CS221 Exam 1 Solutions

Winter 2021

Please read all of the following information before starting the exam:

- This test has 2 problems on 16 pages for a total of 50 possible points.
- **You must submit your exam within 100 mins.** The exam submission window will not automatically close after 100 mins; however, for every minute you submit late, we will deduct 1 point on the exam score. No submissions that are more than 10 minutes late after the 100-min exam window will be accepted.
- Note that different questions are worth different amounts of points. Budget your time accordingly!
- Keep your answers precise and concise. We may award partial credit so show all your work clearly and in order.
- Don't spend too much time on one problem. Read through all the problems carefully and do the easier ones first.
- If you are unsure about a problem statement when taking the exam, state your assumptions. We will take all reasonable assumptions into account when grading.
- This exam is open-book; you may use any inanimate resources, including the course website.
- Being subject to the provisions of the Honor Code means in part that you must observe the rules established for this exam, which are: you may consult only inanimate sources. You may not consult or collaborate with anyone about the questions. Such collaboration is a violation of the Honor Code.
- Good luck!

Problem	Part	Max Score	Score
1	a	12	
	b	8	
	Total	**20**	
2	a	6	
	b	6	
	c	2	
	d	5	
	e	1	
	f	5	
	g	5	
	Total	**30**	

Total Score: $\boxed{} + \boxed{} = \boxed{}$

0. Honor Code (*0 points*) Please write or type down the honor code below and sign your name. Your exam will not be graded if this question is not completed.

"I will not consult or collaborate with anyone about the questions. Such collaboration is a violation of the Honor Code."

1. Movie Genre(s) (*20 points*)

You are interested in classifying the genre(s) of a movie given its plot summary. For simplicity, assume there are only three genres in the universe: action, romance, and comedy. You decide that you will model this as three binary classification problems, one for each genre.

For each plot summary x, you represent the corresponding movie M's genre(s) with $y \in \{0,1\}^3$, where

$$\begin{aligned} y_1 &= \mathbf{1}[M \text{ is an action movie}], \\ y_2 &= \mathbf{1}[M \text{ is a romance movie}], \\ y_3 &= \mathbf{1}[M \text{ is a comedy movie}]. \end{aligned}$$

a. (*12 points*) **Sharing is Caring**

You decide to use a feature extractor ϕ that maps each word in the English vocabulary to the number of occurrences of that word in the plot summary. Assume our English vocabulary D contains only the top 10,000 words in English (by frequency). Any out-of-vocabulary word is ignored.

Recall that the logistic function $\sigma(z) = (1+e^{-z})^{-1}$ takes in a real number and outputs a probability in $(0,1)$. After some research, you learned that the logistic function is often used with the *logistic loss* function for binary classification. Given a label $y \in \{0,1\}$ and predicted probability $p \in [0,1]$,

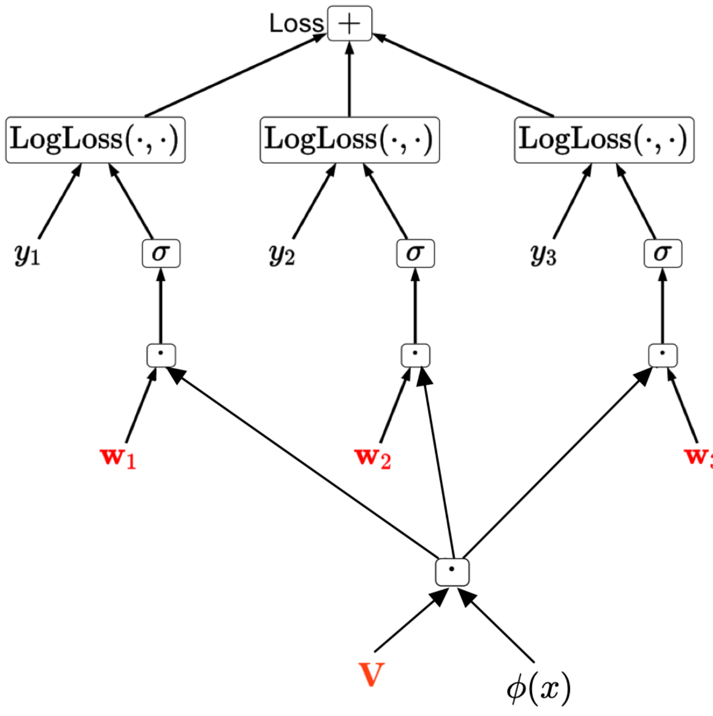
$$\text{LogLoss}(y,p) = -\big(y \log p + (1-y)\log(1-p)\big).$$

You decide to use these shiny new tools to solve your problem.

(i) [1 point] One way to solve your problem is to build three binary classifiers. For a plot summary x and genre $k \in \{1,2,3\}$, the predicted probability that $y_k = 1$ is $p_k = \sigma(\mathbf{w}_k \cdot \phi(x))$. How many parameters will there be in the three classifiers combined? No justification required.

Solution $30,000$. Each classifier has $10,000$ parameters, so there are $3 \times 10,000 = 30,000$ parameters.

(ii) [6 points] You've heard that having too many parameters can lead to bad generalization and want to find a way to reduce it. Since the three binary classification tasks are all about predicting movie genre and are thus related in nature, why not let the three classifiers share parameters? You come up with the following computation graph:



where $\mathbf{V}$ is a trainable weight matrix shared among all three classifiers such that $\mathbf{V}\phi(x) \in \mathbb{R}^2$ for all x.

Now, for any $k \in \{1, 2, 3\}$, the predicted probability that $y_k = 1$ is $p_k = \sigma(\mathbf{w}_k \cdot \mathbf{V}\phi(x))$.

- In this setup, how many parameters are there in the three classifiers combined? No justification required.

Solution 20,006. Since $\mathbf{V}\phi(x) \in \mathbb{R}^2$, $\mathbf{V}$ must be a $2 \times |D|$ matrix, which has $2 \times 10,000$ parameters. For the dimensionality to match, w_1, w_2, w_3 are all 2-dimensional vectors. So there are 20,006 parameters in total.

- The pointwise loss function $\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ is defined as follows:

  $$\begin{aligned}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) &= \sum_{k=1}^{3} \text{LogLoss}(y_k, p_k) \\ &= \sum_{k=1}^{3} -\Big(y_k \log \sigma(\mathbf{w}_k \cdot \mathbf{V}\phi(x)) + (1 - y_k) \log \big(1 - \sigma(\mathbf{w}_k \cdot \mathbf{V}\phi(x))\big)\Big).\end{aligned}$$

  Derive the gradient of the pointwise loss with respect to $\mathbf{w}_1$, i.e., $\nabla_{\mathbf{w}_1}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$. You can use p_1 in your final expression. Show your work.
  Hint: Feel free to use $\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big)$ directly without showing the intermediate steps.

  Solution

  $$\nabla_{\mathbf{w}_1}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) = (p_1 - y_1)\mathbf{V}\phi(x)$$

  Derivation:

  $$\begin{aligned}
  &\nabla_{\mathbf{w}_1}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) && (1)\\
  =&\nabla_{\mathbf{w}_1} \text{LogLoss}(y_1, p_1) \\
  =&\nabla_{\mathbf{w}_1} -\Big(y_1 \log \sigma(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)) + (1 - y_1) \log \big(1 - \sigma(\mathbf{w}_1 \cdot \mathbf{V}\phi(x))\big)\Big) && (2)\\
  =&-\big(y_1 \frac{1}{p_1}\sigma'\big(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)\big)\nabla_{\mathbf{w}_1}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)) \\
  &+ (1 - y_1)\frac{1}{1 - p_1}\Big(-\sigma'\big(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)\big)\Big)\nabla_{\mathbf{w}_1}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x))\big) && (3)\\
  =&-\big(y_1 \frac{p_1(1 - p_1)}{p_1}\nabla_{\mathbf{w}_1}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)) - (1 - y_1)\frac{p_1(1 - p_1)}{1 - p_1}\nabla_{\mathbf{w}_1}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x))\big) && (4)\\
  =&-\big(y_1(1 - p_1)\mathbf{V}\phi(x) - (1 - y_1)p_1\mathbf{V}\phi(x)\big) && (5)\\
  =&-\mathbf{V}\phi(x)(y_1 - p_1) = (p_1 - y_1)\mathbf{V}\phi(x) && (6)
  \end{aligned}$$

(iii) [5 points] You plan to use regular gradient descent to train your classifiers. Suppose you have a dataset of N points $(x^{(i)}, y^{(i)})$. You define the overall training loss as follows:

$$\text{TrainLoss}(\mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) = \frac{1}{N} \sum_{i=1}^{N} \text{Loss}(x^{(i)}, y^{(i)}, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3).$$

Below is your gradient descent algorithm:

Algorithm 1: Gradient Descent

```
1: Randomly shuffle the training data
2: Initialize $\mathbf{w}_1 = \mathbf{w}_2 = \mathbf{w}_3 = \mathbf{0}$, initialize $\mathbf{V}$ with random non-zero matrix
3: for $t = 1, 2, \ldots, T$ do
4:    // calculate gradients
5:    for $k = 1, 2, 3$ do
6:       $\mathbf{v}_k \leftarrow \nabla_{\mathbf{w}_k} \text{TrainLoss}(\mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ // get gradient w.r.t $\mathbf{w}_k$
7:    end for
8:    $\mathbf{U} \leftarrow \nabla_{\mathbf{V}} \text{TrainLoss}(\mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ // get gradient w.r.t $\mathbf{V}$
9:
10:   // update weights
11:   for $k = 1, 2, 3$ do
12:      $\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta \mathbf{v}_k$ // update $\mathbf{w}_k$
13:   end for
14:   $\mathbf{V} \leftarrow \mathbf{V} - \eta \mathbf{U}$ // update $\mathbf{V}$
15: end for
```

Consider a training dataset $\mathcal{D}$ with two datapoints $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$. You run gradient descent on $\mathcal{D}$ and initialize $\mathbf{w}_1 = \mathbf{w}_2 = \mathbf{w}_3 = \mathbf{0}$ per Line 2 in the algorithm described above. Upon initializing $\mathbf{V}$ randomly, we see that

$$\begin{aligned} \mathbf{V}\phi(x^{(1)}) &= [5, -2]^T, \quad y^{(1)} = [0, 1, 1]^T \\ \text{and } \mathbf{V}\phi(x^{(2)}) &= [-1, 2]^T, \quad y^{(2)} = [1, 0, 1]^T. \end{aligned}$$

You run gradient descent on this dataset with $\eta = 0.1$. What is the value of $\mathbf{w}_1$ at the end of one iteration? Show your work.

Note: We are expecting the value of $\mathbf{w}_1$ only, **not** $\mathbf{w}_2$ or $\mathbf{w}_3$.

Solution For all k,

$$\begin{aligned} \mathbf{v}_k = \nabla_{\mathbf{w}_k}\text{TrainLoss}(\mathbf{V}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) &= \frac{1}{2}\sum_{i=1}^{2} \nabla_{\mathbf{w}_k} Loss(x^{(i)}, y^{(i)}, \mathbf{V}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) \\ &= \frac{1}{2}\big((p_k^{(1)} - y_k^{(1)})\mathbf{V}_0\phi(x^{(1)}) + (p_k^{(2)} - y_k^{(2)})\mathbf{V}_0\phi(x^{(2)})\big). \end{aligned}$$

Note that for all k, since $\mathbf{w}_k = \mathbf{0}$, $p_k^{(1)} = p_k^{(2)} = \sigma(\mathbf{0} \cdot \mathbf{V}_0\phi(x)) = \sigma(0) = (1+e^0)^{-1} = 0.5$. And since $\eta = 0.1$, after the first iteration, $\mathbf{w}_k = \mathbf{0} - \eta\mathbf{v}_k = -0.1\mathbf{v}_k$.

We see that

$$\mathbf{v}_1 = \frac{1}{2}\big((0.5 - 0)[5, -2]^T + (0.5 - 1)[-1, 2]^T\big) = [1.5, -1]^T.$$

Therefore,

$$\mathbf{w}_1 = [-0.15, 0.1]^T.$$

Misses that the teaching staff saw include:

- Not averaging the gradient update across the dataset (i.e., dropping the $\frac{1}{N}$ term)
- Calculating gradient but forgetting to update
- Mixing up the subscript used in $\mathbf{w}_1$ with the superscripts used in the datapoints

b. (*8 points*) **Less is More**

You tried running gradient descent, but since $\mathbf{V}$ is a large matrix, your computer does not have enough memory to store the gradient with respect to $\mathbf{V}$, so you couldn't train your model properly.

Luckily, your friend Alice has worked on a similar problem (e.g., classifying fiction genres given synopsis) and trained a model with the same architecture as yours. You ask Alice to share with you the weights of her *trained* classifier, which includes a weight matrix $\mathbf{V}_0$ that has exactly the same shape as $\mathbf{V}$.

Since Alice's problem is similar to yours, you believe the solution should be similar as well. Hence, you decide to initialize $\mathbf{V}$ with $\mathbf{V}_0$ and never update it in order to get around the memory constraint. In other words, **you run gradient descent only on $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$.**

You tell Alice about your new gradient descent algorithm, and the idea to make a new memory-efficient classifier that shares $\mathbf{V}_0$.

Alice responds: "Actually, what you have here isn't a new kind of classifier. It's actually three classifiers with a shared feature extractor." Though shocked at first, after some thought, you agree with her as well.

(i) [2 points] Show that Alice is right by filling in the blank below:

Given a plot summary x, for each genre $k \in \{1, 2, 3\}$,
the predicted probability that $y_k = 1$ is $p_k = \sigma(\mathbf{w}_k \cdot \tau(x))$,
where $\tau(x) =$ ____________ is the shared feature extractor.

No justification required.

Solution $\tau(x) = \mathbf{V}_0\phi(x)$.

(ii) [1 point] In this setup, how many parameters are there in the three classifiers combined? No justification required.

Solution 6. The only parameters are $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$, which are all 2-dimensional.

(iii) [5 points] Between the following:

(A) three classifiers that share $\phi(x)$ as feature extractor (i.e., your idea in **a**(i))

(B) three classifiers that share $\tau(x)$ as feature extractor

Which setup has higher approximation error? Which one has higher estimation error? Justify your answer with 1-2 sentences.

Solution (B) has higher approximation error. (A) has higher estimation error. This is because (A) has a much larger hypothesis class. Each of the three weight vectors in (A) has much higher dimensionality than each of the three weight vectors in (B). More specifically, as we saw in 1a(i), each weight vector in (A) is 10,000-dimensional; as we saw in 1b(ii), each weight vector in (B) is 2-dimensional.

A commonly seen incorrect justification relates estimation/approximation error to the accuracy/inaccuracy of having a pre-trained weight matrix rather than the number of parameters.

2. Karel Loves Cookies! (*30 points*)

We revisit a homework question and check in on Karel, our favorite cookie-loving robot. Recall that Karel is a robot placed in an $m \times n$ grid and wishes to reach the cell with the cookie using the least cost. There are also pitfalls scattered on the grid – be careful not to step into them!

At each step, Karel may use one of the following commands:

1. `turnLeft()`: rotate the current direction 90 degrees counter-clockwise (costs 1);
2. `turnRight()`: rotate the current direction 90 degrees clockwise (costs 1);
3. `move()`: shift by 1 cell along the current direction (costs 1);
4. `leap()`: shift by 2 cells along the current direction (costs 3).

Note that moving or leaping onto a pitfall has an additional cost of 100 and terminates the game, but leaping over a pitfall doesn't have such penalty.

Initially, Karel starts at the top-left corner of the grid (1, 1) and faces east. You may assume that the cookie is always placed at the bottom-right corner (m, n) unless otherwise specified.

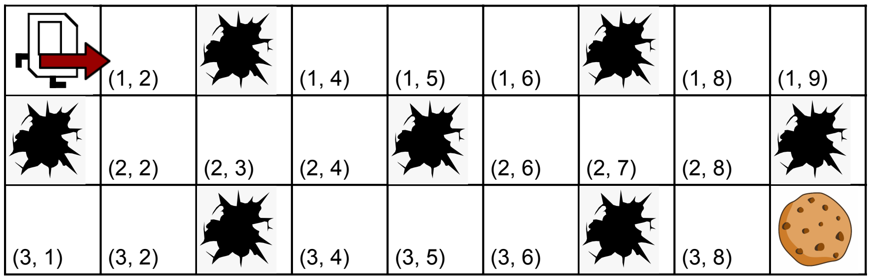


Figure 1: An example 3×9 grid with 7 pitfalls and 1 cookie. Karel starts at (1, 1) and faces east. An example trajectory to the cookie: `move()` into (1, 2), $\rightarrow$ `leap()` into (1, 4) since there's a pitfall in the middle $\rightarrow$ `move()` twice into (1, 6) $\rightarrow$ `leap()` into (1, 8) $\rightarrow$ `move()` into (1, 9) $\rightarrow$ `turnRight()`$\rightarrow$ `leap()` into the cookie!

a. (*6 points*) Fill out the components of the search problem corresponding to the above problem setting. Each state consists of the cell coordinate (i, j) and the direction $d \in \{E,\ S,\ W,\ N\}$.

In your answer write up, you may use helper functions `counter-clockwise`(d), `clockwise`(d), $\triangle i(d)$, $\triangle j(d)$ whose values are defined in Table 1:

- $s_{\text{start}} = (1, 1, E)$.
- $\text{Actions}(i, j, d) = \{a \in \{$`turnLeft()`, `turnRight()`, `move()`, `leap()`$\}$:
  $\text{Succ}((i, j, d), a)$ doesn't exceed the boundary$\}$
- $\text{IsEnd}(i, j, d) =$

d	`counter-clockwise`(d)	`clockwise`(d)	$\triangle i(d)$	$\triangle j(d)$
E	N	S	0	+1
N	W	E	-1	0
W	S	N	0	-1
S	E	W	+1	0

Table 1: Helper functions for answer write up.

Solution IsEnd$(i, j, d) = \mathbf{1}$[cell (i, j) contains cookie or is a pitfall].

Forgetting to include the pitfall condition was a common mistake.

- Succ$((i, j, d), a) =$

Solution

$$\begin{cases} (i, j, \texttt{counter-clockwise}(d)) & a = \texttt{turnLeft()} \\ (i, j, \texttt{clockwise}(d)) & a = \texttt{turnRight()} \\ (i + \triangle i(d), j + \triangle j(d), d) & a = \texttt{move()} \\ (i + 2\triangle i(d), j + 2\triangle j(d), d) & a = \texttt{leap()} \end{cases}$$

A number of students forgot to include the original i and j values when writing out the successor states for `move()` and `leap()`.

- Cost$((i, j, d), a) =$

Solution

$$C_1 + C_2$$

where

$$C_1 = \begin{cases} 1 & a \in \{\texttt{turnLeft()}, \texttt{turnRight()}, \texttt{move()}\} \\ 3 & a = \texttt{leap()} \end{cases}$$

$$C_2 = \begin{cases} 100 & \text{Succ((i, j, d), a) is a pitfall} \\ 0 & \textit{otherwise} \end{cases}$$

Mistakes the teaching staff saw included using 2 instead of 3 for the cost of `leap()`, forgetting to add the cost of moving into a pitfall to the original `move()` or `leap()` action, and forgetting to account for the pitfall cost altogether.

b. (*6 points*) You decide to use the A* algorithm to help Karel find the cookie with the least cost. Define a consistent heuristic function $h(i, j, d)$ for this problem. Briefly explain why your heuristic function is consistent.

[Hint: Think about a relaxed version of the original problem.]

Solution To construct a consistent heuristic, one convenient method is to build a relaxed problem and compute its FutureCost. One possible solution is the Manhattan distance from each cell to the cookie. This is because (1) When all the pitfalls are removed and the direction is ignored, we construct a relaxed version of the original problem, and (2) The FutureCost of this relaxed problem is the Manhattan distance, because the cost of one `leap()` is larger than two `move()`s.

While many students initially proposed the Manhattan distance as a heuristic, a good percentage did not fully explain why it is consistent. In particular, it's important to note that the cost of taking a `leap()` is more expensive than taking two `move()`s. If this were not the case, then the Manhattan distance would not underestimate the minimum cost to the goal state, and hence would not be an admissible (and therefore consistent) heuristic for A*.

c. (*2 points*) Would dynamic programming be an appropriate algorithm for solving this search problem as well? Explain why or why not.

Solution Dynamic programming would not be applicable in this case, as the associated graph is not acyclic. For example, Karel could `turnLeft()` 4 times and wind up facing the same direction in the same location.

One common mistake was to claim dynamic programming is applicable because Karel's *optimal* path does not contain cycles. Unfortunately, while it is true that Karel would not take a cycle on the optimal path, the entire graph must be acyclic for dynamic programming to work. It's a good exercise to think about how even a single cycle in the graph would cause DP to break down and infinitely loop.

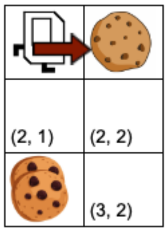
d. (*5 points*) Cookie craving still unsatisfied, Karel heads to a grid environment with multiple cookies. Now assume we associate a *reward* when Karel gets to a cookie-holding location (after all, Karel should have some reason to expend so much energy for a cookie!). Specifically, if Karel reaches a location that has a cookie, then Karel obtains a positive reward of 3 *for each cookie in the square*. We still assume Karel is penalized for each of the 4 commands, but for consistency we frame the costs as *negative rewards* (i.e., take the cost value of each command and negate it).

Consider the following toy instantiation of this environment:

Specifically, we have

- Karel starts at location (1, 1) facing east.
- Location (1, 2) has a single cookie.
- Location (3, 1) has two cookies.

Assume Karel's search terminates upon entering a location with a cookie (i.e., each cookie-bearing location is a terminal state). Karel wants to maximize the *discounted sum of rewards*. Describe Karel's trajectory and sum of rewards on this toy problem when:



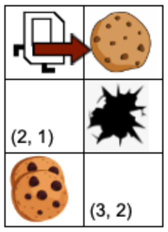
1. The discount factor $\gamma = 0$.

2. The discount factor $\gamma = 1$.

Solution When $\gamma = 0$, Karel moves one square ahead for a total reward of $-1 + 3 = 2$. When $\gamma = 1$, Karel turns to the right and moves two squares ahead for a total reward of $-1 + -1 + -1 + 6 = 3$. This behavior lines up with our interpretation of the discount factor as our emphasis on short versus long-term rewards. When $\gamma = 0$, Karel only focuses on its immediate reward and takes the greedy option, whereas when $\gamma = 1$ Karel accrues negative rewards initially but for the sake of a larger long-term reward.

Some students forgot that each cookie-bearing state is terminal and provided trajectories such that Karel visits both cookie states. Others forgot that Karel must call `turnRight()` first before heading to the state with two cookies, which results in an additional cost.

e. (*1 point*) Look out! The ground caves in under location (2, 2), leaving a pitfall behind.



If Karel ends up in a location with a pitfall, then it falls in and receives a negative reward of -100, ending Karel's search. Assuming the same setup as in part **d.**, does this new obstacle affect Karel's behavior? Briefly explain.

Solution No. Since Karel's actions are still deterministic, the optimal behavior is not affected by the hole.

f. (*5 points*) Now something else is wrong...Karel's controller is malfunctioning! Precisely, when Karel calls `move()` while facing direction D, for some probability p_{slide} it instead moves randomly in one of the three other directions (but is still facing direction D). Unfazed by this new predicament, Karel analyzes the problem through the lens of Markov Decision Processes. However, Karel doesn't know p_{slide} and must resort to Reinforcement Learning to account for this missing information.

(i) [3 points] Karel first wants to use *model-based Monte Carlo* in order to estimate the transition probabilities of the MDP. Karel goes through one trajectory in the environment and experiences the following state-action-rewards:

s_0	a_1	r_1	
(1, 1, E)	`move()`	-1	
s_1	a_2	r_2	
(1, 2, E)	`move()`	-1	
s_2	a_3	r_3	
(1, 1, E)	`move()`	-1	
s_3	a_4	r_4	
(2, 1, E)	`leap()`	-3	
s_4	a_5	r_5	
(2, 3, E)	`turnRight()`	-1	
s_5	a_6	r_6	s_6
(2, 3, S)	`move()`	1	(3, 3, S)

Estimate the transition probability $\hat{T}(s, a, s')$, where $s = (1, 1, E)$, $a =$ `move()`, and $s' = (1, 2, E)$. Additionally estimate p_{slide}. Are these estimates reliable?

- $\hat{T}(s, a, s') =$

  Solution 1/2. Karel experienced (s, a) twice and (s, a, s') once.

- $p_{\text{slide}} =$

  Solution also 1/2. Karel called `move()` 4 times, and 2 of those times it moved in a different direction than it was facing.

Solution We should **not** be very confident in these estimates. Karel needs many more experiences in the environment to gather enough observations such that the Monte Carlo averages are likely to be close to the true probabilities.

A number of students seemed to mistake "reliability" for *accuracy*, stating that we don't know whether the estimates are close to the true values because of the lack of samples. While it's true that we may in fact get lucky and get the correct estimate with just a few experiences, we won't actually know this without knowing the true probabilities! All we have are the observations and their averages, and hence with only a single sample trajectory we can definitively say that we cannot rely on these estimates without gathering more experience.

(ii) [2 points] Karel wants to use SARSA in order to compute the optimal policy for its cookie-gathering problem. But the SARSA method discussed in the lecture modules can only be used to estimate the state-action value for an already-given policy! Is Karel at a dead-end?

As it turns out, the SARSA algorithm that we learned in lecture can easily be adapted to learn Q_{opt} and π_{opt}. Explain how you would modify SARSA to do so.

[Hint: We need to update the data-generating policy we use as the agent gains more experience. The policy should balance exploration of the MDP with exploitation of the current state-action value estimate.]

Solution Rather than having a fixed policy that we use to generate our s, a, r, s', a' quintuplets, we can instead use an ϵ-greedy policy based on our current $\hat{Q}$ estimate. If we let ϵ decrease over time, we will randomly explore less often and exploit our ever-improving estimate $\hat{Q}$ more often. As $\epsilon \to 0$, our estimate $\hat{Q} \to Q_{\text{opt}}$.

A good number of students proposed using an ϵ-greedy data-generating policy, which is a great start. However, it's also necessary to *decrease* ϵ over time (for example, you can set $\epsilon_t = 1/t$, where t is the current timestep) so that the agent begins to exploit the optimal actions more often as the estimate $\hat{Q}$ gets more and more accurate.

g. (*5 points*) Now, Karel gets his engine repaired (which means $p_{slide} = 0$) and is placed on a new 1×5 grid without any pitfalls. But there is a new problem. An invisible ghost is trying to block Karel by creating pitfalls! Whenever Karel arrives at cell, the ghost chooses a non-pitfall cell between Karel and the cookie (if such cell exists) and turns it into a pitfall.

The game is **all-or-nothing**: If Karel can reach the cookie, Karel wins with a utility of $+1$; Otherwise the ghost wins, leaving Karel a utility of -1. The per-step action costs are ignored. We assume that Karel can only use `move()` or `leap()` (i.e. no direction changes), and that the ghost plays adversarially.

You want to use alpha-beta pruning to help Karel. In the following two scenarios (i) Karel plays first, and (ii) ghost plays first, the unpruned game trees are shown in Figure 3. In each case, who wins at the end? Which nodes are pruned or unvisited during alpha-beta pruning? You may assume that the actions of each state node are visited in the same order as in Figure 3.

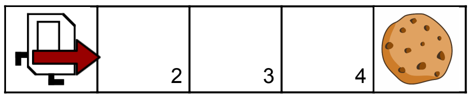


Figure 2: The new 1×5 grid.

Solution (i) 9, 10 (Karel wins); (ii) 8, 9, 10, 11, 12 (Ghost wins).

In (i), after observing that node 3 leads to "Karel wins", since node 2 is the Ghost's turn (a min node), the branch under node 6 must be visited in case the ghost can win back;

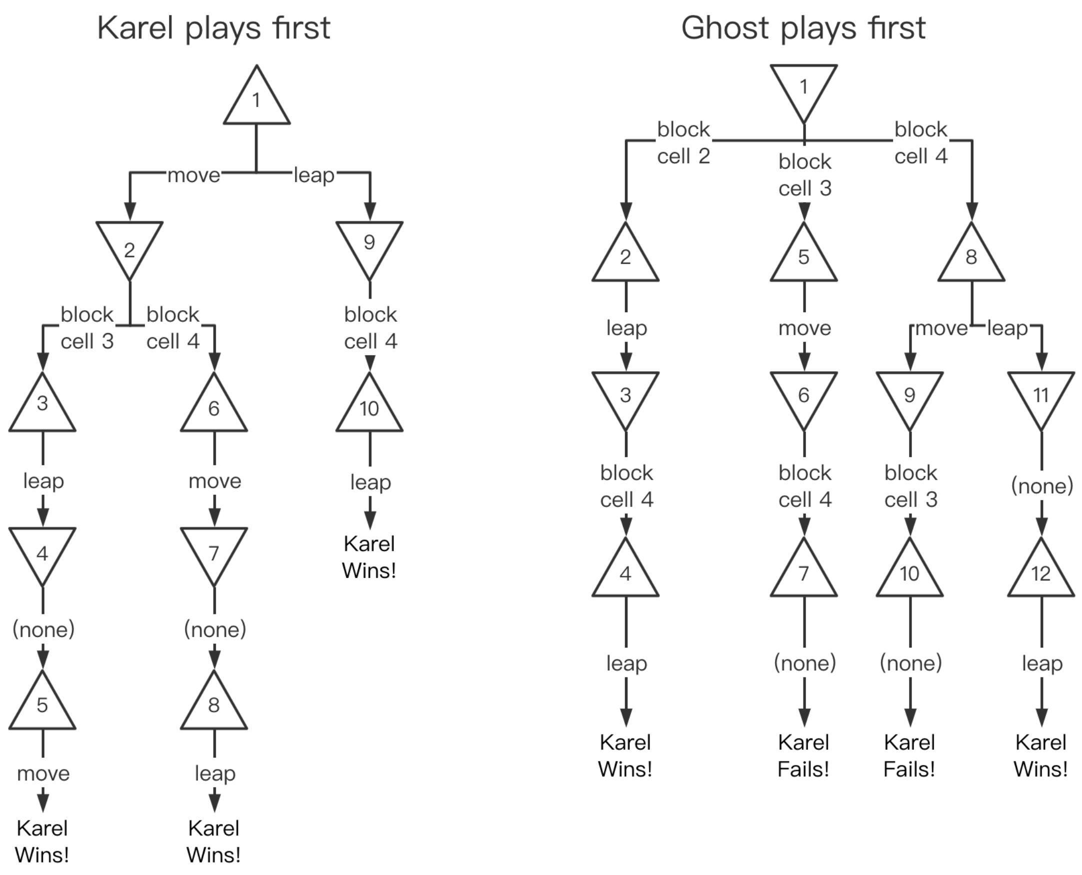


Figure 3: The unpruned game trees for (i) Karel plays first, and (ii) ghost plays first.

However, after we confirmed that node 2 leads to "Karel wins", since node 1 is Karel's turn (a max node), the branch under node 9 can be pruned because Karel can already win for sure by taking `move()` at the first step.

In (ii), after observing that node 2 leads to "Karel wins", since node 1 is the Ghost's turn (a min node), the branch under node 3 must be visited in case the ghost can win back; After we confirmed that node 3 leads to "Ghost wins", the branch under node 4 can be pruned because the ghost can already win for sure by taking "block cell 3" at the first step.

If you believe that any of the questions were ambiguous, please state your assumptions here.