Some Key Concepts Used in Pragmatic Reinforcement Learning and Inverse Reinforcement Learning

Prasanth Sengadu Suresh
THINC Lab, School of Computing,
University of Georgia, Athens, GA 30602
prasuchit@gmail.com

Abstract

While writing my doctoral thesis, it occurred to me that others may benefit from a simple, reliable resource that effectively summarizes key concepts used in reinforcement learning (RL) and inverse reinforcement learning (IRL). Consequently, this tutorial was extracted from the background section of my thesis and hence assumes that the reader has a basic knowledge of probability theory, Markov decision processes (MDPs), and reinforcement learning (RL).

Contents

1 Key concepts in pragmatic RL and IRL

In this section, we explain some fundamental concepts that play into almost all recent RL and IRL algorithms. These techniques play an integral role in the quality of learning and are worth understanding in detail.

1.1 Monte-Carlo vs Temporal Difference learning

In this section we briefly compare the most common techniques used to update the value function in reinforcement learning settings.

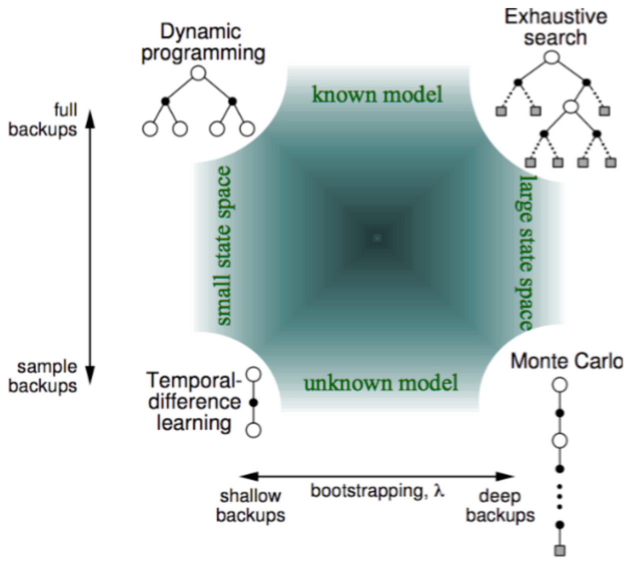

Figure 1: Figure showing the spectrum of classical RL approaches based on the width and depth of bootstrap backups. Image credit: Sutton and Barto (2018).

1.1.1 Monte-Carlo Learning:

The goal of forward RL is to learn a policy π that maximizes the expectation of discounted, long-term reward (also known as the state value function).

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, \pi\right] = \mathbb{E}\left[G_t \,|\, s_0 = s, \pi\right].$$

However, when the dynamics of the environment are unknown, one most common technique to estimate the value function, is by sampling the environment repeatedly and using the returned rewards to estimate the value.

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of state } s_t} + \underbrace{\alpha}_{\text{Learning rate}} \Big[\underbrace{G_t}_{\text{Return at timestep } t} - \underbrace{V(S_t)}_{\text{Current value of state} S_t} \Big].$$

Monte Carlo learning techniques use the simple idea of equating empirical mean return to value (Face). However, this requires complete sampled trajectories and their returns. MC methods usually have high variance but are unbiased estimators of the true value function. Also, since they wait until the end of the episode to update, they are slow to converge.

1.1.2 Temporal Difference Learning:

Temporal Difference (TD) learning, is the concept of using the difference in expected returns between timesteps to update the value function. The most basic version TD-0, only waits one time step before making an update (Face).

TD-0: updates after every step, and hence doesn't have the expected return G_t. Instead, it uses R_{t+1} and the discounted value of the next state to update the value of the current state as shown below:

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of state } S_t} + \underbrace{\alpha}_{\text{Learning rate}} \underbrace{\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD target}} - \underbrace{V(S_t)}_{\text{Current value of state } S_t}\Big]}_{\text{TD Error}}.$$

This is called *bootstrapping*. It's called this because TD bases its update in part on an existing estimate $V(S_{t+1})$ and not a complete sample G_t. By extension, the n-step TD update can be given as:

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of state } s_t} + \underbrace{\alpha}_{\text{Learning rate}} \Big[\underbrace{G_t^{(n)}}_{\text{Return at timestep } t} - \underbrace{V(S_t)}_{\text{Current value of state} S_t}\Big].$$

However, picking the right n is difficult, therefore, we use an intuitive technique to circumvent the problem of picking

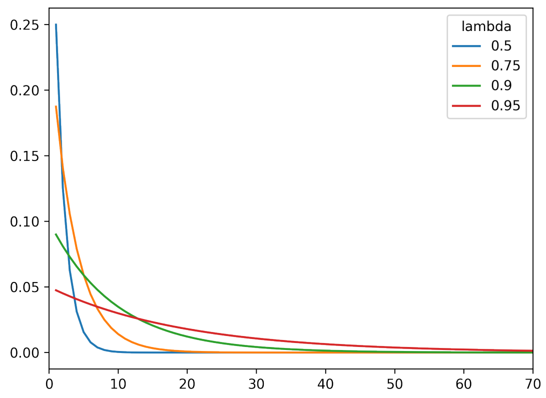

Figure 2: Graph showing the impact of different λ values on the initial return and how it decays over time. Image credit: Reis.

an n value by picking all possible n values at once. This can be done by weighting the n-step return G_t^n by a weight $\lambda \in [0, 1]$ that decays exponentially over time. The nth step is weighted by λ^{n-1} (Reis). λ in the context of TD learning is called the "trace decay parameter" and it controls how much past traces contribute to the current trace. A value close to 0 makes the traces short-term, focusing on recent events, while a value close to 1 makes the traces long-term, considering events further back in time. In other words, λ determines the trade-off between bootstrapping and sampling.

Since we want all of these weights to sum to one (to have a weighted average), we need to normalize them. The normalization constant is easy to derive:

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1-\lambda}.$$

Therefore, multiplying each term by $(1 - \lambda)$ gives us the normalized weights. Now, the λ-return can be written as:

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

The problem with this is the same as that with the Monte Carlo update. We need to wait for a trajectory to complete to compute every possible n-step return and combine them to obtain the λ-weighted average. This flavor of TD-λ is called the *forward view* (Silver).

Eligibility Traces: In the backward view of TD-λ, we use the concept of eligibility traces $e_t(s, a)$ to remember what happened in the past and use current information to update the state-values for every state encountered so far. This is especially useful in cases where the reward is sparse or delayed. Eligibility traces help in assigning credit or blame to states and actions based on their temporal proximity to rewards. They enable the algorithm to remember and track which states and actions occurred recently and adjust their associated values accordingly.

$$\mathbf{e}_t(s,a) = \begin{cases} \gamma\lambda\mathbf{e}_{t-1}(s,a) + 1 & \text{if } (s,a) = (s_t,a_t) \\ \gamma\lambda\mathbf{e}_{t-1}(s,a) & \text{otherwise.} \end{cases}$$

This type of eligibility trace is known as the "accumulating eligibility trace". Alternatively, if we reset a state's eligibility value to 1 every time it is visited instead of adding 1 to it, it is known as the "replacing eligibility trace".

TD-λ Backward View: uses a λ-return target for bootstrapping. A λ-return target takes all n-step targets, and weights each step by λ^{n-1}.

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of } S_t} + \underbrace{\alpha}_{\text{Learning rate}} \cdot \underbrace{\delta_t}_{\text{TD Error}} \cdot \underbrace{e_t}_{\text{Eligibility trace}} \quad \text{where } \delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

We now use the eligibility value as a scaling factor for a TD(0) error. However, notice that every state is updated at once during every update. By propagating the current error information into the states visited in the past, we combine the n-step returns in an online manner.

1.2 Generalized Advantage Estimate

Before we delve into generalized advantage estimate (GAE) (Schulman et al., 2015), let us get some background knowledge to understand how and why GAE improves learning performance.

1.2.1 Advantage Function

The advantage function ($A(s,a)$) measures how much better an action a is compared to others in a given state s (viz., the relative advantage of that action). Mathematically, it is defined as the difference between the state value function ($V(s)$) and the action-value function ($Q(s,a)$):

$$A(s,a) = Q(s,a) - V(s).$$

Bias-variance tradeoff: in machine learning refers to the balance between two types of errors: bias and variance. Bias manifests in oversimplified models that fail to capture the underlying complexities of the data, leading to underfitting. Variance is seen in overly complex models that capture even the noise in the training data as a meaningful signal, resulting in overfitting. Achieving low bias and low variance simultaneously is challenging since reducing one often increases the other.

1.2.2 Generalized Advantage Estimate

While the advantage function is valuable, accurately estimating it can be challenging, especially in environments with long time horizons or high variance in rewards. GAE aims to address these challenges by providing a more stable and efficient estimation of the advantage function. GAE introduces a parameter (λ^{GAE}) that balances the trade-off between bias and variance in estimating the advantage function. This parameter controls how much credit to assign to future states when computing advantages.

$$A_t^{\text{GAE}} = \sum_{\tau=0}^{\infty} (\gamma\lambda^{GAE})^{\tau} \delta_{t+\tau}.$$

where $\delta_{t+\tau}$ represents the advantage at timestep $t+\tau$, t is the timestep at which A is being computed and τ denotes the number of timesteps. Usually λ^{GAE} is chosen based on domain knowledge, empirical experience, heuristics or through theoretical analysis. A common value chosen for λ^{GAE} is between $0.95 - 0.98$. Note that both TD-λ-backward-view and GAE leverage λ to adjust the temporal credit assignment, influencing how credit or blame is propagated backward in time. However, λ in the latter balances generalization versus accurate advantage estimation, while λ in the former affects the trade-off between bootstrapping and sampling in value function updates.

1.3 Importance Sampling

Sampling is widely used in statistics and machine learning to estimate probability distributions that cannot be perfectly captured. Sampling techniques typically draw samples from a target distribution and use approximation techniques to create nearly-identical distributions. While several popular sampling techniques exist like Monte-Carlo, Rejection Sampling, Markov-Chain Monte Carlo, etc exist, one of the most commonly used sampling techniques in RL and IRL is Importance Sampling (Glynn and Iglehart, 1989).

The core idea behind Importance Sampling is to use samples drawn from a known distribution (often termed the "importance distribution" or "proposal distribution") to estimate properties of the target distribution. Instead of directly drawing samples from the target distribution, which might be infeasible, Importance Sampling reweights these samples based on the ratio of the target distribution to the importance distribution. Let us denote the target distribution as P, the importance distribution as Q, and any ith sample drawn from the importance distribution as x_i. The key is that the support of Q should cover the support of P, meaning that Q should assign non-zero probability to regions where P also has non-zero probability.

For each sample drawn from the importance distribution, we compute its weight, which is the ratio of the target distribution's probability density function (pdf) to the importance distribution's pdf at that point. Mathematically, the weight for a sample x_i is given by:

$$w_i = \frac{P(x_i)}{Q(x_i)}.$$

Now, we use these weights to compute estimates of the properties of interest for the target distribution. For example, to estimate the expected value (mean) of a function $f(x)$ with respect to the target distribution P, we compute:

$$\mathbb{E}_P[f(x)] \approx \frac{\sum_{i=1}^{N} w_i f(x_i)}{\sum_{i=1}^{N} w_i}.$$

where N is the number of samples drawn from the importance distribution. Estimating the mean of $f(x)$ with respect to P provides a summary statistic that describes the central tendency of the distribution or function $f(x)$ under consideration (Wikipedia, 2024).

Note that $f(x)$ could be any arbitrary function of x, such as: the pdf of the target distribution P. Estimating the mean of this function with respect to P essentially means estimating the expected value of the distribution itself. It could be a function of a random variable whose expectation you are interested in calculating. For example, the average height of individuals in a population (where x represents height), $f(x)$ could be the height of an individual, and estimating the mean of $f(x)$ with respect to P would give you the average height. $f(x)$ could also represent some function of the variable x that you are interested in estimating, such as a cost function, utility function, or any other measure of interest.

1.4 Lagrangian Method for Constrained Optimization:

Lagrangian method or the Lagrangian Multiplier method (Lagrange, 1853), is a popular technique used to solve constrained optimization problems. Error correction through optimization is ubiquitous in Science and Engineering. Optimization usually entails finding the (global) maxima or minima of a function by iteratively moving towards the point where the gradient of the function with respect to the optimizing variable becomes zero. A constrained optimization problem is one that may have two (common) types of constraints - regional and functional (Courcoubetis and Weber, 2003). For example, consider this generic objective function:

$$L : \underset{x \in X}{maximize} \quad f(x) \text{ subject to } g(x) = b.$$

Here, $x \in X$ is a regional constraint that says the value of x must lie within the space of X. For example, X may be $\mathbb{N}$ - the space of natural numbers. The second constraint $g(x) = b$ is a functional constraint that says the max value obtained for $f(x)$ must also satisfy a secondary condition $g(x) = b$. Let us pick another simple example (Bertsekas, 2014):

$$maximize \quad z = f(x, y) \text{ subject to } g : x + y \leq 100.$$

Using a Lagrange multiplier λ, this equation can be rewritten as:

$$L = f(x, y) + \lambda(100 - x - y)$$

Then we can follow the same steps as any other regular maximization problem:

$$\frac{\partial L}{\partial x} = f_x - \lambda = 0$$
$$\frac{\partial L}{\partial y} = f_y - \lambda = 0$$
$$\frac{\partial L}{\partial \lambda} = 100 - x - y = 0$$

Now we have three equations and three unknowns, which is straightforward to solve. Similarly, the more constraints a problem has, the more number of Lagrangian variables can be used to rewrite it, such that the essence of the optimization is retained. Intuitively, we are trying to find the value for λ such that $\nabla f = \lambda \nabla g$.

1.5 Kullback-Leibler (KL) Divergence:

Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951), proposed by Solomon Kullback and Richard Liebler in 1951, defines a measure of how one probability distribution (P) is different from another, expected probability distribution (Q). It is closely related to relative entropy and is an intuitive way to effectively measure the average likelihood of observing (infinite) data with the distribution P if the particular model Q actually generated the data (Shlens, 2014).

The KL divergence between two probability distributions P and Q is defined as:

$$D_{KL}(P\|Q) = \sum_{x \in \mathcal{X}} P(x) \log_2 \left(\frac{P(x)}{Q(x)} \right).$$

where $\mathcal{X}$ is the set of possible outcomes, $P(x)$ is the probability of observing outcome x according to the distribution P, and $Q(x)$ is the probability of observing outcome x according to the distribution Q.

One popular application of KL divergence is to compute mutual-information. For example, to estimate the similarity of a joint distribution $P(x, y)$ to the product of its marginals $P(x)P(y)$ — this is called mutual information, a general measure of statistical dependence between two random variables (Cover et al., 1991).

$$I(X; Y) = \sum_{x,y} P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)}.$$

Some noteworthy points about KL divergence are:

- KL divergence is not symmetric, meaning $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$.
- It's non-negative, meaning $D_{KL}(P\|Q) \geq 0$, with equality only when $P(x) = Q(x)$ for all x.
- KL divergence quantifies the "information loss" when using a probability distribution Q to approximate P.
- KL divergence is different from Jenson-Shannon (JS) distance (Menéndez et al., 1997) since JS is symmetric. JS distance is half the sum of the KL divergences between the two distributions and their average:

$$D_{JS}(P\|Q) = \frac{1}{2} \left(D_{KL} \left(P\| \frac{P+Q}{2} \right) + D_{KL} \left(Q\| \frac{P+Q}{2} \right) \right).$$

1.6 Wasserstein Metric:

Wasserstein metric (Vaserstein, 1969), also known as Earth Mover's Distance (EMD) or Kantorovich-Rubinstein distance, measures the minimum amount of work required to transform one probability distribution into another.

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{\mathcal{X} \times \mathcal{X}} d(x, y)^p \, d\gamma(x, y) \right)^{1/p}.$$

where μ and ν are probability measures, $\Gamma(\mu, \nu)$ is the set of all joint distributions with marginals μ and ν, $d(x, y)$ is a distance metric between points x and y, p is a parameter determining the order of the distance.

Applications: One of the most popular techniques that use this metric is called WGAN or Wasserstein GAN (Arjovsky et al., 2017). This metric has also been used in a recent RL concept called *Distributional RL* (Bellemare et al., 2017), where instead of learning the expected, discounted, long-term returns (or) the value function, the agent learns the entire distribution of returns. The Distributional Bellman equation for a given state-action pair (s, a) is:

$$Z(s, a) = R(s, a) + \gamma Z'(s', a').$$

where $Z(s, a)$ is the distribution of returns for state-action pair (s, a), $R(s, a)$ is the immediate reward received after taking action a in state s, γ is the discount factor, $Z'(s', a')$ is the distribution of returns for the next state-action pair (s', a'). The distributional Bellman update, which updates the distribution of returns based on the Bellman equation, leads to a contraction in the Wasserstein metric space. This contraction property is desirable because it implies that repeated application of the Bellman operator converges the distribution of returns towards the true distribution, which improves the accuracy of the value estimates.

1.7 Regularization

Regularization (Poggio et al., 1987) is a technique used in machine learning to prevent overfitting and improve the generalization of learned models. In the context of inverse learning, regularization involves penalizing the objective function, to discourage large parameter updates that often lead to overly complex models. This penalty term helps balance the bias-variance trade-off between fitting the training data well and maintaining simplicity in the model.

The goal of regularization is to reduce the model's sensitivity to small fluctuations or noise in the training data, thereby improving its ability to generalize to unseen data. By constraining the model's complexity, regularization can prevent it from memorizing the training data and instead encourage it to capture underlying patterns and relationships that are more likely to hold true across different datasets (Wikipedia, 2024).

Parameter Regularization

Parameter regularization techniques penalize the complexity of the policy or value function by adding a regularization term to the objective function. Common regularization techniques include L^1 and L^2 regularization, which penalize the magnitude of the parameters.

L^1 Regularization: adds a penalty term proportional to the absolute value of the parameters:

$$\text{Regularized Objective} = \text{Original Objective} + \lambda \sum_i |w_i|$$

where w_i are the parameters of the policy or value function, and λ is the regularization strength hyperparameter.

L^2 Regularization: adds a penalty term proportional to the square of the parameters:

$$\text{Regularized Objective} = \text{Original Objective} + \lambda \sum_i w_i^2$$

where w_i and λ have the same meaning as in L^1 regularization.

Entropy Regularization

Entropy regularization encourages exploration by adding the entropy of the policy to the objective function. This is especially helpful in cases of large state-action spaces, such as continuous domains.

$$\text{Regularized Objective} = \text{Original Objective} - \lambda\mathbb{E}[\text{Entropy}(\pi)]$$

where π is the policy, and λ is the regularization strength hyperparameter.

KL Divergence Regularization

KL divergence regularization encourages the learned policy to stay close to a old policy used as reference. It penalizes deviations from the reference policy by adding the KL divergence between the learned policy and the reference policy to the error function. This is the type of regularization used in both TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017).

$$\text{Regularized Objective} = \text{Original Objective} + \lambda\text{KL}[\pi_{\text{new}}\|\pi_{\text{old}}].$$

where λ is the regularization strength hyperparameter.

1.8 Backpropagation and a common conundrum

Backpropagation (Rumelhart et al., 1986), arguably, the backbone of modern deep-learning methods, is a paradigm to efficiently compute gradients of the loss function with respect to the weights of the network. These gradients are then used to update the weights in order to minimize the loss function, which is the objective of learning. Backpropagation involves a forward pass - that uses the input data fed into the network to extract features, layer by layer, while being transformed by activation functions and weighted sums; loss computation - At the last layer, the network makes a prediction based on the weights assigned to the features in that layer. By comparing this prediction to the ground-truth, a loss value is computed; backpropagation - Using chain rule from calculus, starting from the output layer and moving backwards, the gradient of the loss function with respect to the output of each neuron is computed and multiplied with the gradient of the previous layer until the first layer is reached.

Now, assume our objective function L is a function of two variables θ and ϕ and we are interested in optimizing its expected value with respect to both parameters θ and ϕ:

$$L(\theta, \phi) = \mathbb{E}_{X\sim p_\phi(x)}\Big[f_\theta(X)\Big].$$

Since both the objective and its gradients are intractable in general, we estimate them using samples from $p_\phi(x)$. The gradient with respect to θ is straightforward:

$$\nabla_\theta L(\theta, \phi) = \nabla_\theta \mathbb{E}_{X\sim p_\phi(x)}\Big[f_\theta(X)\Big].$$

We can take the gradient ∇_θ into the expectation using the Leibniz integral rule (Amazigo and Rubenfeld, 1980) as follows:

$$\nabla_\theta L(\theta, \phi) = \mathbb{E}_{X\sim p_\phi(x)}\Big[\nabla_\theta f_\theta(X)\Big].$$

Now, L can be estimated using Monte Carlo sampling:

$$\nabla_\theta L(\theta, \phi) \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta f_\theta(X^n) \quad \text{where} \quad X^n \sim p_\phi(x) \text{ i.i.d.}$$

However, when it comes to computing the gradient with respect to ϕ, the problem gets complicated because ∇_ϕ cannot be taken into the expectation since X sampled from $p_\phi(x)$ is also parameterized by ϕ:

$$\nabla_\phi L(\theta, \phi) = \nabla_\phi \mathbb{E}_{X\sim p_\phi(x)}\Big[f_\theta(X)\Big] \neq \mathbb{E}_{X\sim p_\phi(x)}\Big[\nabla_\phi f_\theta(X)\Big].$$

This is a common issue encountered in posterior computation in variational inference, value function and policy learning in reinforcement learning, derivative pricing in computational finance, and inventory control in operations research, amongst many others.

1.8.1 Score Function Estimators

This is one of the methods that can help sidestep the aforementioned predicament (Fu, 2006; Williams, 1992; Glynn, 1990). This uses a method called "log-derivative trick" (which says: $\nabla_\phi p_\phi(x) = p_\phi(x)\nabla_\phi \log p_\phi(x)$) to express the gradient as:

$$\nabla_\phi L(\theta, \phi) = \mathbb{E}_{X\sim p_\phi(x)}\Big[f_\theta(X)\nabla_\phi \log p_\phi(X)\Big].$$

Now, we can compute the expectation of the gradient using Monte Carlo estimate as:

$$\nabla_\phi L(\theta, \phi) \approx \frac{1}{N}\sum_{n=1}^{N} f_\theta(X^n)\nabla_\phi \log p_\phi(X^n) \quad \text{where} \;\; X^n \sim p_\phi(x) \text{ i.i.d.}$$

The advantage of this method is that it does not require $f_\theta(x)$ to be differentiable or even continuous as a function of x, this method can be used with both discrete and continuous variables. The downside is that this basic version can suffer from high variance which would then require variance reduction techniques to obtain a usable estimate.

1.8.2 Reparameterization Trick

This method is the low variance alternative that simply samples from a fixed distribution $q(z)$ and transforms it using a function $g_\phi(z)$ to represent a sample obtained from $p_\phi(x)$. For example, by sampling from the standard Normal distribution $\mathcal{N}(0, 1)$ and transforming it using the location-scale transformation technique: $g_{\mu,\sigma} = \mu + \sigma Z$, we can get a sample from $\mathcal{N}(\mu, \sigma^2)$. This two-stage reformulation of the sampling process, called the reprameterization trick (Maddison et al., 2016), allows us to transfer the dependence on ϕ from p into f by writing $f_\theta(x) = f_\theta(g_\phi(z))$ for $x = g_\phi(z)$.

$$L(\theta, \phi) = \mathbb{E}_{X\sim p_\phi(x)}\Big[f_\theta(X)\Big] = \mathbb{E}_{Z\sim q_z(x)}\Big[f_\theta(g_\phi(Z))\Big].$$

Since $q(z)$ does not depend on ϕ, the gradient can now be estimated similar to the gradient with respect to θ.

$$\nabla_\phi L(\theta, \phi) = \mathbb{E}_{Z\sim q_z(x)}\Big[\nabla_\phi f_\theta(g_\phi(Z))\Big] = \mathbb{E}_{Z\sim q_z(x)}\Big[f'_\theta(g_\phi(Z))\nabla_\phi g_\phi(Z)\Big].$$

In the context of RL, specifically policy gradient techniques, this trick is used to sample from a known, deterministic distribution that can be transformed to obtain sampled actions to compute the gradient during backpropagation (Schulman et al., 2015).

1.9 Target Networks

The difference between Q-learning and DQN (Mnih et al., 2015) is that the latter replaces an exact value function with a function approximator - a neural network. While Q-learning updates exactly one state/action value at each timestep, DQN updates many. This causes the action values for the very next state to drastically change from the previous timestep, causing drastic instability in learning. The effect is typically referred to as "catastrophic forgetting" in neural network literature (French, 1999). Using a stable, secondary target network as the error measure is one way of combating this effect. Conceptually, it is equivalent to waiting for a significant change in values before updating the network as opposed to updating it after every time step. By allowing the network more time to consider many recent actions, learning stability increases manifold.

2 Comparison of forward RL methods

For our application, we need a technique that can learn a decentralized policy using a centralized reward function. We also need a method that can work both for continuous and discrete action spaces and can converge well in both. As can be seen from Section 2 and Section 2, TRPO and PPO are the only two options among the most popular RL techniques that have all these features. (Schulman et al., 2017) shows that the clipped surrogate objective used by PPO allows

Algorithm	**Objective Function**
Q-learning	$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$
SARSA	$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
DQN (Deep Q Network)	$L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]$
A3C (Asynchronous Advantage Actor-Critic)	$L(\theta) = \sum_t \big(- \log(\pi(a_t\|s_t;\theta))(R_t - V(s_t;\theta_v)) + \beta H(\pi(\cdot\|s_t;\theta))\big)$
DDPG (Deep Deterministic Policy Gradient)	$L(\theta^\mu) = -\mathbb{E}_{s_t\sim\rho^\beta}[\nabla_{\theta^\mu} Q(s,a\|\theta^Q)\|_{s=s_t,a=\mu(s_t\|\theta^\mu)}]$
SAC (Soft Actor Critic)	$J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[\alpha \log \pi_\theta(a_t\|s_t) - Q_\phi(s_t,a_t)]$
PPO (Proximal Policy Optimization)	$L(\theta) = \mathbb{E}_t\left[\min\left(\frac{\pi_\theta(a_t\|s_t)}{\pi_{\theta_{\text{old}}}(a_t\|s_t)} A^{\text{clip}}(\theta), 1+\epsilon, 1-\epsilon\right) A_t - \beta H(\pi(\cdot\|s_t;\theta))\right]$
TRPO (Trust Region Policy Optimization)	$\max_\theta \mathbb{E}_t\big[\frac{\pi_\theta(a_t\|s_t)}{\pi_{\theta_{\text{old}}}(a_t\|s_t)} A^{\text{clip}}(\theta)\big]$

Table 1: Objective functions of key RL algorithms (Larsen et al., 2021; Achiam, 2018)

Algorithm	**Obsv Space**	**Action Space**	**Type**	**Policy**
Q-learning	Discrete	Discrete	Value-based	Off-policy
SARSA	Discrete	Discrete	Value-based	On-policy
DQN (Deep Q Network)	Continuous	Discrete	Value-based	Off-policy
A3C (Asynchronous Advantage Actor-Critic)	Continuous	Discrete	Actor-critic	On-policy
DDPG (Deep Deterministic Policy Gradient)	Continuous	Continuous	Actor-critic	Off-policy
SAC (Soft Actor Critic)	Continuous	Continuous	Actor-critic	Off-policy
PPO (Proximal Policy Optimization)	Continuous	Continuous or Discrete	Actor-critic	On-policy
TRPO (Trust Region Policy Optimization)	Continuous	Continuous or Discrete	Actor-critic	On-policy

Table 2: Comparison of key RL algorithms (Larsen et al., 2021; Achiam, 2018; Wikipedia, 2024).

it to use a pessimistic bound over the objective function and hence converge to a better optima. Yu et al. (2022) also show that PPO scales better when extended to cooperative multiagent scenarios. This is why PPO is one of the most sought-after RL algorithms currently.

We used the implementation of StableBaselines3 to design Dec-PPO for the forward-rollout phase of Dec-AIRL, however, there are many other popular RL libraries that provide elegant implementations of PPO like Ray RLLib, Dopamine, ACME, Mushroom-RL, Tianshou, etc. While Stable-Baselines3 provides almost all the features required for most RL reseach applications, it also has some experimental techniques like Recurrent-PPO, on its contrib branch of the library.

References

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

H. Face. Deep rl course. `https://huggingface.co/learn/deep-rl-course/en/unit2/mc-vs-td`.

A. Reis. Rl blog. `https://amreis.github.io/ml/reinf-learn/2017/11/02/reinforcement-learning-eligibility-traces.html`.

D. Silver. Rl lectures. `https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf`.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

P. W. Glynn and D. L. Iglehart. Importance sampling for stochastic simulations. *Management science*, 35(11):1367–1392, 1989.

Wikipedia. Imporance sampling, 2024. URL `https://en.wikipedia.org/wiki/Importance_sampling`.

J. L. Lagrange. *Mécanique analytique*, volume 1. Mallet-Bachelier, 1853.

C. Courcoubetis and R. Weber. Lagrangian methods for constrained optimization. *Wiley-interscience series in systems and optimization*, page 333, 2003.

D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

J. Shlens. Notes on kullback-leibler divergence and likelihood. *arXiv preprint arXiv:1404.2000*, 2014.

T. M. Cover, J. A. Thomas, et al. Entropy, relative entropy and mutual information. *Elements of information theory*, 2 (1):12–13, 1991.

M. Menéndez, J. Pardo, L. Pardo, and M. Pardo. The jensen-shannon divergence. *Journal of the Franklin Institute*, 334(2):307–318, 1997.

L. N. Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/arjovsky17a.html`.

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. *Readings in computer vision*, pages 638–643, 1987.

Wikipedia. Regularization in machine learning, 2024. URL `https://en.wikipedia.org/wiki/Regularization_(mathematics)`.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, None(None):None, 2017.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323 (6088):533–536, 1986.

J. C. Amazigo and L. A. Rubenfeld. Advanced calculus and its applications to the engineering and physical sciences. *(No Title)*, 1980.

M. C. Fu. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems*, 28, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

T. N. Larsen, H. Ø. Teigen, T. Laache, D. Varagnolo, and A. Rasheed. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI*, 8:738113, 2021.

J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

Wikipedia. Comparison of key reinforcement learning algorithms, 2024. URL `https://en.wikipedia.org/wiki/Reinforcement_learning#Comparison_of_key_algorithms`.

C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.