CS 748 (Spring 2021): Weekly Quizzes

Instructor: Shivaram Kalyanakrishnan

May 5, 2021

Note. Provide justifications/calculations/steps along with each answer to illustrate how you arrived at the answer. State your arguments clearly, in logical sequence. You will not receive credit for giving an answer without sufficient explanation.

Submission. You can either write down your answer by hand or type it in. Be sure to mention your roll number. Upload to Moodle as a pdf file.

Week 11

Question. This week's question requires coding, and is in two parts. You should be able to complete Part a after watching this week's lecture; Part b is based on the lecture from Week 12. You only have to turn in your submission at the end of Week 12.

You are required to code up procedures related to POMDP planning for the grid world task covered in the Week 11 lecture (for a full specification see slides 1 and 9). There is no need to write generic code that works for all POMDPs; you will find it easier to hard-code equations specific to the task at hand.

a. Write a program `beliefUpdate.sh` that takes in a belief state p, q, an action a, and an observation o as input. It must output the next belief state b' as a 4-dimensional vector. We adopt the convention that for any *input* belief state b, the mass on the goal state, 3, is zero; $b(1)$ is denoted p and $b(2)$ denoted q. You can assume $p, q, p + q \in [0, 1]$, which means that $b(4) = 1 - p - q$ is also a valid probability. The actions are **L** and **R**, and the observations are GOAL and NO-GOAL. The example worked out in class should execute as follows (minor differences in output due to floating point issues are okay).

Command: `./beliefUpdate.sh 0.333333 0.333333 R NO-GOAL`
Output: `0.1 0.45 0 0.45`

Here is another check.

Command: `./beliefUpdate.sh 0.5 0.25 R GOAL`
Output: `0 0 1.0 0`

We will test the correctness of your update by calling `beliefUpdate.sh` with different inputs p, q, a, o. [3 marks]

b. Write a program `valueIteration.sh` that takes in depth $t \in \{1, 2, 3\}$ and a belief state p, q as input and computes as output the maximum t-step value achievable from the belief state. The input depth t will be at most 3; you can implement the algorithm by enumerating all possible policy trees of depth t (no need for parsimony); evaluating each of them to obtain its alpha vector; and computing the maximum dot product of the alpha vectors with the input belief state.

   If GOAL is observed, there is no need for a subsequent action; we leave it to you whether to introduce a dummy action to keep the tree syntax consistent or to handle the goal state differently. In any case, make sure that values get computed correctly. We retain the convention on the input belief state b ($b(1) = p, b(2) = q, b(3) = 0, b(4) = 1 - p - q$). Since there is no discounting, notice that the value of every 1-step policy tree (that is, every action) should be -1. Here is one example of expected input-output behaviour (the input sequence is t, p, q).

   Command: `./valueIteration.sh 1 0.5 0.25`
   Output: `-1.0`

   We will test your code by varying the inputs. [5 marks]

Internally use any programming language of your choice, but create the shell scripts with the given names as wrappers. Make sure your code executes on the docker used for CS 747 last semester.

Week 10

Question. This week's question encourages you to explore and find out more for yourself about uses of crowdsourcing beyond the ESP game covered in the week's lecture.

a. What is a "prediction market"? Give an example of one, and explain how its setup contrasts with that of the ESP game. It would be a good idea to start from the Wikipedia page (`https://en.wikipedia.org/wiki/Prediction_market`) and thereafter search for more information. [2 marks]

b. What was the "DARPA Network Challenge"? What was the approach adopted by the winning team? You will get most of the information you need on the associated Wikipedia page (`https://en.wikipedia.org/wiki/DARPA_Network_Challenge`). [2 marks]

An ideal answer will take 6–10 lines for each of the two parts; don't write pages and pages. Write in your own words; do not reproduce snippets from existing sources. Cite all your sources in your response.

Solution.
a. A prediction market is a surprisingly effective approach, based on crowdsourcing, to predict the occurrence of a future event—usually one that is hard to predict with conventional models. Although prediction markets are most commonly associated with the financial world, they have also been used, for example, to predict the winner of a forthcoming election. Suppose there are two candidates A and B in the election, and one would like to predict the winner a few weeks ahead of time. The prediction market would allow for participants to trade options of the type "A wins" and "B wins", with the incentive that a "B wins" option can be redeemed for something of value (such as money or fame) should candidate B eventually win (and similarly, "A wins" options redeemable if A wins). As a consequence, the price at which trades occur ahead of the actual event can usually yield a good estimate of the "probability" that either candidate will win—usually increasing in prediction accuracy in the run up to the event. In effect, the trade price implicitly aggregates the knowledge of the participants into a single prediction, which in some cases is significantly more accurate than conventional predictors.

Whereas the ESP game is primarily played for fun, prediction markets can attract domain experts who can profit by applying their knowledge. A prediction game is an ongoing process with relevant information (such as prices) constantly being updated and made available to participants; the game ends once an outcome (being predicted) is registered in reality. By contrast, each session in ESP is much shorter, with no live information given to the pair of players until the session ends. ESP can be played virtually for ever, since no player is likely to exhaust the set of all images available on the Internet.

b. The "DARPA Network Challenge" was a competition conducted by DARPA in 2009 to assess the ability of the general population to collaborate and solve an information-gathering problem spread across geography, yet limited by a short time horizon. The organisation announced, about a month in advance, that it would place ten large balloons in locations around the US on a particular day, and keep them in place for roughly half a day. The first registered team to submit accurate locations of all ten balloons would be eligible for prize money.

The prize was indeed claimed by a team from MIT that submitted an accurate list of locations within the stipulated time period. The team had used social media to announce its participation, along with an incentive structure for people to report their sightings to the team. People could join the team and also invite others to join the team. If a person X reported an accurate location for one of the balloons, not only was X to be given a share of the prize money, so were those in the chain of invitations leading to X (with diminishing fractions). The scheme therefore incentivised people to join the team, and also to invite others. Multiple measures were put in place to determine the legitimacy of submissions—fake submissions were not a major hindrance.

Week 9

Question. Read the first three sections of the paper by Rosin and Belew (1997), which is linked from the course page . In the context of competitive coevolution, what is an "arms race"? Describe in 2–3 lines each the three main technical ideas proposed by the authors, namely competitive fitness sharing, shared sampling, and hall of fame. [4 marks]

Solution. Competitive coevolution occurs in a world that is made up of two evolving populations: one denoted a *host* population $\mathcal{H}$ and the other a *parasite* population $\mathcal{P}$. The populations compete with each other: abstractly, any host $h \in \mathcal{H}$ and parasite $p \in \mathcal{P}$ can be treated as players in a zero-sum game. Hence, increasing the fitness of the hosts (their success probability in the game) is equivalent to decreasing the fitness of the parasites: however, both populations usually only aim to increase their own fitness.

An arms race emerges, if for example, the hosts evolve some aspect of behaviour b_1^h that gives them an advantage over the current parasite population. In nature, in a world comprised of zebra and lion populations, a genetic change that allows zebras to run faster might correspond to such an advantage. With evolution, however, the parasites are likely to evolve a counter-skill b_1^p to mitigate against b_1^h: for example, lions, too, could increase their speeds. Next, the hosts evolve b_2^h in response to b_1^p, and the process continues: a *race* to procure arms that are more powerful than the opponents'.

Given $\mathcal{P}$, how to define the fitness of each host $h \in \mathcal{H}$? The most direct approach would be to count the total number of wins of h against the members of $\mathcal{P}$. However, selection based on this aggregate number risks losing members of $\mathcal{H}$ that, while below par in aggregate, may have the ability to beat "rarely beatable" parasites. Say p_0 is a parasite that is only beaten by host h_0, but h_0 is unsuccessful against most other parasites. Should h_0 be retained in subsequent iterates of the host population? Competitive fitness sharing implements the view that indeed the special ability of h_0 is worth preserving for the future; it does so by weighting each win of $h \in \mathcal{H}$ against $p \in \mathcal{P}$ by the reciprocal of the number of losses of p against $\mathcal{H}$.

While the definition afforded above by competitive fitness sharing suffices to evaluate each host $h \in \mathcal{H}$, in practice the computation would involve playing h against each $p \in \mathcal{P}$. This computational effort can be reduced by first selecting a "representative" sample $\bar{\mathcal{P}} \subseteq \mathcal{P}$ (with $|\bar{\mathcal{P}}|$ typically much smaller than $|\mathcal{P}|$), so games can be restricted to parasites in $\bar{\mathcal{P}}$. Shared sampling is a process that incrementally builds the sample $\bar{\mathcal{P}}$ by repeatedly looking for parasites that win over hosts that are not beaten (substantially) by the currently-picked set of parasites. The mathematical definition of the process borrows from that of competitive fitness sharing.

In the process of coevolution, if, say, the fitness of a host $h \in \mathcal{H}$ is defined solely based on its games with the *current* set of parasites $\mathcal{P}$, it could well happen that current champions play poorly against previous generations of parasites. The hall of fame is a simple approach to prevent populations from "having a short memory". In this approach, the winners (both from $\mathcal{H}$ and from $\mathcal{P}$) are retained and used for all future fitness evaluations—forcing future champions to play well against both current and previous winners among the opponents.

Week 8

Question. Consider an inverse RL problem to which the inputs are S, A, T, γ, and $\pi^\star$, with notation as usual. Assume that $\pi^\star$ maps S to A (in other words, is deterministic), and also that T implements a continuing task, with $\gamma < 1$. Loosely, our aim is to find a reward function R such that $\pi^\star$ is an optimal policy for MDP (S, A, T, R, γ). This question examines the possibility that $\pi^\star$ is the *unique* optimal policy for the MDP.

a. Show that if we take the reward function R to be solely dependent on the start state of a transition (as done by Ng and Russell (2000))—that is, we take $R : S \to \mathbb{R}$—there need not exist R such that $\pi^\star$ is the unique optimal policy for (S, A, T, R, γ). [2 marks]

b. By contrast, suppose rewards depend on both the start state and action in a transition—that is, we have $R : S \times A \to \mathbb{R}$. Show that there must exist R such that $\pi^\star$ is the unique optimal policy for (S, A, T, R, γ). [2 marks]

Solution.
a. Consider a single-state MDP with two or more actions. Any reward function that depends only on state will result in the the same value for all the actions (here equivalent to policies). Hence, there can be no unique optimal policy for the MDP with this sort of reward function.

b. Define $R : S \times A \to \mathbb{R}$ by

$$R(s, a) = \begin{cases} 1 & \pi^\star(s) = a, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, we give a 1-reward to the action taken by $\pi^\star$ from each state, and a 0-reward to actions not taken by $\pi^\star$. It is easy to see that for each $s \in S$, $V_R^{\pi^\star}(s) = \frac{1}{1-\gamma}$. On there other hand, any policy π that differs from $\pi^\star$ in at least one state s must have

$$\begin{aligned}
V_R^{\pi}(s) &= \mathbb{E}_\pi[r^0 + \gamma r^1 + \gamma^2 r^2 + \ldots | s^0 = s] \\
&= \gamma \mathbb{E}_\pi[r^1 + \gamma r^2 + \gamma^2 r^3 + \ldots | s^0 = s] \\
&\leq \gamma(1 + \gamma + \gamma^2 + \ldots) \\
&= \frac{\gamma}{1-\gamma} \\
&< \frac{1}{1-\gamma} \\
&= V_R^{\pi^\star}(s).
\end{aligned}$$

The function R we have constructed results in $\pi^\star$ being the unique optimal policy for (S, A, T, R, γ).

Week 7

Question. This question is about the trajectory taken by Howard's PI on MDPs with $n \geq 2$ states and 2 actions. The actions are taken as 0 and 1, and policies as n-length binary strings.

For bit $x \in \{0, 1\}$, let x^{c} denote the complement: that is, $0^{\mathrm{c}} = 1$ and $1^{\mathrm{c}} = 0$. For a policy $\pi = x_1 x_2 \ldots x_n$, let π^{c} denote the policy obtained by negating each bit in π: that is, $\pi^{\mathrm{c}} = x_1^{\mathrm{c}} x_2^{\mathrm{c}} \ldots x_n^{\mathrm{c}}$.

Let $\pi_0, \pi_1, \ldots, \pi_T$ be the sequence of policies visited on some MDP if starting with π_0 and applying Howard's PI. π_T, where $T \geq 0$, is an optimal policy that concludes the run of the algorithm. For $1 \leq i < j \leq T$, show that $\pi_i \neq \pi_j^{\mathrm{c}}$. In other words, show that no two policies visited by Howard's PI (subsequent to the initial policy π_0) can be complements of each other. You can accomplish this proof by using the policy improvement and "deprovement" theorems. [4 marks]

Solution. Consider the policy

$$\pi = x_1 x_2, \ldots, x_n$$

where $x_i \in \{0, 1\}$ for $1 \leq i \leq n$. Assume that π is not an optimal policy, and hence has $1 \leq m \leq n$ improvable states. Without loss of generality, assume that the first m states are improvable. If π is encountered by Howard's PI, the next policy visited will therefore be

$$\pi' = x_1^{\mathrm{c}} x_2^{\mathrm{c}} \ldots x_m^{\mathrm{c}} x_{m+1} x_{m+2} \ldots x_n.$$

By the policy improvement theorem, we know that $\pi' \succ \pi$. By policy deprovement, we also have that $\pi \succeq \pi''$, where

$$\pi'' = x_1 x_2 \ldots x_m x_{m+1}^{\mathrm{c}} x_{m+2}^{\mathrm{c}} \ldots x_n^{\mathrm{c}}$$

since states $m + 1$, $m + 2$, $\ldots$, n are not improvable in π. Observe that $\pi'' = (\pi')^{\mathrm{c}}$.

In other words, what we have shown above is that if π is not an optimal policy, and Howard's PI improves from π to π', then π' already dominates $(\pi')^{\mathrm{c}}$. Hence, surely $(\pi')^{\mathrm{c}}$ cannot be visited by Howard's PI subsequent to the visit to π'. It follows that if $\pi_0, \pi_1, \ldots, \pi_T$ is the sequence of policies visited by Howard's PI on some run, then no two policies from the sequence $\pi_1, \pi_2, \ldots, \pi_T$ can be complements of each other—on the contrary, if $\pi_i = \pi_j^{\mathrm{c}}$ for some $1 \leq i < j \leq T$, the argument above is violated when we consider π_i as our π'. Indeed it is possible that $\pi_1 = \pi_0^{\mathrm{c}}$: this happens if π_0 has *all* of it states to be improvable.

Week 6

Question. We aim to determine the winner of a tournament among players $1, 2, \ldots, n$. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. Each game between a pair of players results in a win for one and a loss for the other; however, the outcome is *stochastic*. Specifically, for players $i, j \in [n], i \neq j$, p_{ij} denotes the probability that i defeats j when these players face off (hence $p_{ji} = 1 - p_{ij}$). For player $i \in [n]$, the *quality* q_i is the probability that i will defeat an opponent picked uniformly at random:

$$q_i = \frac{1}{n-1} \sum_{j \in [n], j \neq i} p_{ij}.$$

The winner of the tournament is taken to be $\text{argmax}_{i \in [n]} q_i$.

Provide an algorithm L that "interacts" with any given tournament by repeatedly conducting games between pairs of players, and stops at some point to declare the winner. The entire preceding history is available at each stage to make the next decision (picking a pair of players, or declaring the winner). Assume that L will only be asked to interact with tournaments that have a unique winner. We require that for any arbitrary such tournament, L will terminate and declare the winner correctly with probability at least $1 - \delta$.

Also give an upper bound on L's expected sample complexity (number of games conducted) as a function of the win probabilities p_{ij} for $i, j \in [n], i \neq j$. There is no need for your to optimise L to bring down its sample complexity, so long as it is finite.

Give a high-level sketch justifying the claims of correctness (delivering the PAC guarantee) and of satisfying the sample complexity bound. No need for a detailed mathematical working. [5 marks]

Solution. We describe one of many possible approaches to solve the problem.

To "test" the quality of a particular player $i \in [n]$, we pick an opponent $j \in [n] \setminus \{i\}$ uniformly at random, and play i against j. We record a score of 1 if i defeats j and 0 otherwise. By this construction, observe that the expected score from a test of i is exactly q_i; in other words, we have a procedure that simulates draws of a Bernoulli random variable with mean q_i. Our task now reduces to identifying the best arm in an n-armed bandit instance, with the mean of arm $i \in [n]$ being q_i. But this problem is precisely solved by LUCB; using $\epsilon = 0$ is okay because we are guaranteed the existence of a unique winner. We can run LUCB to guarantee correctness as well as an expected sample complexity of $O(H \log \frac{H}{\delta})$ samples, where H is determined by the "gaps" between the qualities. If we assume $q_1 > q_2 \geq q_3 \geq \cdots \geq q_n$, we have

$$H = \frac{1}{(q_1 - q_2)^2} + \sum_{j=2}^{n} \frac{1}{(q_1 - q_j)^2}.$$

Week 5

Question. This week's question is based on the AlphaGo Zero program (Silver *et al.*, 2017).

a. In this program, the authors use a *single* neural network (with parameters θ) to represent both value function and policy. This choice of maintaining a combined representation is validated empirically in the context of Go, but in general, what do you perceive as the main strengths and weaknesses of sharing parameters between value function and policy? [2 marks]

b. The parameters θ are updated through multiple iterations. Each iteration generates data for a *supervised learning* problem; the weights are adjusted by gradient descent to minimise the corresponding loss function. What is the (input-output) data generated for the supervised learning problem; how is it generated? [2 marks]

Solution.
a. Sharing parameters between value function and policy might limit the space of (value function, policy) functions that can be represented. For example, in AlphaGo Zero, the value function is a linear combination of a set of features, and the policy is "linear with softmax" over the *same* features. Even if these features themselves are learned (since they are a hidden layer in a neural network), the value function and policy must remain "tied" through them—which might prevent more effective (value function, policy) pairs from being represented.

On the other hand, it is reasonable to assume that good policies and value functions will both benefit from "good" features describing the raw input. Sharing parameters makes the process of learning such features more efficient: if the features are trained to simultaneously benefit both prediction and control, they are likely to abstract out relevant patterns from the raw input.

b. Suppose the current weights are θ, the current policy is π_θ and the value function v_θ. An improved policy π' is obtained by applying MCTS. Whereas π' still indirectly depends on θ, its computation involves an (expensive) forward search. However, recording the action probabilities prescribed by π' for any state s enables the application of supervised learning to directly associate the action probabilities of π' with s through the neural network. Such state-to-policy mappings are recorded in self play. The outcomes of games (± 1) also double up as the values of states encountered along the game. Hence, a single game of self-play generates $\pi'(s), \text{value-target}(s)$ for every state s encountered along the game. The authors pick a single state (and corresponding labels) from each game and create the training data set for supervised learning. The actual loss function aims to match the policy, reduce the value prediction error, as well as regularise θ.

Week 4

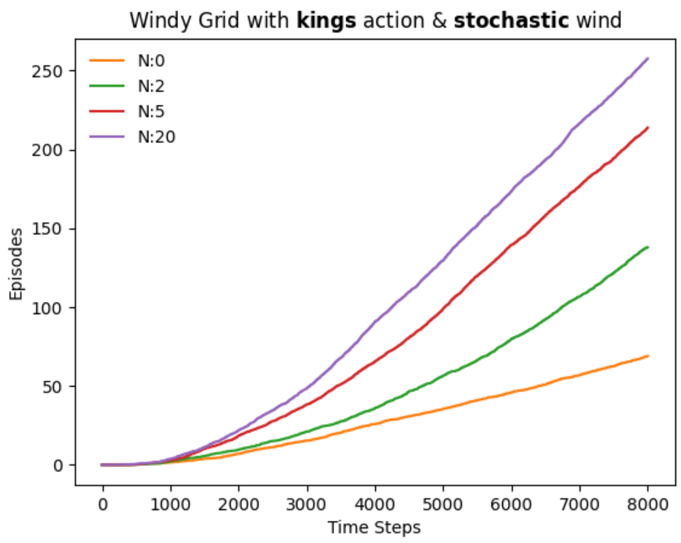
Question. This week you are given a short programming assignment related to Dyna-Q, and also a regular question related to the helicopter control application of Ng *et al.* (2003) (see the week's lecture slides for reference).

a. Recall the "Windy Gridworld" task (Example 6.5, Sutton and Barto, 2018) that you implemented as a part of a programming assignment in CS 747. In particular consider Exercise 6.10 from the textbook, which asks you to implement a "stochastic wind", with the agent allowed to make "King's moves". (1) In your CS 747 assignment, you have implemented a learning agent that uses Sarsa. You can reuse your code. (1) First, change the learning algorithm to Q-learning (using the same values for α and ϵ). Generate a plot similar to the one in the textbook under Example 6.5. (2) Implement Dyna-Q. The textbook provides pseudocude for deterministic environments, but the class lecture (Slide 5) generalises to stochastic environments—which you will have to use in this case. Observe that the parameter N specifies the number of model-based updates. Your earlier plot can be viewed as an instance of Dyna-Q with $N = 0$. To it add curves for $N = 2$, $N = 5$, and $N = 20$. Let each plot be an average of at least 10 independent runs with different random seeds. You can decide whether to place all the plots in a single graph or to have separate graphs. [4 marks]

b. Ng *et al.* (2003) make minor changes to the policy class Π_{H} used for the hovering task to obtain the policy class Π_{T} for trajectory-following. What changes are these, and why are they brought in? Π_{T} appears to be a superset of Π_H; why do you think the authors did not use Π_{T} itself for hovering, too? [2 marks]

In a single pdf file, put your plot(s) for (a) along with a description of your experiment (hyperparameters, design choices, observations, etc.); also include your answer to (b). Submit a compressed directory containing the pdf file and your code for (a). We will examine your code but not run it ourselves. Feel free to use a programming language/environment of your choice.

Solution.
a. Here is a plot prepared by TA Santhosh Kumar G. As expected, performance on the task progressively improves as N is increased, even if the agent's *computational burden* also increases.



b. In this application, policies for hovering as well as trajectory-following are encoded as neural networks. The authors carefully design these networks such that each control (an output) receives information from the state variables (and derivatives) that are construed to be useful for setting that control. The exact way in which the inputs influence the control is determined by the network topology and weights. Since the weights are optimised by policy search, it is a good idea to keep the dimensionality low—to this end the authors do not include input-output connections deemed unnecessary (based on domain knowledge).

Trajectory-following is arguably a harder task than hovering; the authors include three additional connections to π_{H} so as to obtain π_{T}. These connections provide information that is needed for trajectory-following but is deemed less relevant for hovering. Although the hovering policy could also be found by searching π_{T} instead of π_{H}, the search is likely to be less efficient due to the additional weights.

Week 2

Question. This question takes you through the steps to construct a proof that applying MCTS at every encountered state with a non-optimal rollout policy π_{R} will lead to higher aggregate reward than that obtained by following π_{R} itself.

a. For MDP (S, A, T, R, γ), a *nonstationary* policy $\pi = (\pi^0, \pi^1, \pi^2, \dots)$ is an infinite sequence of stationary policies $\pi^i : S \to A$ for $i \geq 0$ (we assume the per-time-step policies are deterministic and Markovian). V^π and Q^π have the usual definitions. For $s \in S, a \in A$, (1) $V^\pi(s)$ is the expected long-term reward obtained by acting according to π from state s, and (2) $Q^\pi(s, a)$ is the expected long-term reward obtained by taking a from s, and thereafter acting according to π (π^0 gives the second action, π^1 gives the third, etc.). Denote by $\textsc{head}(\pi)$ the stationary policy π^0 that is first in the sequence, and by $\textsc{tail}(\pi)$ the remaining sequence—itself a nonstationary policy—$(\pi^1, \pi^2, \pi^3, \dots)$. Show that $V^\pi \succeq V^{\textsc{tail}(\pi)} \implies V^{\textsc{head}(\pi)} \succeq V^{\textsc{tail}(\pi)}$. It will help to begin with a Bellman equation connecting V^π and $V^{\textsc{tail}(\pi)}$. Thereafter the proof will follow the structure in the analysis of the policy improvement theorem. [3 marks]

b. Consider an application of MCTS in which a tree of depth $d \geq 1$ is constructed and rollout policy $\pi_{\text{R}} : S \to A$ is used. Assume that an *infinite* number of rollouts is performed: hence evaluations within the search tree, and those at the leaves using π_{R}, are *exact*. Although tree search is undertaken afresh from each "current" state, we may equivalently view tree search as the application of a nonstationary policy $\pi = (\pi^0, \pi^1, \pi^2, \dots, \pi^{d-1}, \pi_{\text{R}}, \pi_{\text{R}}, \pi_{\text{R}}, \dots)$ on the original MDP, starting from the current state, where for $i \in \{0, 1, \dots, d-1\}, s \in S$,

$$\pi^i(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} T(s, a, s')\{R(s, a, s') + \gamma V^{(\pi^{i+1}, \pi^{i+2}, \pi^{i+3}, \dots)}(s')\}.$$

This nonstationary policy π is constructed in the agent's mind, but it is eventually π^0 that is applied in the (real) environment. By our convention, $\pi^d, \pi^{d+1}, \pi^{d+2}, \dots$ all refer to π_{R}. Show that $V^{(\pi^{d-1}, \pi^d, \pi^{d+1}, \dots)} \succ V^{\pi_{\text{R}}}$, and show for $i \in \{0, 1, \dots, d-2\}$ that $V^{(\pi^i, \pi^{i+1}, \pi^{i+2}, \dots)} \succeq V^{(\pi^{i+1}, \pi^{i+2}, \pi^{i+3}, \dots)}$. [4 marks]

c. Put together the results from a and b to show that $V^{\pi^0} \succ V^{\pi_{\text{R}}}$. [1 mark]

Solution.

a. Observe that $V^\pi = B^{\textsc{head}(\pi)}(V^{\textsc{tail}(\pi)})$. Thus, the antecedent is $B^{\textsc{head}(\pi)}(V^{\textsc{tail}(\pi)}) \succeq V^{\textsc{tail}(\pi)}$. Since the Bellman operator preserves $\succeq$, a repeated application gives

$$\begin{aligned}
B^{\textsc{head}(\pi)}(V^{\textsc{tail}(\pi)}) &\succeq V^{\textsc{tail}(\pi)}, \\
(B^{\textsc{head}(\pi)})^2(V^{\textsc{tail}(\pi)}) &\succeq B^{\textsc{head}(\pi)}(V^{\textsc{tail}(\pi)}), \\
(B^{\textsc{head}(\pi)})^3(V^{\textsc{tail}(\pi)}) &\succeq (B^{\textsc{head}(\pi)})^2(V^{\textsc{tail}(\pi)}), \\
&\vdots
\end{aligned}$$

and hence $\lim_{l \to \infty}(B^{\textsc{head}(\pi)})^l(V^{\textsc{tail}(\pi)}) = V^{\textsc{head}(\pi)} \succeq V^{\textsc{tail}(\pi)}$.

b. For $s \in S$,

$$\begin{aligned} V^{(\pi^{d-1},\pi^{d},\pi^{d+1},\dots)}(s) &= \max_{a \in A} \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma V^{\pi_{\mathrm{R}}}(s')\} \\ &\geq \sum_{s' \in S} T(s,\pi_{\mathrm{R}}(s),s')\{R(s,\pi_{\mathrm{R}}(s),s') + \gamma V^{\pi_{\mathrm{R}}}(s')\} \\ &= V^{\pi_{\mathrm{R}}}(s), \end{aligned}$$

and moreover, since π_{R} is not optimal, there is some state $\bar{s} \in S$ such that $V^{(\pi^{d-1},\pi^{d},\pi^{d+1},\dots)}(\bar{s}) > V^{\pi_{\mathrm{R}}}(\bar{s})$. In short, $V^{(\pi^{d-1},\pi^{d},\pi^{d+1},\dots)} \succ V^{\pi_{\mathrm{R}}}$. If we assume that $V^{(\pi^{i+1},\pi^{i+2},\pi^{i+3},\dots)} \succeq V^{(\pi^{i+2},\pi^{i+3},\pi^{i+4},\dots)}$, we observe that for $s \in S$,

$$\begin{aligned} V^{(\pi^{i},\pi^{i+1},\pi^{i+2},\dots)}(s) &= \max_{a \in A} \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma V^{(\pi^{i+1},\pi^{i+2},\pi^{i+3},\dots)}(s')\} \\ &\geq \sum_{s' \in S} T(s,\pi^{i+1}(s),s')\{R(s,\pi^{i+1}(s),s') + \gamma V^{(\pi^{i+1},\pi^{i+2},\pi^{i+3},\dots)}(s')\} \\ &\geq \sum_{s' \in S} T(s,\pi^{i+1}(s),s')\{R(s,\pi^{i+1}(s),s') + \gamma V^{(\pi^{i+2},\pi^{i+3},\pi^{i+4},\dots)}(s')\} \\ &= V^{(\pi^{i+1},\pi^{i+2},\pi^{i+3},\dots)}(s), \end{aligned}$$

which completes our proof.

c. Applying the result in b for $i = 0$ gives $V^{\pi} \succeq V^{\mathrm{TAIL}(\pi)}$, which, from a, implies $V^{\pi^0} \succeq V^{\mathrm{TAIL}(\pi)}$. However, from b, we also have

$$V^{\mathrm{TAIL}(\pi)} = V^{(\pi^{1},\pi^{2},\pi^{3},\dots)} \succeq V^{(\pi^{2},\pi^{3},\pi^{4},\dots)} \succeq \cdots \succeq V^{(\pi^{d-1},\pi^{d},\pi^{d+1},\dots)} \succ V^{\pi_{\mathrm{R}}},$$

which means $V^{\pi^0} \succ V^{\pi_{\mathrm{R}}}$.

Week 1

Question.

a. Consider the value iteration algorithm provided on Slide 12 in the week's lecture. Assume that the algorithm is applied to a two player zero-sum Markov game (S, A, O, R, T, γ). Using Banach's fixed point theorem, show that the algorithm converges to the game's minimax value $V^\star$ (which you can assume to be unique, even though we did not present a proof in the lecture). Clearly state any results that your derivation uses. [3 marks]

b. Consider $G(2, 2)$, the class of two player general-sum Matrix games in which each of the players, A and O, has exactly two actions. In a "pure-strategy Nash equilibrium" (π^A, π^O), the individual strategies π^A and π^O are both pure (deterministic). Do there exist games in $G(2, 2)$ that have *no* pure-strategy Nash equilibria, or are all games in $G(2, 2)$ guaranteed to have at least one pure-strategy Nash equilibrium? Justify your answer. [2 marks]

Solution.

a. In this setting value iteration repeatedly applies operator $B^\star : (S \to \mathbb{R}) \to (S \to \mathbb{R})$ defined by

$$B^\star(X)(s) \overset{\text{def}}{=} \max_{\pi \in \text{PD}(A)} \min_{o \in O} \sum_{a \in A} \pi(a) \left\{ R(s, a, o) + \gamma \sum_{s' \in S} T(s, a, o, s') X(s') \right\}$$

for $X : S \to \mathbb{R}$ and $s \in S$. First, observe that $V^\star$ is the unique fixed point of this operator. If we start the iteration with any arbitrary $V^0 : S \to \mathbb{R}$ and set $V^{t+1} \leftarrow B^\star(V^t)$ for $t \geq 0$, Banach's fixed point theorem assures convergence to $V^\star$ if $B^\star$ is a contraction mapping. We show the same by considering arbitrary $X : S \to \mathbb{R}$ and $Y : S \to \mathbb{R}$. We use the rule that for functions $f : U \to \mathbb{R}$ and $g : U \to \mathbb{R}$ on the same domain U, $|\max_{u \in U} f(u) - \max_{u \in U} g(u)| \leq \max_{u \in U} |f(u) - g(u)|$. For convenience we use $\alpha(Z, s, \pi, o)$ as shorthand for $\sum_{a \in A} \pi(a)\{R(s, a, o) + \gamma \sum_{s' \in S} T(s, a, o, s') Z(s')\}$ for $Z \in \{X, Y\}$, $s \in S$ $\pi \in \text{PD}(A), o \in O$.

$$\begin{aligned}
\|B^\star(X) - B^\star(Y)\|_\infty &= \max_{s \in S} |B^\star(X)(s) - B^\star(Y)(s)| \\
&= \max_{s \in S} \left| \max_{\pi \in \text{PD}(A)} \min_{o \in O} \alpha(X, s, \pi, o) - \max_{\pi \in \text{PD}(A)} \min_{o \in O} \alpha(Y, s, \pi, o) \right| \\
&\leq \max_{s \in S} \max_{\pi \in \text{PD}(A)} \left| \min_{o \in O} \alpha(X, s, \pi, o) - \min_{o \in O} \alpha(Y, s, \pi, o) \right| \\
&= \max_{s \in S} \max_{\pi \in \text{PD}(A)} \left| \max_{o \in O}(-\alpha(X, s, \pi, o)) - \max_{o \in O}(-\alpha(Y, s, \pi, o)) \right| \\
&\leq \max_{s \in S} \max_{\pi \in \text{PD}(A)} \max_{o \in O} |(-\alpha(X, s, \pi, o)) - (-\alpha(Y, s, \pi, o))| \\
&= \gamma \max_{s \in S} \max_{\pi \in \text{PD}(A)} \max_{o \in O} \left| \sum_{s' \in S} T(s, a, o, s')(Y(s') - X(s')) \right| \\
&\leq \gamma \max_{s \in S} \max_{\pi \in \text{PD}(A)} \max_{o \in O} \sum_{s' \in S} T(s, a, o, s') \|X - Y\|_\infty = \|X - Y\|_\infty.
\end{aligned}$$

b. The table below corresponds to a game with no pure-strategy Nash equilibria. Cells show "A's reward, O's reward" when A plays the row action and O plays the column action.

	c	d
a	$1, 1$	$1, 2$
b	$0, 1$	$2, 0$