Exercise Set 1 - Reinforcement Learning
Chapter 1, 2 - MDPs and Dynamic Programming

Instructions

This is the first exercise booklet for Reinforcement Learning. It covers both ungraded exercises to practice at home or during the tutorial sessions as well as graded homework exercises and graded coding assignments. The graded assignments are clearly marked. Please note the following:

- Make sure you deliver answers in a clear and structured format. LaTeXhas our preference. Messy handwritten answers will not be graded.
- Pre-pend the name of your TA to the file name you hand in and remember to put your name and student ID on the submission;
- The deadline for this assignment is **September 13th 2023 at 19:00** and will cover the material of chapters 1-2. All questions marked 'Homework' in this booklet need to be handed in on Canvas. The coding assignments need to be handed in separately through the Codegrade platform integrated on canvas.

Contents

Chapter 0: Prerequisites

0.1 Multi-armed Bandits - Introduction Lab

1. Download the notebook *RL_WC1_bandit.ipynb* from Canvas and follow the instructions.

0.2 Prior knowledge self-test

0.2.1 Linear algebra and multivariable derivatives.

Consider the following matrices and vectors:

$$A = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \quad c = \begin{pmatrix} y - x^2 \\ \frac{\ln x}{y} \end{pmatrix} \quad d = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \quad e = \begin{pmatrix} x \\ y \end{pmatrix}$$

1. Compute AB, AB^T, and $d^T B d$.
2. Find the inverses of A and B.
3. Compute $\frac{\partial c}{\partial x}$ and $\frac{\partial c}{\partial e}$.
4. Consider the function $f(\boldsymbol{x}) = \sum_i^N i x_i$, which maps an N-dimensional vector of real numbers $\boldsymbol{x}$ to a real number. Find an expression for $\frac{\partial f}{\partial \boldsymbol{x}}$ in terms of integers 1 to N.

0.2.2 Probability theory

Assume X and Y are two independent random variables, respectively with mean μ, ν, and variance σ^2, τ^2.

1. What is the expected value of $X + \alpha Y$, where α is some constant?
2. What is the variance of $X + \alpha Y$?

Suppose now that we have a training set consisting of points $(x_1, x_2, ..., x_n)$ and real values y_i associated to each point x_i. Assume that there is a function f such that $y = f(x) + \epsilon$, where ϵ is a noise vector with zero mean and variance σ^2.

Assume we are looking for a function $\hat{f}$ that minimises the squared error $(y - \hat{f}(x))^2$. Recall the bias-variance decomposition of this squared error on an unseen sample x:

$$\mathrm{E}\left[(y - \hat{f}(x))^2\right] = \left(\mathrm{Bias}[\hat{f}(x)]\right)^2 + \mathrm{Var}[\hat{f}(x)] + \sigma^2 \tag{1}$$

where

$$\mathrm{Bias}[\hat{f}(x)] = \mathrm{E}[\hat{f}(x)] - f(x) \qquad \mathrm{Var}[\hat{f}(x)] = \mathrm{E}[\hat{f}(x)^2] - \mathrm{E}[\hat{f}(x)]^2$$

and the expectation E[.] ranges over different choices of the training set $(x_1, x_2, ..., x_n; y_1, y_2, ..., y_n)$, all sampled from the same joint distribution $P(X, Y)$.

3. Explain what errors the various terms in equation (1) represent (for instance, under what circumstances is a particular term large or small?).
4. Explain why this decomposition is also known as the 'bias-variance trade-off'.

0.2.3 OLS, linear projection, and gradient descent.

Suppose now that we have a training set $\boldsymbol{X}$ consisting of n vectors (datapoints) of size m (features). Associated to this we have an n-dimensional vector of real values $\boldsymbol{y}$. Suppose we want to regress a linear model to this data using ordinary least-squares (OLS). That is, we fit linear model $f_\beta(\boldsymbol{X}) = \boldsymbol{X}\beta$, such that we minimise $||\boldsymbol{y} - f_\beta(\boldsymbol{X})||_2^2$ over the parameters β.

1. What is the dimensionality of the parameter vector β?
2. Show by differentiation that the OLS estimator $\hat{\beta}$ equals $(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$.

Another method for deriving the OLS estimator is by thinking geometrically. Imagine $\boldsymbol{y}$ and $\boldsymbol{X}\beta$ as vectors in an n-dimensional vector space. Further note that the regressors $\boldsymbol{X}\beta$ actually span an m-dimensional subspace of this larger vector space (the column space of $\boldsymbol{X}$), see Figure 1.

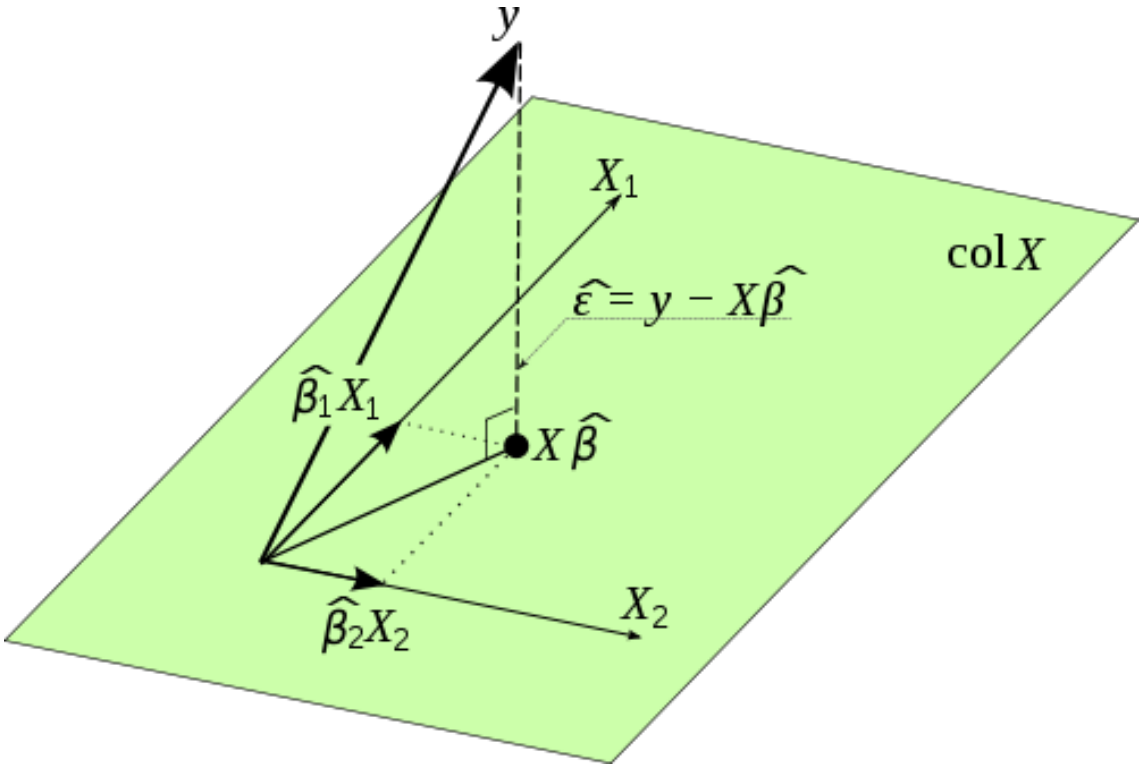

Figure 1: Geometric representation of OLS, from Wikipedia. The X_i are columns of $\boldsymbol{X}$.

3. Define as $\boldsymbol{\epsilon}_\beta$ the residual vector $(\boldsymbol{y} - \boldsymbol{X}\beta)$. Argue that minimising the L_2 norm of this vector over β (as we do in OLS) is equivalent to choosing β such that $\boldsymbol{X}\beta = P(\boldsymbol{y})$, where $P(.)$ orthogonally projects $\boldsymbol{y}$ onto the linear subspace spanned by the regressors.
4. Argue that this is equivalent to setting $\boldsymbol{X}^T\boldsymbol{\epsilon}_\beta = 0$.
5. Show that from this also follows that $\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$.

When our models get more complicated, we quickly lose the ability to optimise them analytically. Instead, we use numerical optimisation methods such as gradient descent. Although we do not need gradient descent to fit the model described above, we can still do so.

6. In gradient descent we minimise a loss function $L_\beta(\boldsymbol{y}, \boldsymbol{X})$ over the parameter values β. What is the loss function corresponding to the OLS description from before?

7. Assume we use a learning rate α. Write the update rule for a single gradient descent step on our parameters β.

Chapter 1: Introduction & MDPs

1.1 Introduction

1. Explain what is meant by the 'curse of dimensionality'.

2. Suppose you are trying to design a predator agent that can learn to catch a randomly moving prey on a 5×5 ***toroidal*** grid. You have been given the (x, y)-coordinates of the predator and the (x, y)-coordinates of the prey to use as the state.

   (a) How many possible states are there in this naive approach?

   (b) There is a way of reducing the state space considerably a priori. Write down how you would adapt the given state representation to reduce the size of the state space. Note that you only care about a representation that you can use to solve the problem.

   (c) How many possible states are there now?

   (d) What is the advantage of doing this?

   (e) Consider the Tic-Tac-Toe example in Chapter 1.5 of the book. Here, too, we can exploit certain properties of the problem to reduce the size of the state space. Give an example of a property you can exploit.

3. Suppose you want to implement a Reinforcement Learning agent that learns to play Tic-Tac-Toe, as outlined in Chapter 1 of the book.

   (a) Which agent do you think would learn a better policy in the end: a greedy agent that always chooses the action it currently believes is best, or a non-greedy agent that sometimes tries new actions? Why?

4. Assume we start with an exploration rate of ϵ, meaning that whenever the agent chooses an action, it has a probability of ϵ to pick an action at random, and a probability of $1 - \epsilon$ to pick the greedy action. If we assume the environment has been sufficiently explored, we may want to reduce the amount of exploration after some time.

   (a) Write down how you would do this.

   (b) Does your method work if the opponent changes strategies? Why/why not? If not, provide suggestions on a heuristic that can adapt to changes in the opponent's strategy.

1.2 Exploration

1. In ϵ-greedy action-selection for the case of n actions, what is the probability of selecting the greedy action?

2. Consider a 3-armed bandit problem with actions $1, 2, 3$. If we use ϵ-greedy action-selection, initialization at 0, and ***sample-average*** action-value estimates, which of the following sequence of actions are certain to be the result of exploration? $A_0 = 1, R_1 = -1, A_1 = 2, R_2 = 1, A_2 = 2, R_3 = -2, A_3 = 2, R_4 = 2, A_4 = 3, R_5 = 1$.

3. You are trying to find the optimal policy for a two-armed bandit. You try two approaches: in the pessimistic approach, you initialize all action-values at -5, and in the optimistic approach you initialize all action-values at +5. One arm gives a reward of +1, one arm

gives a reward of -1. Using a greedy policy to choose actions, compute the resulting Q-values for both actions after three interactions with the environment. In case of a tie between two Q-values, break the tie at random. *Note*: the initialization is *not* a sample.

4. Which initialization leads to a higher (undiscounted) return? What if you had broken the tie differently?

5. Which initialization leads to a better estimation of the Q-values?

6. Explain why one of the two initialization methods is better for exploration.

1.3 Markov Decision Processes

1. (a) For the first four examples outlined in Section 1.2 of the book, describe the state space, action space and reward signal.

   (b) Come up with an example of your own that you might model as an MDP. State the action space, state space and reward signal.

   (c) Come up with an example for a problem that you might have trouble solving with an MDP. Why doesn't this fit the framework?

   (d) In mazes, the agent's position is often seen as the state. However, the agent's position alone in not always a sufficient description. Come up with an example where the state consists of the agent's location and one or more other variables.

   (e) Consider the example in exercise 3.3 of the book. Why might you choose to view the actions as handling the accelerator, brake and steering wheel? What is the disadvantage of doing this?

   (f) Why might you choose to view the actions as choosing where to drive? What is the disadvantage of doing this?

   (g) Can you think of some way to combine both approaches?

2. (a) Eq. 3.8 in the book gives the discounted return for the continuing case. Write down the formula for the discounted return in the episodic case.

   (b) Show that $\sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$ if $0 \leq \gamma < 1$. (Hint: if you're stuck, have a look at the Wikipedia page on *geometric series*)

   (c) Consider exercise 3.7 in the book. Why is there no improvement in the agent?

   (d) How would adding a discount factor of $\gamma < 1$ help solve this problem?

   (e) How might changing the reward function help solve this problem?

Chapter 2: Dynamic Programming

2.1 Dynamic programming

The figure below describes an MDP consisting of 3 states. The figure indicates the actions available at each state (which are deterministic apart from for state 2). Purple boxes indicate the reward received after taking a certain action with the arrow indicating the subsequent state. State 3 is the terminal state.

1. Perform Value iteration for all states until the change in values between iterations < 10 for all states. Use a discount factor of 1 for this question. Initialise values at 0 for all states.

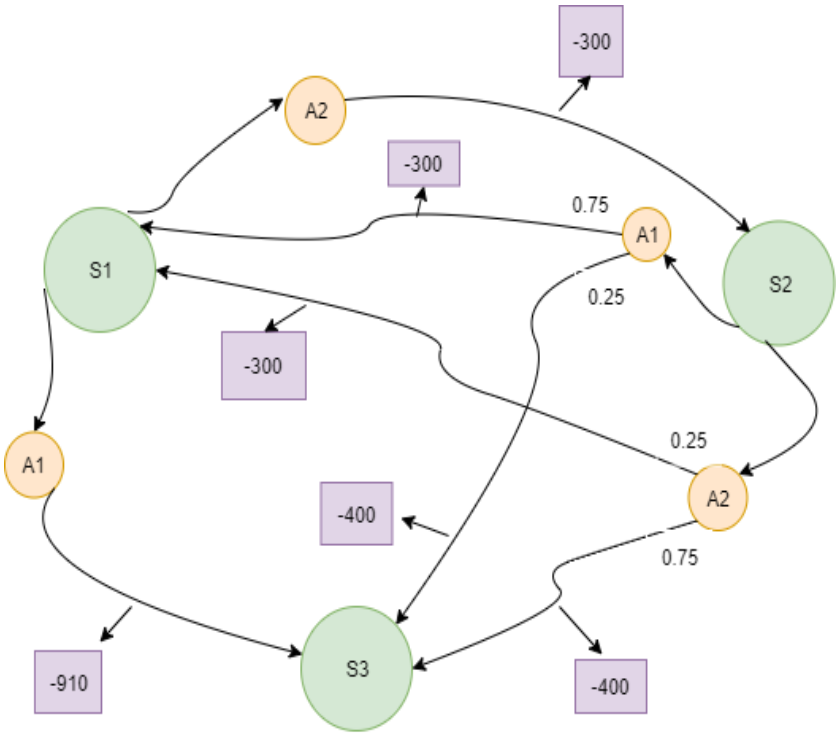

Figure 2: Representation of a Markov Decision Process

2.2 Contraction Mappings & Convergence of Bellman Optimality Operator

This exercise is intended to be done during the tutorial session. Do not panic if you find the exercise very hard if attempting it by yourself.

It is important to understand the convergence behavior of reinforcement learning methods. In one iteration of the value iteration algorithm, an 'input' value function v_n is mapped to an 'output' value function v_{n+1}. This mapping is called the *Bellman optimality operator* B^*, which

is given by

$$(B^*V)(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s', s, a) V(s') \right], \tag{2}$$

where V be the value function which gives the value per state, and $R(s,a)$ the expected immediate reward for action a in state s and $p(s', s, a)$ the probability of transitioning to state s' after taking action a in state s.

To understand convergence of value iteration, we can look at the behavior of this operator. In this exercise, we will prove that this operator is a contraction mapping. This is very helpful, as contraction mappings are known to converge.

A contraction mapping is a mapping $T : X \to X$ (on metric space $(X, ||\cdot||)$) if there exists a constant $0 \leq c < 1$ such that

$$||T(x) - T(y)|| \leq c||x - y|| \quad \forall x \in X, \forall y \in X \tag{3}$$

Thus, after applying the operator, the points are closer together than before by at least the factor c. Informally, this implies that no matter how two points x and y are initialized, repeated applications of the operator will push them closer and closer together to a single fixed point that is mapped to itself by the operator.

Formally, this results is stated in the following Theorem, from Csaba Szepesvari's *Algorithms for Reinforcement Learning*:

Theorem 1. (Banach's Fixed Point Theorem). Let V be a Banach space and $T : V \to V$ be a contraction mapping. Then T has a unique fixed point. Further, for any $v_0 \in V$, if $v_{n+1} = Tv_n$, then $v_n \to_{||\cdot||} v$, where v is the unique fixed point of T and the convergence is geometric:

$$||v_n - v|| \leq \gamma^n ||v_0 - v|| \tag{4}$$

(Note: a Banach space is a special kind of metric space. The precise definition is not relevant for this question).

1. Here we'll attempt to better understand the fixed point of a contraction mapping:
   (a) Consider the contraction mapping $T[x] := 1 + \frac{1}{3}x$. Find the fixed point of this mapping.
   (b) Consider the contraction mapping $(T[f])(s) := \frac{-1}{2} f(s) + s^2$. Find the fixed point of this mapping.
2. Here, we'll show step by step that B^* is a contraction mapping.
   (a) In the Bellman optimality operator (2), the max plays an important role. Show that

   $$|\max_z f(z) - \max_z h(z)| \leq \max_z |f(z) - h(z)| \tag{5}$$

   for arbitrary $f(z)$ and $h(z)$.
   You can use a proof by cases based on the sign of $\max_z f(z) - \max_z h(z)$.

   Consider an MDP with finite state space $\mathcal{S}$, finite action space $\mathcal{A}$, and discount factor γ and the Bellman optimality operator as given in (2)

   In our case, the metric space is $(\mathcal{B}(\mathcal{S}), ||\cdot||_\infty)$, where $\mathcal{B}(\mathcal{S})$ is the space of bounded functions on $\mathcal{S}$. $\mathcal{B}(\mathcal{S}) = \{V : \mathcal{S} \to \mathbb{R} : ||V||_\infty < +\infty\}$. $(\mathcal{B}(\mathcal{S}), ||\cdot||_\infty)$ is a normed vector space. Thus, the distance between value functions is given by the supremum norm $||V_1 - V_2||_\infty = \max_{s \in \mathcal{S}} |V_1(s) - V_2(s)|$, or the largest difference between state values.

2. (b) Show that for any $\tilde{s}$,

$$\max_a \left| \gamma \sum_{s'} p(s', \tilde{s}, a) \left(V_1(s') - V_2(s') \right) \right| \leq \gamma \max_s |V_1(s) - V_2(s)|$$

(c) Using the results from (a) and (b), show that B^* is a contraction mapping in supremum norm, i.e.

$$||(B^*V_1) - (B^*V_2)||_\infty \leq c||V_1 - V_2||_\infty \tag{6}$$

(d) Why is this an important result?

2.3 * Exam question: Dynamic programming

In these exercise booklets, you'll find exercises like this marked with ''. These exercises have been taken from previous exams (perhaps lightly edited) and can require a bit more insight. They should give you a good idea of the level of exam questions.*

1. Judge the following statements with True/False and add a short explanation

   (a) Value Iteration tends to converge to better policies than Policy Iteration.

   (b) Value Iteration combines one sweep (one backup per state) of Policy Evaluation and one sweep of Policy Improvement in each sweep.

2. We consider the optimal value function in state s, $v_*(s) = \max_\pi v_\pi(s)$, where v_π is the value function for policy π. The Bellman optimality equation at this state is given by:

$$v_*(s) = \max_a \sum_{s',r} p\left(s', r \mid a, s\right) \left[r + \gamma v_*\left(s'\right) \right],$$

   where $p(s', r|a, s)$ represents the transition probabilities to state s' with reward r, given that action a was taken in state s, and γ is a discount factor. Explain why this equation holds when policy iteration stabilizes (you may assume a tabular setting).

2.4 Homework: Coding Assignment - Dynamic Programming

1. Download the *RLLab1_DP.zip* from Canvas and follow the instructions.

   Important: if you are working with a partner, in the codegra.de tool, first join a group together before handing in any solution. *If you already hand in a solution before your group is complete, your partner will not be able to join your group.* Note that codegra.de has separate groups from the homework assignment on Canvas! Thus, you need to join a group together both on Canvas and on Codegra.de!

2. In the lab you implemented the value iteration and policy iteration algorithm.

   - For which of these algorithms do you expect a single iteration to run faster? Briefly explain why. (For policy iteration, one iteration includes the policy evaluation and policy improvement steps).
   - Which of the algorithms do you expect to take fewer total iterations? Briefly explain why.

2.5 Homework: Dynamic Programming

1. Write the value, $v^\pi(s)$, of a state s under policy π, in terms of π and $q^\pi(s, a)$. Write down both the stochastic and the deterministic policy case.

2. The Value Iteration update, given in the book in Eq. 4.10 can also be rewritten in terms of Q-values. Give the Q-value Iteration update.

3. In Policy Iteration, we first evaluate the policy by computing its value function, and then update it using a Policy Improvement step. You will now change the Policy Evaluation algorithm on page 74 of RL:AI to compute action values. Give the new policy evaluation update in terms of $Q^\pi(s, a)$ instead of $V^\pi(s)$. That is, write an update for $Q^\pi(s, a)$ in terms of the current *action-value* function. Your answer should not contain a state-value function.

4. Now change the Policy Improvement step of the algorithm on page 80 of RL:AI in terms of $Q^\pi(s, a)$ instead of $V^\pi(s)$.

5. You might have noticed the policy evaluation step on page 80 is different from the separate algorithm on page 75. What is the difference, and why do you think this difference exists?