If Q-value iteration is $Q_{k+1} = T(Q_k)$, then stochastic approximation version (online) is $Q_{k+1} = (1-\alpha)Q_k + \alpha[T(Q_k) + \eta_k]$ were $\eta)k$ is zero-mean noise.
Q-Learning is the first provably convergent direct adaptive optimal control algorithm.

- Great impact on the field of modern Reinforcement Learning
- smaller representation than models
- automatically focuses attention to where it is needed, i.e., no sweeps through state space
- though does not solve the exploration versus exploitation dilemma
- epsilon-greedy, optimistic initialization, etc ...

However, Q-Learning requires visiting all state-action pairs, which makes the guaranteed properties applicable to finite state-action systems. Q-Learning is unstable and diverges in the continuous state/action case. With additional tricks, it sometimes works (DQN, RAINBOW, Double DQN).

4.3.2 Fitted Q-Iteration

If we don't have a model of the MDP transitions, can't get $E[V(s')]$ in any of the value-function-iteration steps above. By contrast, we can estimate the Q-function from samples,/ Moreover, $E[V(s'_i)] \approx \max_{a'_i} Q(s'_i, a'_i)$.
So, collect data, then iterate:

1. Set $y_i \leftarrow \max_{a_i} \left(r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)\right)$
2. Set $\phi \leftarrow \arg\min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

Pros and Cons:

+ works even for off-policy samples (unlike actor-critic)
+ only one network, no high-variance policy gradient
- no convergence guarantees for non-linear function approximation (more on this later)

Since the Q function defines a canonical policy, we essentially have a policy evaluation, policy improvement loop that forms the Fitted Q-Iteration.

Online Fitted Q-Iteration: FQI has an obvious online version, which avoids keeping a history of transitions. Note that the literature seems to not favor on-line Q-Learning. Even the discrete case is not guaranteed to work.

1. take some action a_i, observe (s_i, a_i, s'_i, r_i)
2. Set $y_i \leftarrow \max_{a_i} (r(s_i, a_i) + \gamma \max_a Q_\phi(s'_i, a))$
3. $\phi \leftarrow \phi - \alpha \frac{\partial Q_\phi}{\partial \phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$

Another way to describe Fitted Q-Iteration:

1. Sample M Transitions, add to buffer $\mathcal{B}$
2. Save Target Network Parameters $\phi' \leftarrow \phi$
3. Sample mini-batch (s_i, a_i, r_i, s'_i) from $\mathcal{B}$
4. Not-quite-gradient $\phi \leftarrow \phi + \alpha \sum_i \frac{\partial Q_\phi}{\partial \phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a')]\right)$

The idea is that we fit to convergence of ϕ before resampling.

4.3.3 Deep Q-Learning

DQN says: replay buffer, use a fixed $Q_{\phi'}$ over multiple updates to ϕ.

1. Save Target Network Parameters $\phi' \leftarrow \phi$
2. Sample M Transitions, add to buffer $\mathcal{B}$
3. Sample mini-batch (s_i, a_i, r_i, s'_i) from $\mathcal{B}$
4. Not-quite-gradient $\phi \leftarrow \phi + \alpha \sum_i \frac{\partial Q_\phi}{\partial \phi}(s_i, a_i) \left(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a')]\right)$

Unlike FQI, here we update ϕ' slowly, with new experience gained before convergence of the Q function regression.

4.4 Actor-Critic

Actor-critic: Generalized policy iteration where the **evaluation uses a TD value iteration** step (not MC) and the **improvement uses policy gradients**.
Instead of Monte Carlo returns, one can use some other estimate of return $r(\tau)$ starting from s_{it} at time t of trajectory i. One candidate is to use $r(\tau) \approx Q^\pi(s_{it}, a_{it})$, the action-value estimate to predict future returns when computing the gradient. This action-value function is the critic, and it is updated using bootstrapping (Target for x in state is reward in state plus x in successor state).

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_i^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{it}|s_{it}) \hat{Q}^\pi(s_{it}, a_{it}) \tag{40}$$

Consider a baseline, the average of total rewards

$$b = \frac{1}{N} \sum^N r(\tau) \tag{41}$$

It turns out that

$$E\left[\nabla_\theta \log \pi_\theta(\tau) b\right] = 0 \tag{42}$$

Therefore,

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_i^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{it}|s_{it}) A^\pi(s_{it}, a_{it}) \tag{43}$$

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t] \tag{44}$$

$$V^\pi(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)}\left[Q^\pi(s_t, a_t)\right] \tag{45}$$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{46}$$

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}\left[V^\pi(s_{t+1})\right] \tag{47}$$

where the last term is approximately $V^\pi(s_{t+1})$, so that

$$A^\pi(s_t, a_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \tag{48}$$

So, baselines lead us to use advantage as an estimate of reward-to-go in policy gradient. Then, advantage can be expressed using reward and value functions. Therefore, let's just learn V^π.
One way to estimate V^π is through policy evaluation, using Monte Carlo methods, which is what policy gradients does. Alternatively,

$$V^\pi(s_t) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \tag{49}$$

which requires state resetting.
$\hat{V}^\pi_\phi$ is the critic of actor π_θ.
Batch actor-critic:

1. sample $\{s_i, a_i\}$ from $\pi_\theta(a|s)$ (run robot)
2. fit $\hat{V}^\pi_\phi(s_t)$ to sampled reward sums $\left(y_{i,t} \approx r(s_{i,t}, a_{i,t}) + \hat{V}^\pi_\phi(s_{i,t+1})\right)$
3. evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}^\pi_\phi(s'_i) - \hat{V}^\pi_\phi(s_i)$
4. $\nabla_\theta J(\theta) = \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Online:

1. take action $a \sim \pi_\theta(a|s)$, get (s, a, s', r)
2. update $\hat{V}^\pi_\phi(s)$ using target $r + \gamma \hat{V}^\pi_\phi(s')$
3. evaluate $\hat{A}^\pi(s, a) = r(s, a) + \gamma \hat{V}^\pi_\phi(s') - \hat{V}^\pi_\phi(s)$
4. $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Online in practice uses some batch data, leading to asynchronous parallel actor-critic.

Better Advantages

- To reduce variance, use an MC version $\hat{A}^\pi(s_{i,t}, a_{i,t}) = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) - \hat{V}^\pi_\phi(s_{i,t})$
- Or n-step returns: $\hat{A}^\pi_n(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}^\pi_\phi(s_t) + \gamma^n V^\pi_\phi(s_{t+n})$
- Or Generalized Advantage Estimates (similar to eligibility traces):
  $\hat{A}^\pi_{GAE}(s_t, a_t) = \sum_{n=1}^{\infty} w_n A^\pi_n(s_t, a_t)$. Choose $w_n \propto \lambda^{n-1}$, and get
  $\hat{A}^\pi_{GAE}(s_t, a_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'}$, where $\delta_{t'} = r(s_{t'}, a_{t'}) + \gamma \hat{V}^\pi_\phi(s_{t'+1}) - \hat{V}^\pi_\phi(s_{t'})$.

5 Eligibility Traces

TD learning can often be accelerated by the addition of eligibility traces. When the lookup-table TD algorithm described above receives input (r_t, s_{t+1}) , it updates the table entry only for the immediately preceding signal s_t. That is, it modifies only the immediately preceding prediction. But since r_{t+1} provides useful information for learning earlier predictions as well, one can extend TD learning so it updates a collection of many earlier predictions at each step.
There are two interpretations that lead to different algorithms that are equivalent. These are the forward and backward views.

5.1 Forward View

The returns starting in state s_t till time T is given by

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T \tag{50}$$

We may define n-step returns as

$$G_t^{(1)} = r_{t+1} + \gamma V(s_{t+1}) \tag{51}$$
$$G_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+1}) \tag{52}$$
$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}) \tag{53}$$

We can weight n-step returns (total weightb= 1) to define TD target:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{54}$$

Algorithm. Collect episode, for current state s_t, calculate G_t^λ, and then increment $V_t(s_t)$ by $\Delta V_t(s_t) = \alpha \left[G_t^\lambda - V_t(s_t)\right]$.

$$V_{new}(s_0) = V_{old}(s_0) + \alpha \left[r_0 + \gamma V_{old}(s_1) - V_{old}(s_0)\right] \text{ (one-step)} \tag{55}$$
$$V_{new}(s_0) = V_{old}(s_0) + \alpha \left[(1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_0^{(n)} - V_{old}(s_0)\right] \text{ (forward eligibility trace)} \tag{56}$$

This implementation enables an offline version of $TD(\lambda)$, which doesn't really alleviate the waiting-for-n-steps issue of MC methods.

5.2 Backward View

Eligibility traces may also be implemented as a short-term memory of many previous input signals so that each new observation can update the parameters related to these signals. Eligibility traces are usually implemented by an exponentially-decaying memory trace, with decay parameter λ.

Algorithm.

1. During episode, at each time, update for **all** states:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & \text{if} s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1, & \text{if} s = s_t \end{cases} \tag{57}$$

2. In s_t, calculate $\delta_t = r_t + \gamma V(s_{t+1}) -$
3. Update for **all** states $V_{t+1}(s) \leftarrow V_t(s) + \alpha \delta_t e_t(s)$

This implementation enables an online version of $TD(\lambda)$. It may be shown to be equivalent to the updates using the forward view.

5.3 Parameter λ

One interpretation of an eligibility trace is that it is a continuous parametrization, through $\lambda \in [0, 1]$, from Temporal Differences ($\lambda = 0$) at one end to Monte Carlo Methods at the other ($\lambda = 1$). This generates a family of TD algorithms TD(λ), $0 \leq \lambda \leq 1$. For some situations, an intermediate value of λ is better than the extremes. This also applies to non lookup-table versions of TD learning, where traces of the components of the input vectors are maintained. Eligibility traces do not have to be exponentially-decaying traces, but these are usually used since they are relatively easy to implement and to understand theoretically.

6 Options

MDP+Options = SMDP
Theorem: For any MDP, and any set of options, the decision process that chooses among the options, executing each to termination, is an SMDP.

7 Continuous States And Actions

Continuous control in robotics is currently dominated by policy gradient methods, or their actor-critic counterparts. Policy gradient = policy iteration. Evaluate advantage under old policy. New policy maximizes old advantage. Want new policy to factor in new advantage.
So, if we bound KL divergence of the new and old policy, we definitely increase the objective function, that is, distribtion mismatch under changing policies does not kill us.
Implementations:

- TRPO
  Needs a simulator to work, due to the VINE or single-path method
- PPO (optimizes IS objective directly)

7.1 TRPO

Off-policy Importance Sampling that weights the advantage function (or Generalized Advantage Estimate). However, the update step size is limited by a KL-divergence constraint.
The work starts off from [Kakade and Langford, 2002], which analyzes the improvement of a policy $\tilde{\pi}_{\tilde{\theta}}$ relative to a policy π_θ.
Consider a stochastic policy π, and let $\eta(\pi)$ denote its expected discounted reward.

$$\eta(\pi) = \mathbb{E}_{s_0,a_0,\dots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right], \text{ where} \tag{58}$$

$$s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \tag{59}$$

The standard definitions are then

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1},a_{t+1},\dots}\left[\sum_{l=0}^{\infty}\gamma^l r(s_{t+l})\right], \tag{60}$$

$$V^\pi(s_t) = \mathbb{E}_{a_t,s_{t+1},\dots}\left[\sum_{l=0}^{\infty}\gamma^l r(s_{t+l})\right], \tag{61}$$

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s), \text{ where} \tag{62}$$

$$a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ for } t \geq 0 \tag{63}$$

They provide a lower bound on the value of the new policy, that depends on a bunch of terms. This bound starts with

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0,a_0,\dots\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t, a_t)\right], \text{ where} \tag{64}$$

$$a_t \sim \tilde{\pi}(\cdot|s_t) \tag{65}$$

Let ρ_π be the unnormalized discounted visitation frequencies where

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 + \gamma P(s_2 = s) + \cdots, \text{ where } s_0 \sim \rho_0. \tag{66}$$

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{t=0}^{\infty}\sum_{s} P(s_t = s|\tilde{\pi})\sum_{a}\tilde{\pi}(a|s)\gamma^t A^\pi(s,a) \tag{67}$$

$$= \eta(\pi) + \sum_{s}\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\tilde{\pi})\sum_{a}\tilde{\pi}(a|s)A^\pi(s,a) \tag{68}$$

$$= \eta(\pi) + \sum_{s}\rho_{\tilde{\pi}}(s)\sum_{a}\tilde{\pi}(a|s)A^\pi(s,a) \tag{69}$$

This equation imples that any policy update $\pi \to \tilde{\pi}$ that has a nonnegative expected advantage at every state s, i. e. $\sum_a \tilde{\pi}(a|s)A^{\pi}(s,a) \geq 0$, is guaranteed to non-decrease the expected returns. This implies the well known result that $\tilde{\pi} = \arg\max_a A^{\pi}(s,a)$, improves the policy if there is even one state with a positive advantage value and nonzero state visitation probability.
Equation (69) is difficult to optimize because of $\rho_{\tilde{\pi}}(s)$. Instead, we introduce the following local approximation to η:

$$L_{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\pi}(s) \sum_a \tilde{\pi}(a|s)A^{\pi}(s,a) \tag{70}$$

This approximation uses $\rho_{\pi}(s)$, not $\rho_{\tilde{\pi}}(s)$. Implicitly, the approximation ignores changes in state visitation frequency. For differentiable policies, L_{π} matches η to first order.

7.1.1 Results from [Kakade and Langford, 2002]

To address this issue [Kakade and Langford, 2002] proposed a policy updating scheme called conservative policy iteration, for which they could provide explicit lower bounds on the improvement of η. Define $\pi' = \arg\max_{\pi'} L_{\pi_{\text{old}}}(\pi')$. The new policy π_{new} was defined to be the following mixture:

$$\pi_{\text{new}}(a|s) = (1-\alpha)\pi_{\text{old}}(a|s) + \alpha\pi'(a|s) \tag{71}$$

The term α controls the distance between π_{new} and π_{old}. With the new policy so defined, we get the lower bound

$$\eta\left(\pi_{\text{new}}\right) \geq L_{\pi_{\text{old}}}\left(\pi_{\text{new}}\right) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2, \text{ where} \tag{72}$$

$$\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A^{\pi}(s,a)\right]\right| \tag{73}$$

It is likely that this lower bound doesn't depend on π' being defined as it was, but the proof technique might. The validity of this premise is important for clarifying the seeming ambiguity in the use of KL divergence for α in the next section.

7.1.2 Monotonic Improvement Guarantee for General Stochastic Policies

The lower bound on $\eta\left(\pi_{\text{new}}\right)$ contains a term α that controls the distance between π_{new} and π_{old}. It does so by defining a mixture policy involving π_{old} and π', the latter maximizing approximation to $\eta\left(\pi_{\text{old}}\right)$.
Instead of defining direction by π' and euclidean distance by α, TRPO proposes to use a total variation metric for α whose square is bounded by a KL divergence between policies.

$$\eta\left(\pi_{\text{new}}\right) \geq L_{\pi_{\text{old}}}\left(\pi_{\text{new}}\right) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2, \text{ where} \tag{74}$$

$$\epsilon = \max_{s,a} |A^{\pi}(s,a)| \tag{75}$$

The authors use two different approaches to prove this result.
Now,

$$\alpha = D_{\text{TV}}^{\max}\left(\pi_{\text{new}}, \pi_{\text{old}}\right) \tag{76}$$

$$D_{\text{TV}}^{\max}\left(\pi, \tilde{\pi}\right) = \max_s D_{\text{TV}}\left(\pi(\cdot|s) \| \tilde{\pi}(\cdot|s)\right) \tag{77}$$

$$D_{\text{TV}}\left(p\|q\right) = \frac{1}{2}\sum_i |p_i - q_i| \text{ (for discrete distributions)} \tag{78}$$

Since $\left(D_{\text{TV}}\left(p\|q\right)\right)^2 \leq D_{\text{KL}}\left(p\|q\right)$, we can rewrite the bound

$$\eta\left(\tilde{\pi}\right) \geq L_{\pi}\left(\tilde{\pi}\right) - CD_{\text{KL}}^{\max}\left(\pi, \tilde{\pi}\right), \text{ where} \tag{79}$$

$$C = \frac{4\epsilon\gamma}{(1-\gamma)^2}, \text{ and} \tag{80}$$

$$D_{\text{KL}}^{\max}\left(\pi, \tilde{\pi}\right) = \max_s D_{\text{KL}}\left(\pi(\cdot|s) \| \tilde{\pi}(\cdot|s)\right) \tag{81}$$

7.1.3 Algorithm 1

This lower bound can be turned into an algorithm for finding a sequence of policies with increasing expected returns η. In short, for each policy π_i we must calculate the advantage at all state-action pairs, which then defines $L_{\pi_i}(\pi)$. Then, maximize $M_i(\pi) = L_{\pi_i}(\pi) - CD_{\mathrm{KL}}^{\max}(\pi_i, \pi)$, for C calculated appropriately. The idea is that we are maximizing the lower bound for $\eta(\pi_{i+1})$, which is known as minorization-maximization. That is, $\eta(\pi_{i+1}) \geq M_i(\pi_{i+1})$. By definition, $\eta(\pi_i) = L_{\pi_i}(\pi_i) = M_i(\pi_i)$, so that $\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M_i(\pi_i)$.

7.1.4 TRPO Algorithm

Trust region policy optimization, which we propose in the following section, is an approximation to Algorithm 1, which uses a constraint on the KL divergence rather than a penalty to robustly allow large updates.
Instead of penalizing KL divergence in the objective, they use a constraint on KL divergence.

$$\max_{\theta} \quad L_{\theta_{\text{old}}}(\theta) \tag{82}$$

$$s.t. \quad D_{\mathrm{KL}}^{\max}(\theta_{\text{old}} \| \theta) \leq \delta \tag{83}$$

Let q denote an action sampling distribution. These expressions are approximated using

$$\max_{\theta} \quad \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}, a \sim q}\left[\frac{\pi_{\theta_{\text{old}}}(s,a)}{q(s,a)} Q^{\theta_{\text{old}}}(s,a)\right] \tag{84}$$

$$s.t. \quad \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot|s) \| \pi_{\theta}(\cdot|s)\right)\right] \leq \delta \tag{85}$$

The rest of the description discusses how to sample and estimate these objective functions in what is mostly a neighborhood of θ_{old}.
NOTE! This is not a policy gradient step, but a bona-fide optimization using sampling-based local estimates of the objective and KL-divergence constraint.

7.2 PPO

Old summary. Like TRPO, Off-policy Importance Sampling that weights the advantage function (or Generalized Advantage Estimate).
Unlike TRPO, uses a clipping of the importance-weighted advantage estimate.
Achieves a bound on update without needing hard constraint on KL-divergence.
When actor and critic share parameters, need a constraint to keep new critic after actor-driven update to be close to old.
Also need a term to increase exploration, using entropy.

From paper. In the introduction of the PPO paper, the authors write

> Q-learning with function approximation fails on many simple problems and is poorly understood, vanilla policy gradient methods have poor data efficiency and robustness and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that include noise such as dropout or parameter sharing between the policy and value function, or with auxiliary tasks.

They start of with the most basic gradient estimator for policy gradients (PG):

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \hat{A}^{\pi}(s_t, a_t)\right] \tag{86}$$

Here, the expectation $\hat{\mathbb{E}}_t$ indicates the empirical average over a nite batch of samples, in an algorithm that alternates between sampling and optimization.

They mention that this gradient is coming from

$$L_\theta^{PG} = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}^\pi(s_t, a_t) \right] \tag{87}$$

Then they say

> While it is appealing to perform multiple steps of optimization on this loss LPG using the same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates see Section 6 results are not shown but were similar or worse than the "no clipping or penalty" setting.

When recounting TRPO, they state the equations as

$$\max_\theta \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_{\text{old}}}(s_t, a_t)} A^{\theta_{\text{old}}}(s_t, a_t) \right] \tag{88}$$

$$s.t. \quad \hat{\mathbb{E}}_t \left[D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot|s_t) \| \pi_\theta(\cdot|s_t) \right) \right] \leq \delta \tag{89}$$

They then state that

> This problem can efficiently be approximately solved using the conjugate gradient algorithm, after making a linear approximation to the objective and a quadratic approximation to the constraint.

Note that the optimality conditions for the quadratically constrained linear objective lead to a system of linear equations, which is where the CGM comes into the picture.
They remind us that the TRPO turned the KL-divergence penalty into a hard constraint because of the difficulty in choosing a penalty coefficient (C in TRPO paper, β here).

Clipping. They propose to return to the penalty by using clipping.
Define

$$r_t(\theta) = \frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_{\text{old}}}(s_t, a_t)} \tag{90}$$

While TRPO is maximizing

$$L_\theta^{CPI} = \hat{\mathbb{E}}_t \left[r_t(\theta) A^{\theta_{\text{old}}}(s_t, a_t) \right] \tag{91}$$

subject to constraints, where CPI comes from conservative policy iteration, we now maximize

$$L_\theta^{CLIP} = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right] \tag{92}$$

where ϵ is a hyperparameter, they say ≈ 0.2.

Adaptive KL Penalty. An alternate approach is to keep doing the unconstrained penalized version of TRPO, for different values of β until the effect is to achieve a target KL-divergence.

Implementation. For implementations that use automatic differentation, one simply constructs the loss LCLIP or LKLPEN instead of LPG, and one performs multiple steps of stochastic gradient ascent on this objective.

Additional Losses. When the actor and critic share network parameters, a penalty $L_t^{VF}(\theta)$ for fitting the value function is used. Additionally, an incentive for higher entropy $S[\pi_\theta](s_t)$ policies is added, to promote exploration. These two loss terms have coefficients c_1 and c_2 respectively.

7.3 ACKTR

Like TRPO, Off-policy Importance Sampling that weights the advantage function (or Generalized Advantage Estimate).

Also uses PPO's clipping.

From Jonathan's description, uses a natural gradient, unlike PPO.

7.4 DDPG [Lillicrap et al., 2015]

While DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces.
An obvious approach to adapting deep reinforcement learning methods such as DQN to continuous domains is to to simply discretize the action space. However, this has many limitations, most no- tably the curse of dimensionality
DQN is able to learn value functions using such function approximators in a stable and robust way due to two innovations:

1. the network is trained off-policy with samples from a replay buffer to minimize correlations between samples;
2. the network is trained with a target Q network to give consistent targets during temporal difference backups.

In this work we make use of the same ideas, along with batch normalization [Ioffe and Szegedy, 2015], a recent advance in deep learning.
The replay buffer is a finite history of experience units (s_t, a_t, r_t, s_{t+1}), where newer experiences replace older ones, and actor-critic updates happen by uniformly sampling this buffer to create a minibatch of experience. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.
So, we have the state s_t at time t. We take an action a_t coming from the distribution $\pi(s_t)$, and collect rewards $r(s_t, a_t)$. The discounted reward in any run is $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$. The goal is to find the policy π^* that maximizes the expected value of R_1.

$$J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$$

where the $\sim$ operator shows the distributions of the random variables over which the expectation is taken. We denote the discounted state visitation distribution for a policy π as ρ^π .
We can also define

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i>t} \sim \pi}\left[R_t | s_t, a_t\right]$$

The Bellman equations is

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]\right]$$

If the target policy is deterministic we can describe it as a function $\mu: S \leftarrow A$ and avoid the inner expectation in the Bellman equation for Q^π:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^\pi(s_{t+1}, a_{t+1})\right]$$

The expectations depend on the environment here, so that we can learn off-policy. Q-learning is an off-policy algorithm that picks $\mu = \arg\max_a Q^\pi(s, a)$.
We can approximate Q^π and μ using functions parametrized by θ^Q and θ^μ respectively when doing the learning.
With these parametrizations, we can learn by minimizing

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[\|Q(s, a|\theta^Q) - y_t\|^2\right]$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$$

Let's unpack this:

1. Perhaps we use β as a placeholder for the (unknown) policy inducing the distribution in the buffer?
2. The targets y_t make sense of the observed rewards in light of the current policy
3. y_t actually depends on θ^Q but this dependencex is ignored
4. We didn't actually use θ^μ so far. It get used in Deterministic Policy Gradient, as $\mu(s|\theta^\mu)$

So, in DPG the actor μ is updated by computing the gradient of J.

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \right] = \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t} \right]$$

Notice the use of the chain rule.
Directly implementing Q-learning with neural networks proved to be unstable in many environments. Since the network $Q(s, a|\theta^Q)$ being updated is also used in calculating the target value (equation 5), the Q update is prone to divergence. Our solution is similar to the target network used in (Mnih et al., 2013) but modified for actor-critic and using "soft" target updates, rather than directly copying the weights.
We create a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$and $\mu'(s|\theta^{\mu'})$ respectively, that are used for calculating the target values. The weights of these target networks are then updated by having them slowly track the learned networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning. This simple change moves the relatively unstable problem of learning the action-value function closer to the case of supervised learning, a problem for which robust solutions exist. We found that having both a target μ' and Q' was required to have stable targets y_i in order to consistently train the critic without divergence. This may slow learning, since the target network delays the propagation of value estimations. However, in practice we found this was greatly outweighed by the stability of learning.
Batch normalization is the approach to take care of difference in scale among the components of the input.
A major challenge of learning in continuous action spaces is exploration. An advantage of off- policies algorithms such as DDPG is that we can treat the problem of exploration independently from the learning algorithm. We constructed an exploration policy μ' by adding noise sampled from a noise process $\mathcal{N}$ to our actor policy

$$\mu'(s_t) = \mu(s_t|\theta^\mu) + \mathcal{N} \tag{93}$$

7.5 Guided Policy Search

Build local linear dynamics models and quadratic cost models, find a new linear policy using iLQR. Then, train a neural network to produce the same outputs. By corrupting linear policy with noise, we get a stochastic policy, and then use KL-divergence as a way to guide. [Might not be right]

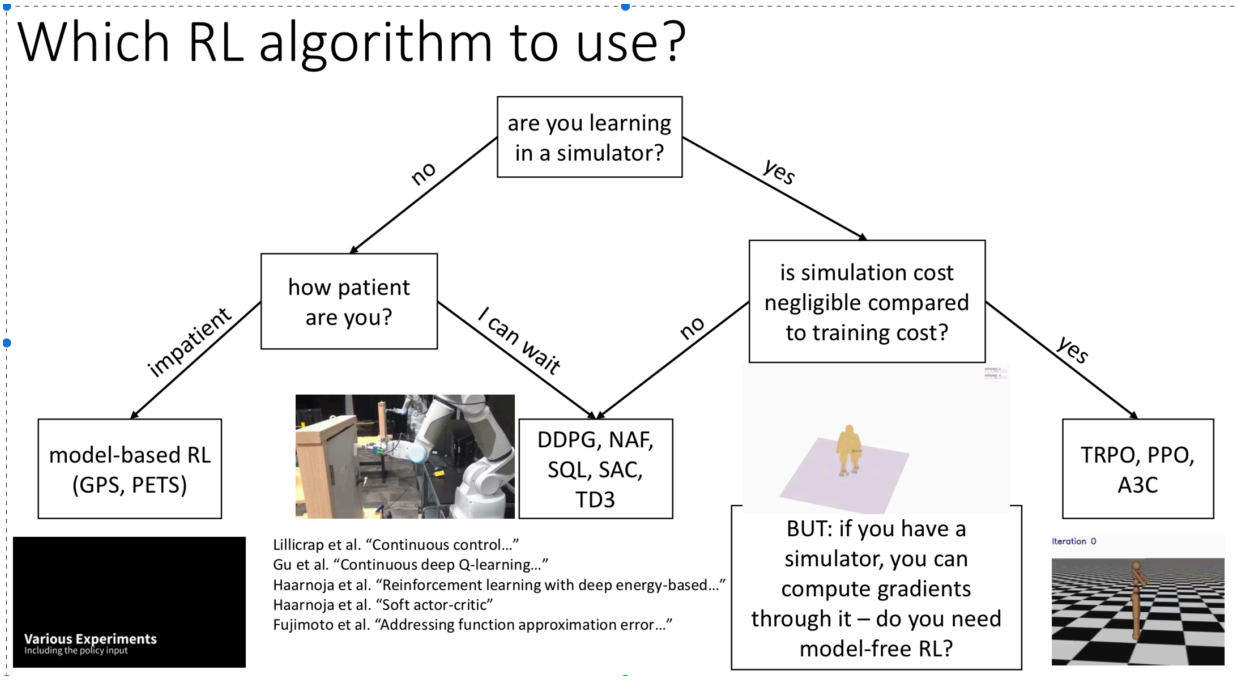

Figure 1: Image credit: Sergey Levine's Lecture Slides

8 Fundamental performance limits

There are two metrics for on-line algorithms: sample complexity and regret.

Sample complexity. Defined as the time required to find an approximately optimal policy. Well defined for any kind of RL problems. A Probably Approximately Correct (PAC) framework.

Regret of an algorithm π . Defined as the difference between the cumulative reward of the optimal policy and that gathered by π. Or, integral off difference between optimal reward and reward gathered by π.
Can have problem-specific lower bounds (given MDP M) or minimax lower-bounds that depend on sizes of S and A.

8.1 Episodic RL State-of-the-Art

Regret minimisation

- Minimax lower bound $\Omega(\sqrt{HSAT})$ No problem-dependent lower bound is derived so far
- Algorithms: PSRL (Osband et al., 2013), UCBVI (Gheshlaghi Azar et al., 2017), UBEV (Dann et al., 2017)

Sample complexity

- Minimax lower bound $\Omega(\frac{H^2SA}{\epsilon^2}\log\frac{1}{\delta})$ No problem-dependent lower bound is derived so far
- Algorithms: UCFH (Dann & Brunskill, 2015), UBEV (Dann et al., 2017)

Upper Confidence Bound Value Iteration for episodic RL

8.2 Discounted RL

No regret analysis for this class.
Two definitions of sample complexity.
Minimimax bounds exist for these, no problem specific bounds.
Two classes of algorithms: Model-based algorithms

- Maintain an approximate MDP model by estimating transition probabilities and reward function, and derive a value function from the approximate MDP. The policy is then derived from the value function.
- E3 (Kearns & Singh, 2002), R-max (Brafman & Tennenholtz, 2002), MoRmax (Szita & Szepesv Iari, 2010), UCRL (Lattimore & Hutter, 2012)

Model-free algorithms

- Directly learn a value (or state-value) function, which results in a policy.
- Delayed Q-Learning (Strehl et al. 2006), Median-PAC (Pazis et al., 2016)

8.3 Ergodic RL

There are four MDP classes

- Ergodic: strongly connected graph under any policy
- Unichain: common strongly connected SUBgraph under any policy
- Communicating: there exists a policy that creates a SCG
- Weakly communicating: There exists a policy with a recurrent class.

Important parameters characterizing MDPs: Diameter, gap1 ,gap2

9 Observations on RL

Summary. Policy gradients are essentially versions of **policy improvement** approaches. These approaches use the advantage of an action at a state to change the policy. Therefore, policy gradients change the policy parameters based on advantages at states.
There are different ways to calculate this advantage, and they are essentially methods of **policy evaluation**.

Opinion. There's a chicken-and-egg problem here the policy determines the connectivity of the MDP, and the connectivity determines the value of that policy in this MDP.

Issue. If your initial policy is bad, your valuation of state is rubbish. The fundamental problem seems to be that we will waste time changing bad policies based on bad estimates.

Q-Learning. Q-learning appears to break the chicken-and-egg problem by specifying the right policy to learn from. The catch is that you need your policy to visit all state-action pairs infinitely often. This need makes it inapplicable to continuous state and/or action spaces, which is what Schulman also observes in the PPO paper.

Implication. TRPO and PPO seem to refuse to update policy parameters unless the returns improve. This approach likely costs more in experience but allows progress, compared to regular policy gradients that might use bad advantage estimates. If you cannot simulate at that level, you're currently SOL in DRL.

Topology and RL. An MDP has no underlying topology of states, until you specify transition probabilities (dynamics, action-dependent), and narrow it down with a policy (closed-loop dynamics). In continuous state and action spaces, these spaces have a topology even before you specify the dynamics or a control that defines the closed-loop. **Is this distinction exploitable, and is anyone doing that already?** It's possible that the value function and policy functions being parametrized as continuous functions via standard neural networks is how this difference is accounted for. However, the learning doesn't seem to care, it's all based on local advantage estimates.

References

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 23

[Kakade and Langford, 2002] Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274. 2, 19, 20

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. 2, 23