Machine Learning - Project Report

Lukas Johannes Ruettgers Isaac Hong Zhang Jie Chenrui Guo

January 10, 2024

1 Introduction

2 Reinforcement Learning

2.1 Problem Formulation

Reinforcement Learning addresses the problem of learning successful behaviour in an unknown environment. This environment is usually modeled as a stochastic process, namely a *Markov Decision process* (MDP) (S, A, r, P, γ). Every process-relevant detail of the environment is included in a state vector $s \in S$, where S is the set of all possible states the environment can be in. The agent has a state-independent set of actions $a \in A$ that describe his options to influence or interact with the environment. The current state s of the environment is usually affected by the action a of the agent, else the action would be useless. The *transition model* P describes how this action changes the environment by providing the new distribution of the states $P(s' \mid s, a)$ after the action was executed. By the Markov assumption, the subsequent state only depends on the prior state and the action and not on states or actions that happened before. Further note that stochastic processes like the MDP regard time as evolving in fixed time steps $t = 0, 1, \dots$, such that the above term can also be written as $P(s_{t+1} \mid s_t, a_t)$. We however refrained from explicitly denoting the time index in the definition of the transition model P as it might suggest that the transition rule is time-dependent. However, we can consider time-independent transition rules without loss of generality since we can model any time-dependent variables in the state variable s itself. After an action a was executed and led to a new state s', the agent receives a reward $r(s, a, s')$ defined by the reward function $r : S \times A \times S \mapsto \mathbb{R}$. This is the only signal from which the agent can infer what behaviour is desirable. The reinforcement learning problem therefore models learning experience as the experience of state transitions (s, a, r, s'), from which the agent shall derive which actions lead in which states to what rewards. Naïvely, we might formulate the objective of the agent as maximizing the expected cumulative reward

$$J(a_0, \cdots) = \mathbb{E}_{s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)} \left[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) \right]$$

over a given *horizon* T. However, this objective is ill-defined for an infinite horizon $T = \infty$, where any recurring non-zero reward could accumulate to infinity and render the maximization problem useless. To account for this, a *discount factor* $\gamma \in (0, 1)$ is usually used to model the rate with which rewards become more irrelevant over time, such that the objective becomes

$$J(a_0, \cdots) = \mathbb{E}_{s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \right].$$

A common choice is $\gamma = 0.99$, which means that after 100 steps, the reward experienced after the very first action is approximately 20 times less important than the current reward perceived at the 100th step. However, this problem formulation is still impractical, since an infinite horizon T will have infinite number of actions as optimization parameters. Instead, we generally describe the behaviour of an agent by a *policy* $\pi(a \mid s)$, which defines a probability distribution over the actions conditioned

on each possible state. For a finite number of states and actions, this policy can also be regarded as a lookup table that stores the optimal action distribution in each state. In the case of continuous state domains, this approach renders infeasible and we are left with approximating this function by a set of finite parameters θ. Neural network architectures have proven to serve as accurate non-linear function approximators and are widely used in current Deep RL algorithms to approximate $\pi(a \mid s) \approx \pi_\theta(a \mid s)$. The objective function then finally writes as

$$\begin{aligned} J(\theta) &= \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \prod_{i=0}^{t} \left(P(s_{i+1} \mid s_i, a_i)\pi_\theta(a_i \mid s_i)\right) \\ &= \mathbb{E}_{s_{t+1}\sim P(s_{t+1}|s_t,a_t), a_t\sim\pi_\theta(a_t|s_t)} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1})\right]. \end{aligned}$$

In the following, we will write $\mathbb{E}_{s_t,a_t\sim\pi_\theta}[\cdot]$ for notational compactness and to steer the focus on the policy that is to be learnt, while the transition model p is given albeit unknown.

2.2 Algorithms and Approaches

2.2.1 Value-based methods

The popular methods in classical RL are often categorized as *value-based* and *policy-based* methods. Value-based methods consider the RL problem as a problem of learning the true value of a state $V(s)$ with regard to the expected cumulative reward that is obtainable from it. That is, we define

$$V(s) = \mathbb{E}_{s_t,a_t\sim\pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \mid s_0 = s\right].$$

Similarly, we can further condition this value on the action chosen in this state and obtain a *state-action value*

$$Q(s, a) = \mathbb{E}_{s'\sim P(s'|s,a)} \left[r(s, a, s') + V(s')\right].$$

If such value functions could be accurately estimated, then an approximately optimal policy could be obtained by deterministically choosing the action that is expected to lead to the states with the highest rewards,

$$a^* = \arg\max_{a\in A} Q(s, a) = \arg\max_{a\in A} \mathbb{E}_{s'\sim P(s'|s,a)} \left[r(s, a, s') + V(s')\right], \quad \pi^*(a \mid s) = \begin{cases} 1, & a = a^* \\ 0, & a \neq a^* \end{cases}.$$

The above equality between $V(s)$ and $Q(s, a)$ with regard to optimality in a is known as the *Bellman equation* and paves thFRe way to solving the problem via Dynamic Programming, where an initial estimate $V^{(0)}(s)$ of each state's value is iteratively updated by empirically perceived rewards,

$$V^{(j+1)}(s) = \max_{a\in A} \mathbb{E}_{s'\sim P(s'|s,a)} \left[r(s, a, s') + V(s')\right].$$

In a similar fashion, $Q(s, a)$ is iteratively updated.

2.2.2 Policy-based methods

The above schematic algorithm requires executing each action in each state, while the final policy is defined at the end. An alternative approach might include policy behaviour directly from the beginning. That is, we do not update the value functions by executing each possible action and regard them as equally likely, but executing actions with a probability according to a current policy. Because the policy and value functions depend on each other, this paves the way to an iterative update of the policy and the value functions. In the *policy evaluation* step, we firstly update our value functions as

$$V_\pi^{(j+1)}(s) = \mathbb{E}_{s'\sim P(s'|s,a), a\sim\pi(s)} \left[r(s, a, s') + V(s')\right]$$

for a fixed number of steps or until a convergence criteria is satisfied. Then, we use the updated value functions for *policy improvement* and update $\pi \leftarrow \arg\max_{a \in A} Q_\pi(s, a)$. The repeated iteration over these two processes is called *policy iteration* and often serves as a template for more sophisticated methods. Note that the policy that is improved — the *target policy* — and the policy that is used for interaction with the environment in the policy evaluation step — the *behaviour policy* — can be different. This might be desirable to foster exploration during training, while our final policy should completely exploit the learned knowledge to make the optimal choices. Our current formulation uses the same greedy policy for both steps and hence constitutes an *on-policy* algorithm. But deviating from the target policy with a small probability $\varepsilon \in (0, 1)$ and randomly choosing the next action to explore the environment will allow the behaviour policy to escape from local minima and widen its horizon of reward experiences in view of environments that are too high-dimensional or costly too access exhaustively. This *ε-greedy* behaviour policy is the most widely adapted example of an *off-policy* algorithm, but one could also re-use old policies to enrich the learning progress with prior experience. However, both of the formulations above are intractable for infinite state spaces, which we usually face in real-world problems, which requires heuristic approximations in practice.

2.2.3 Policy Gradient

One alternative viewpoint of the RL problem formulation is to directly regard it as a parametrized optimization problem

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \right].$$

To simplify the following notation, we introduce the term *trajectory* $\tau = (s_0, a_0, \ldots, s_T)$ to describe an entire state-action sequence and abbreviate the probability of a trajectory as

$$P_{\pi_\theta}(\tau) = P(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t),$$

where $P(s_0)$ is the prior probability distribution of initial states, and define the reward of a trajectory as

$$r(\tau) = \sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}).$$

This way, we streamline the definition of the objective function to $J(\theta) = \mathbb{E}_{\tau \sim P_{\pi_\theta}} [r(\tau)]$. However, note that this is just a notational simplification. The semantic equality only holds if we multiply the probability of the trajectory from s_0 to s_t with the reward *from* t to T and not from 0 to T, where $0 \leq t \leq T - 1$. This imperfection does not matter for the following theoretical analysis in which we can regard the rewards as constants that have been collected during execution in the environment. To maximize $J(\theta)$ using gradient ascent, we first have to derive how to compute the gradient; or more specifically, how we compute the gradient of $\pi_\theta(a_t \mid s_t)$, since all other terms are independent of θ. Owing to the chain rule, we have the equality

$$\pi_\theta(a_t \mid s_t) \cdot \nabla \log\left(\pi_\theta(a_t \mid s_t)\right) = \pi_\theta(a_t \mid s_t) \cdot \frac{1}{\pi_\theta(a_t \mid s_t)} \cdot \nabla \pi_\theta(a_t \mid s_t) = \nabla \pi_\theta(a_t \mid s_t), \quad (PG)$$

which allows us to obtain the Policy Gradient Theorem

$$
\begin{aligned}
\nabla J(\theta) &= \nabla \mathbb{E}_{\tau \sim P_{\pi_\theta}}\left[r(\tau)\right] \\
&\overset{\text{Def.}}{=} \nabla \int_\tau P_{\pi_\theta}(\tau) r(\tau)\, d\tau \\
&\overset{\text{Def.}}{=} \nabla \int_\tau \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \prod_{i=0}^{t} \left(P(s_{i+1} \mid s_i, a_i)\pi_\theta(a_i \mid s_i)\right)\, d\tau \\
&\overset{\text{sum rule}}{=} \int_\tau \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \nabla \prod_{i=0}^{t} \left(P(s_{i+1} \mid s_i, a_i)\pi_\theta(a_i \mid s_i)\right)\, d\tau \\
&\overset{\text{prod. rule}}{=} \int_\tau \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \sum_{j=0}^{t} P(s_{j+1} \mid s_j, a_j) \left(\nabla \pi_\theta(a_j \mid s_j)\right) \prod_{i=0, i\neq j}^{t} \left(P(s_{i+1} \mid s_i, a_i)\pi_\theta(a_i \mid s_i)\right)\, d\tau \\
&\overset{\text{(PG)}}{=} \int_\tau \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \sum_{j=0}^{t} \left(\nabla \log \pi_\theta(a_j \mid s_j)\right) \prod_{i=0}^{t} \left(P(s_{i+1} \mid s_i, a_i)\pi_\theta(a_i \mid s_i)\right)\, d\tau \\
&\overset{\text{Def.}}{=} \mathbb{E}_{\tau \sim P_{\pi_\theta}}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_{t+1}) \sum_{j=0}^{t} \nabla \log \pi_\theta(a_j \mid s_j)\right] \\
&\overset{\text{rearrange}}{=} \mathbb{E}_{\tau \sim P_{\pi_\theta}}\left[\sum_{j=0}^{T-1} \nabla \log \pi_\theta(a_j \mid s_j) \sum_{t=j}^{T-1} \gamma^t r(s_j, a_j, s_{j+1})\right] \\
&\overset{T\to\infty}{=} \mathbb{E}_{\tau \sim P_{\pi_\theta}}\left[\sum_{j=0}^{\infty} \nabla \log \pi_\theta(a_j \mid s_j) Q_{\pi_\theta}(s_j, a_j)\right].
\end{aligned}
$$

The benefit of this transformation is that we reduced computing the gradient of an expectation value over the rewards to the expectation value of a policy gradient term. Even though the rigorous exact computation remains intractable for infinite or high-dimensional state and action spaces, we can avail to common strategies in Inference Statistics to estimate this expectation value given only few samples of the entire population of possible trajectories. That is, one could for example collect N trajectories at each gradient ascent step and approximate the estimation value of the log terms as the arithmetic mean of the log terms over these N trajectories, as is done in classical *Monte Carlo* RL. This is the approach that the REINFORCE algorithm takes, which belongs to one of the first algorithms in the category of policy gradient methods. To further alleviate the computation of the log term itself, REINFORCE approximates $\pi(a \mid s)$ as a normal Gaussian probability distribution. While the mean of this distribution is estimated using a neural network $\mu \approx f_\theta(s)$, the covariance matrix is often heuristically fixed as Σ. This way, we have

$$
\pi_\theta(a_t \mid s_t) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left(-\frac{1}{2}(f_\theta(s_t) - a_t)\Sigma^{-1}(f_\theta(s_t) - a_t)\right), \text{ and consequently}
$$

$$
\log \pi_\theta(a_t \mid s_t) = \log\left(\frac{1}{\sqrt{(2\pi)^n |\Sigma|}}\right) - \frac{1}{2}(f_\theta(s_t) - a_t)\Sigma^{-1}(f_\theta(s_t) - a_t).
$$

Because the first term is constant with regard to θ, this finally yields us the gradient

$$
\nabla \log \pi_\theta(a_t \mid s_t) = -\Sigma^{-1}(f_\theta(s_t) - a_t)\frac{\partial f_\theta}{\partial \theta}.
$$

As all activation functions in today's neural network architectures are differentiable, $\frac{\partial f_\theta}{\partial \theta}$ is simply obtained for each weight θ_i by the back-propagation algorithm. The pseudocode for REINFORCE is depicted below in Figure 1.

Algorithm 1 REINFORCE

Initialize: Neural network π_θ with parameters θ, learning rate α, covariance matrix Σ, batch size N.

while not converged **do**
  Collect N trajectories $(s_0, a_0, r_0, \dots, s_T)$ by executing π_θ and store them in a buffer B.
  for all trajectories τ in B **do**
    for all transitions $(s_t, a_t, r_t) \in \tau$ **do**
      Forward s_t through π_θ to obtain the distribution $\pi_\theta(a_t \mid s_t)$.
      Track the gradient history to obtain $\nabla \log \pi_\theta(a_t \mid s_t) = -\Sigma^{-1}(f_\theta(s_t) - a_t)\frac{\partial f_\theta}{\partial \theta}$.
      Run the BP algorithm to update $\theta \leftarrow \theta + \alpha \cdot \nabla \log\left(\pi_\theta(a_t \mid s_t)\right) r_t$.
    end for
  end for
end while

Figure 1: REINFORCE pseudo code algorithm.

As for normal Monte-Carlo RL methods, the estimation on long-horizon trajectories exhibit a huge variance, because the domain of possible state-action combinations grows exponentially in the horizon T. On the other side, the *Temporal Difference* update

$$V^{(j+1)}(s) = V^{(j)}(s) + \alpha\left(r(s, a, s') + \gamma V^{(j)}(s') - V^{(j)}(s)\right),$$

which considers only one transition with empirical reward in the update of the value function, has a low variance. But it has a larger bias, which comes from the larger weight estimate of the value $V(s_{t+1})$ of the subsequent state, which was only discounted once by the discount rate γ. To mitigate the variance while preserving a low bias, an unbiased baseline can be subtracted from the rewards. The value function $V_{\pi_\theta}(s)$ turns out to be such a good baseline, resulting in the adjusted term

$$J(\theta) = \mathbb{E}_{\tau \sim P_{\pi_\theta}}\left[\sum_{j=0}^{\infty} \nabla \log \pi_\theta(a_j \mid s_j)\left(Q_{\pi_\theta}(s_j, a_j) - V_{\pi_\theta}(s)\right)\right].$$

This term is also known as the *advantage function* $A_{\pi_\theta}(s_t, a_t) := Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s)$, which measures the advantage, when taking action a_t over alternative actions in s_t and continuing to act by the policy π_θ. If the rewards perceived in this trajectory are high, then a_t truly exhibits an advantage over the average action, because $V(s_t)$ is defined as the *expectation value* of cumulative rewards. To approximate this advantage function with a good trade-off between Temporal Difference with low variance and Monte Carlo with low bias, one could theoretically choose any $1 \leq n \leq T$, accumulate the empirical rewards until step n and cap off with the prior estimate of the value function at the n-th state,

$$V^{(j+1)}(s_t) = A_n(s_t, a_t) := \sum_{i=0}^{T-1}\left(\gamma^i r(s_{t+i}, a_{t+i}, s_{t+i+1})\right) + \gamma^n V^{(j)}(s_{t+n}) - V^{(j)}(s_t).$$

This term $A_n(s_t, a_t)$ is often referred to as the n-step approximation of the *advantage function*. The *Generalized Advantage Estimation* combines all these terms into one and steers their importance decay with one hyperparameter $\lambda \in (0, 1)$, culminating in

$$A_{GAE}(s_t, a_t) = \frac{1}{1-\lambda}\sum_{n=1}^{T} \lambda^n A_n(s_t, a_t) = \frac{1}{1-\lambda}\sum_{i=0}^{T}(\lambda\gamma)^i \left(r(s_{t+i}, a_{t+i}, s_{t+i+1} + \gamma V(s_{t+i+1}) - V(s_{t+i})\right).$$

2.2.4 Proximal Policy Optimization

There are also numerous other optimization techniques to improve the objective function $J(\theta)$. To unravel this bouquet of opportunities, consider an alternative parametrization θ'. The *Performance*

Difference Lemma, whose proof follows a similar spirit as the one for the Policy Gradient Theorem, states that

$$J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim P_{\pi_{\theta'}}} \left[\sum_{t=0}^{\infty} A_{\pi_\theta}(s_t, a_t) \right],$$

which intuitively means that θ' is desirable over θ if the new policy $\pi_{\theta'}$ executes those actions with a higher probability that are advantageous even if we follow the old policy π_θ in the subsequent states. Unfortunately, the evaluation of this term requires an approximation of the distribution of states and actions under the very θ' that we are trying to optimize. Naïvely, this would require to empirically estimate the distribution under $P_{\pi_{\theta'}}$ with a large number of trajectories for each θ' we want to consider as a new parameter candidate, which imposes a severe computational limitation on the number of θ's that one can explore in each update step. Instead we can constrain θ' to remain close to θ with regard to P_{π_θ}. Then approximating

$$\mathbb{E}_{\tau \sim P_{\pi_{\theta'}}} \left[\sum_{t=0}^{\infty} A_{\pi_\theta}(s_t, a_t) \right] \approx \mathbb{E}_{\tau \sim P_{\pi_\theta}} \left[\sum_{t=0}^{\infty} \frac{\pi_{\theta'}(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} A_{\pi_\theta}(s_t, a_t) \right]$$

will not exhibit a large distributional shift in the likelihood of states. *Trust Region Policy Optimization* formulates this as a constrained optimization problem and maximize

$$J(\theta') = \mathbb{E}_{\tau \sim P_{\pi_\theta}} \left[\sum_{t=0}^{\infty} \frac{\pi_{\theta'}(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} A_{\pi_\theta}(s_t, a_t) - \delta \, \mathrm{KL}\left(\pi_\theta(a_t \mid s_t) || \pi_{\theta'}(a_t \mid s_t)\right) \right],$$

which can be solved by conjugate gradient algorithm that however requires second-order derivatives to approximate the penalty KL divergence term. Instead, *Proximal Policy Optimization* proposes a first-order approximation

$$J(\theta') = \mathbb{E}_{\tau \sim P_{\pi_\theta}} \left[\sum_{t=0}^{\infty} \min \left(1 + \varepsilon, \frac{\pi_{\theta'}(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} \right) A_{\pi_\theta}(s_t, a_t) \right]$$

that clips overly advantageous deviations from the current policy π_θ and therefore evokes a more stable, stepwise optimization procedure. Since the estimation of the advantage function requires a state-value function V, a separate neural network is trained as *critic*. Together with the critic loss and a term that encourages entropy on the action distribution of $\pi_{\theta'}(a_t \mid s_t)$, the above $J(\theta')$ constitute the loss function $J(\theta', \psi')$. The original paper trains multiple agents in parallel ((1)), but since it is no specific feature of PPO, we omitted it in the pseudocode in Figure 2.

Algorithm 2 Proximal Policy Optimization

Initialize: Actor network π_θ, value network V_ψ, clip offset ε, scaling coefficients c_1, c_2 for entropy and value loss, batch size N, training epochs K, minibatch size M.

while not converged **do**
  Collect T transitions (s_t, a_t, r_t, s_{t+1}) by executing π_θ and store them in a buffer B.
  Compute advantage estimators $\hat{A}^{(i)}_{\pi_\theta}$ for each transition $1 \leq i \leq T$.
  for epoch $1 \leq j \leq K$ **do**
    for all transitions (s_t, a_t, r_t, s_{t+1}) in minibatch of size M **do**
      Optimize θ' and ψ' with regard to $J(\theta', \psi')$.
    end for
    Update $\theta \leftarrow \theta', \psi \leftarrow \psi'$
  end for
end while

Figure 2: PPO pseudo code algorithm.

3 Experiment Objective

4 Experiment Results

5 Conclusion

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.