Active Learning for Reward Modelling

Sam Clarke

September 2019

Abstract

Contents

Chapter 1

Introduction

1.1 Relation to Material Studied on the MSc Course

Chapter 2

Reinforcement Learning

Reinforcement learning (RL) refers simultaneously to a problem, methods for solving that problem, and the field that studies the problem and its solution methods. The problem of RL is to learn what to do—how to map situations to actions—so as to maximise some numerical reward signal [5, pp. 1-2].

In this section we first introduce the elements of RL informally. We then formalise the RL Problem as the optimal control of incompletely-known Markov decision processes (finite? do I talk about PO?). We give a taxonomy of different RL solution methods and conclude with a description of one such method, Deep Q-Learning (DQN) that is of particular importance in this dissertation.

2.1 Elements of Reinforcement Learning

This subsection will introduce agent, environment, policy, reward signal, value function and [model] informally, similar to S&B 1.3. Is this necessary or should I skip straight to the formalism?

2.2 Finite Markov Decision Processes

Finite Markov Decision Processes (finite MDPs) are a way of mathematically formalising the RL problem: they capture the most important aspects of the problem

faced by an agent interacting with its environment to achieve a goal. We introduce the elements of this formalism: the agent-environment interface, goals and rewards, returns and episodes. Then...

2.2.1 The Agent-Environment Interface

MDPs consist firstly of the continual interaction between an agent selecting actions, and an environment responding by changing state, and presenting the new state to the agent, along with an associated scalar reward. Recall that the agent seeks to maximise this reward over time through its choice of actions.

More formally, consider a sequence of discrete time steps, $t = 1, 2, 3, \ldots$. At each time step t, the agent receives some representation of the environment's *state*, $S_t \in \mathcal{S}$, and chooses an *action*, $A_t \in \mathcal{A}$. On the next time step, the agent receives reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. These interactions repeat over time, giving rise to a *trajectory*:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

A *finite* MDP is one where the sets of states, actions and rewards are finite. In this case, the random variables S_t and R_t have well-defined discrete probability distributions which depend only on the preceding state and action. This allows us to define the *dynamics* of the MDP, a probability mass function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, as follows. For any particular values $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ of the random variables S_t and R_t, there is a probability of these values occurring at time t, given any values of the previous state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$:

$$p(s', r \mid s, a) := \mathbb{P}\left(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right).$$

A *Markov* Decision Process is one where all states satisfy the Markov property.

A state s_t of an MDP satisfies this property iff:

$$\mathbb{P}(s_{t+1}, r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = \mathbb{P}(s_{t+1} \mid s_t, a_t).$$

This implies that the immediately preceding state s_t and action a_t are sufficient statistics for predicting the next state s_{t+1} and reward r_{t+1}.

2.2.2 Goals and Rewards

The reader may have noticed that we first introduced MDPs as a formalism for an agent interacting with its environment to achieve a goal, yet have since spoken instead of maximising a reward signal $R_t \in \mathbb{R}$ over time. Our implicit assumption is the following hypothesis:

Hypothesis 1 (Reward Hypothesis)**.** *All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward). [5, p. 53]*

However, this hypothesis gives no information about how to construct such a scalar signal; only that it exists. Indeed, recent work has shown that it is far from trivial to do so; possible failure modes include negative side effects, reward hacking and unsafe exploration [2]. This is central to the topic of this dissertation—our aim is to improve the sample efficiency of one particular method of reinforcement learning when the reward signal is unknown.

2.2.3 Returns and Episodes

Having asserted that we can express the objective of reinforcement learning in terms of scalar reward, we now formally define this objective. Consider the following objective:

Definition 1 ((Future discounted) return)**.** *Let a sequence of rewards between time step $t+1$ and T (inclusive) be $R_{t+1}, R_{t+1}, \ldots, R_T$. Let $\gamma \in [0, 1]$ be a discount factor*

of future rewards. Then we define the (future discounted) return of this sequence of rewards [5, p. 57]:

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.1}$$

One reason for introducing a discount factor is because we would like this infinite sum to converge. Accordingly, we impose the condition that $\gamma < 1$ whenever the reinforcement learning task is *continuous*, that is to say, there may be an infinite number of non-zero terms in the sequence of rewards $\{R_{t+1}, R_{t+2}, R_{t+3}, \dots\}$.

The other kind of task is called *episodic*. Here, interactions between the agent and environment occur in well-defined subsequences, each of which ends in a special *terminal state*. The environment then resets to a starting state, which may be fixed or sampled from a distribution. To adapt the definition in (2.1) to this case, we introduce the convention that zero reward is given after reaching the terminal state. This is because we typically analyse such tasks by considering a single episode—either because we care about that episode in particular, or something that holds across all episodes [5, p. 57]. Observe that summing to infinity in (2.1) is then identical to summing over the episode, and that the sum is well-defined regardless of the discount factor γ.

2.2.4 Policies and Value Functions

Policy determines the behaviour of the agent. Formally, a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ defines a probability distribution over actions, given a state. That is to say, $\pi(a \mid s)$ is the probability of selecting action a if an agent is following policy π and in state s.

The *state-value function* $v_\pi : \mathcal{S} \mapsto \mathbb{R}$ for a policy π gives the expected return of starting in a state and following that policy. More formally,

Definition 2 (State-value function)**.** *Let π be a policy and $s \in \mathcal{S}$ be any state. We write $\mathbb{E}_\pi[.]$ to denote the expected value of the random variable G_t as defined in*

(2.1). Then the state-value function (or simply, value function) for policy π is:

$$v_\pi(s) := \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]. \tag{2.2}$$

The *action-value function* $q_\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ for a policy π is defined similarly. It gives the expected return of starting in a state, taking a given action, and following policy π thereafter.

Definition 3 (Action-value function)**.** *Let π be a policy, $s \in \mathcal{S}$ be any state and $a \in \mathcal{A}$ any action. Then the action-value function (or, Q-function) for policy π is:*

$$q_\pi(s, a) := \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \tag{2.3}$$

2.2.5 Optimal Policies and Optimal Value Functions

The problem of Reinforcement Learning is thus to find an optimal policy.

All optimal policies share the same value functions. We call these the *optimal state-value function*, v_*, and the *optimal action-value function* q_*:

Definition 4 (Optimal state-value function (from [5, p. 62]))**.**

$$v_* := \max_\pi v_\pi(s) \quad \forall s \in \mathcal{S}.$$

Definition 5 (Optimal action-value function (from [5, p. 63]))**.**

$$q_* := \max_\pi q_\pi(s, a) \quad \forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}.$$

There is a simple connection between optimal Q-function and optimal policy that will be used in Section 2.3:

Claim 1. *If an agent has q_*, then acting according to the optimal policy when in some state s is as simple as finding the action a that maximises $q_*(s, a)$ [5, p. 64].*

2.2.6 Bellman Equations

These value functions obey special recursive relationships called Bellman equations. The equations are proved by formalising the simple idea that the value of being in a state is the expected reward of that state, plus the value of the next state you move to. Each of the four value functions defined in Sections 2.2.4 and 2.2.5 satisfy slightly different equations. We prove the Bellman equation for the value function and state the remaining three for completeness.

Proposition 1 (Bellman equation for v_π [5, p. 59])**.** *Let π be a policy, p the dynamics of an MDP, γ a discount factor and v_π a state-value function. Then:*

$$v_\pi(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi(.|s)\\ s',r\sim p(.,.|s,a)}} \left[r + \gamma v_\pi(s')\right] \tag{2.4}$$

Proof.

$$\begin{aligned}
v_\pi(s) &:= \mathbb{E}_\pi \left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1}\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}} p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \\
&= \mathop{\mathbb{E}}_{\substack{a\sim\pi(.|s)\\ s',r\sim p(.,.|s,a)}} \left[r + \gamma v_\pi(s')\right]
\end{aligned}$$

□

Proposition 2 (Bellman equation for q_π)**.** *Let π be a policy, p the dynamics of an*

MDP, γ a discount factor and q_π an action-value function. Then:

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{s',r\sim p(.,.|s,a)}\left[r + \gamma \mathop{\mathbb{E}}_{a'\sim\pi(.|s')}\left[q_\pi(s',a')\right]\right] \tag{2.5}$$

Proposition 3 (Bellman equation for v_* [5, p. 63])**.** *Let p be the dynamics of an MDP, γ a discount factor and v_* an optimal value function. Then:*

$$v_*(s) = \max_{a\in\mathcal{A}} \mathop{\mathbb{E}}_{s',r\sim p(.,.|s,a)}\left[r + \gamma v_*(s')\right] \tag{2.6}$$

Proposition 4 (Bellman equation for q_* [5, p. 63])**.** *Let p be the dynamics of an MDP, γ a discount factor and q_* an optimal Q-function. Then:*

$$q_*(s,a) = \mathop{\mathbb{E}}_{s',r\sim p(.,.|s,a)}\left[r + \gamma \max_{a'\in\mathcal{A}}\left[q_*(s',a')\right]\right] \tag{2.7}$$

2.3 Reinforcement Learning Solution Methods

One method of solving the reinforcement learning problem is to explicitly solve a set of Bellman optimality equations. For example, in a finite MDP with n states and m actions, the Bellman equations for q_* are a set of $n \cdot m$ equations in $n \cdot m$ unknowns[1]. Given the dynamics p of the MDP, standard techniques for solving systems of equations can be applied. Then, via Claim (1), the agent has an optimal policy [5, p. 64].

However, in reality, we rarely have access to p, or sufficient computational resources to solve this system of equations exactly [5, p. 66]. Thus, the literature on RL solution methods focuses on finding approximate solutions.

In this section, we give a brief and incomplete taxonomy of RL solution methods, most of which give approximate solutions. Then, we focus on one such method, the deep Q-network, which is important for the rest of this thesis.

[1] This assumes that the agent can take any action in any state.

2.3.1 Taxonomy of RL Solution Methods

The following taxonomy draws on those given in [5] and [1].

- RL Solution Methods
  - Tabular Solution Methods
    - Dynamic Programming
      - Policy Iteration
      - Value Iteration
    - Monte Carlo Methods
      - MC Prediction
      - MC Control
    - Temporal-Difference Learning
      - Sarsa
      - Q-Learning
  - Approximate Solution Methods
    - Deep RL
      - Model-Free
        - Policy Optimisation
          - Policy Gradient
          - A2C / A3C
          - PPO
          - TPRO
        - Q-Learning
          - DQN
          - C51
        - Policy Optimisation + Q-Learning
          - DDPG
          - TD3
          - SAC
      - Model-Based
        - Given the Model
          - AlphaZero
        - Learn the Model
          - I2A

2.3.2 Deep Q-network

The deep Q-network (DQN) agent approximates q_* using a deep convolutional neural network $Q(s, a; \theta)$ as the function approximator [4]. θ are the parameters (weights) of the neural network, which is called a Q-network. It can be trained by performing gradient descent on the parameters θ_i at iteration i to reduce the mean-squared error between $Q(s, a; \theta_i)$ and the Bellman equation for $q_*(s, a)$, given in Proposition (4). Since we do not have access to the true value of this target, we instead use

approximate target values $y = r + max_{a'} Q(s', a'; \theta_i^-)$, with θ_i^- some previous network parameters.

One could then perform standard gradient descent on this loss function using the experience collected by the agent $\langle (s_t, a_t, r_{t+1}, s_{t+1}) \rangle_{t=0}^{T}$ as data, as one would do in the supervised learning setting. However, there are three important differences in the reinforcement learning setting: correlations both (i) in the data set, and (ii) between $Q(s, a; \theta_i)$ and the targets. Furthermore, (iii) updates to Q may change the policy and thus change the data distribution. This leads to instability in training. To address this, the authors propose two algorithmic innovations. Firstly, instead of training on experience in the order that it is collected, the agent maintains a buffer of experience $D_t = \{e_1, e_2, \ldots, e_t\}$ where $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$. When making learning updates, drawing minibatches uniformly at random from this buffer breaks correlations in the experience sequence and smooths over changes in the data distribution, alleviating problems (i) and (iii). This is termed *experience replay*. Secondly, to reduce correlations between Q and the targets and alleviate problem (ii), the approximate target values are updated to match the parameters Q only every C steps for some hyperparameter $C > 1$.

With these changes, we arrive at a loss function $\ell_i(\theta_i)$ for each learning update i:

$$
\begin{aligned}
\ell_i(\theta_i) &= \mathbb{E}_{s,a,r}\left[\left(\mathbb{E}_{s'}\left[y \mid s, a\right] - Q(s, a; \theta_i)\right)^2\right] \\
&= \mathbb{E}_{s,a,r}\left[\mathbb{E}_{s'}\left[y - Q(s, a; \theta_i) \mid s, a\right]^2\right] \\
&= \mathbb{E}_{s,a,r}\left[\mathbb{E}_{s'}\left[\left(y - Q(s, a; \theta_i)\right)^2 \mid s, a\right] - \mathbb{V}ar_{s'}\left[y - Q(s, a; \theta_i)\right) \mid s, a\right]\right] \\
&= \mathbb{E}_{s,a,r,s'}\left[\left(y - Q(s, a; \theta_i)\right)^2\right] - \mathbb{E}_{s,a,r}\left[\mathbb{V}ar_{s'}\left[y\right]\right]
\end{aligned}
$$

where the expectations and variances are with respect to samples from the experience replay. This loss function is then optimized by stochastic gradient descent with respect to the network parameters θ_i. Note that the final term is independent of these parameters, so we can ignore it. Finally, the authors found that stability is

improved by clipping the error term $y - Q(s, a; \theta_i)$ to be between -1 and 1.

We summarise this training procedure in Algorithm 1.

Algorithm 1 Deep Q-learning with experience replay.

1: Initialise replay memory D to capacity N
2: Initialise neural network Q with random weights θ as approximate optimal action-value function
3: Initialise neural network $\hat{Q}$ with identical weights $\theta^- = \theta$ as approximate target action-value function
4: Reset environment to starting state s_0
5: **for** $t = 0, \ldots, T$ **do**
6: With probability ϵ execute random action a_t
7: otherwise execute action $a_t = \arg\max_a Q(s_t, a; \theta)$
8: Observe next state and corresponding reward $s_{t+1}, r_{t+1} \sim p(., . \mid s_t, a_t)$
9: Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D_t
10: Randomly sample minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1}) \sim D_t$
11: Set $y_j = \begin{cases} r_{j+1} \text{ if episode terminates at step } j+1 \\ r_{j+1} + \gamma \max_{a'} \hat{Q}(s_{j+1}, a_j; \theta^-) \text{ otherwise} \end{cases}$
12: Do gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$ w.r.t network parameters θ
13: Every C steps set $\theta^- = \theta$
14: **end for**

Note that line 11 assumes we are training DQN to perform an episodic task, hence the first case which follows the convention given in Section 2.2.3 whereby zero reward is given for all states after the terminal state. If the task were instead continuing, line 11 would simply be $y_j = r_{j+1} + \gamma \max_{a'} \hat{Q}(s_{j+1}, a_j; \theta^-)$

2.4 Reinforcement Learning from Unknown Reward Functions

For many domains in which we might want to use RL, states of the environment are not inherently associated with rewards. Thus, we have the additional task of specifying a reward function, which maps states to rewards. Argue the case that directly specifying a reward function is hard, in order to motivate the next section: Reward Learning

2.4.1 Reward Learning with Handcrafted Feature Transformations

Active Preference-Based Learning of Reward Functions

Batch Active Preference-Based Learning of Reward Functions

DemPref

2.4.2 Reward Learning in Deep RL

Chapter 3

Uncertainty in Deep Learning

Standard deep learning models output point estimates. For example, a model trained to classify pictures of dogs according to their breed takes a picture of a dog and outputs its predicted breed. However, what will the model do if it is given a picture of a cat? [3].

We probably want the model to be able to recognise that this is an out of distribution example, and request more training data, or simply say that it doesn't know the answer. However, since standard deep learning models output only point estimates, the model will just go ahead and classify the cat as some breed of dog, just as confidently as any other input.

Thus, the field of Bayesian Deep Learning aims to equip neural networks with the ability to output a point estimate along with its uncertainty in that estimate. Historically, many of the attempts to do so were not very practical. For example, one algorithm, Bayes by Backprop, requires doubling the number of model parameters, making training more computationally expensive, and is very sensitive to hyperparameter tuning. However, recent techniques allow almost any network trained with a stochastic regularisation technique, such as dropout, to, given an input, obtain a predictive mean and variance (uncertainty), without any complicated augmentation to the network [3, p. 15].

3.1 Bayesian Neural Networks

3.2 Model Uncertainty in BNNs

Chapter 4

Active Learning

4.1 Acquisition Functions

4.1.1 Max Entropy

4.1.2 Variation Ratios

4.1.3 Mean STD

4.1.4 BALD

4.2 Applying Active Learning to RL without a reward function

Discussion of previous work.

4.2.1 APRIL

4.2.2 Active Preference-Based Learning of Reward Functions with handcrafted feature transformations

Active Preference-Based Learning of Reward Functions

Batch Active Preference-Based Learning of Reward Functions

DemPref

4.2.3 Deep RL from Human Preferences

Chapter 5

Method

5.1 Training Protocol

5.2 Acquisition Functions

5.3 Uncertainty Estimates

5.4 Implementation Details

Chapter 6

Experimental Details

We used the environments provided by OpenAI Gym to run our experiments.

6.1 CartPole-v0

Chapter 7

Results

7.1 CartPole-v0

Chapter 8

Conclusions

8.1 Summary

8.2 Evaluation

8.3 Future Work

References

[1] Joshua Achiam. Spinning Up in Deep RL, 2019.

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete Problems in AI Safety. pages 1–29, 2016.

[3] Yarin Gal. Uncertainty in Deep Learning. *Phd Thesis*, 1(1):1–11, 2017.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[5] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (2nd Edition, in preparation)*. 2018.

Appendices

Appendix A

Some Appendix Material