CS221 Exam Solutions

CS221
Fall 2019

Name: _____
by writing my name I agree to abide by the honor code

SUNet ID: _____

Read all of the following information before starting the exam:

- This test has 3 problems printed on 28 pages and is worth 150 points total. It is your responsibility to make sure that you have all of the pages.
- Only the printed (top) side of each page will be scanned so write all your answers on that side of the paper.
- 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. Try to understand the problems intuitively; it really helps to draw a picture.
- You cannot use any external aids except one double-sided $8\frac{1}{2}$" x 11" page of notes.
- Good luck!

Problem	Part	Max Score	Score
1	a	15	
	b	10	
	c	10	
	d	15	
2	a	15	
	b	10	
	c	15	
	d	10	
3	a	10	
	b	15	
	c	10	
	d	15	

Total Score: ☐ + ☐ + ☐ = ☐

1. Under Attack (*50 points*)

You work for a cybersecurity company, Sanymtec, to monitor online forums. Recently, you've noticed an increasing number of curious sentences that look like this:

```
"stoafnrd scisetints fnid evenidce taht the erath is falt"
```

These sentences are perfectly readable by humans, but when you feed them into your machine learning models, they are totally confused and make wildly incorrect predictions. Your automatic fake news detection system is under attack!

a. (*15 points*)

As a first step, you would like to classify a sentence x as either adversarial ($y = 1$) or not ($y = -1$). Your boss doesn't want you to use the hinge loss because she's worried that the attacker might be able to more easily reverse engineer the system. So you decide to investigate alternative loss functions.

For each of the five loss functions $\text{Loss}(x, y, \mathbf{w})$ below:

- Determine whether $\text{Loss}(x, y, \mathbf{w})$ is usable for classification if we minimize the training loss with stochastic gradient descent (SGD).
- If the answer is yes, compute its gradient $\nabla_{\mathbf{w}}\text{Loss}(x, y, \mathbf{w})$.
- If the answer is no, explain in one sentence why it is not usable.

(i) [3 points] $\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \lfloor(\mathbf{w} \cdot \phi(x))y\rfloor, 0\}$, where $\lfloor a \rfloor$ returns a rounded down to the nearest integer.

Solution No, because the gradient is 0 almost everywhere, so we cannot apply gradient descent.

(ii) [3 points] $\text{Loss}(x, y, \mathbf{w}) = \max\{(\mathbf{w} \cdot \phi(x))y - 1, 0\}$.

Solution No, because for this loss function, a larger margin corresponds to a larger loss, which is the opposite of what we want.

(iii) [3 points]

$$\text{Loss}(x, y, \mathbf{w}) = \begin{cases} 1 - 2(\mathbf{w} \cdot \phi(x))y & \text{if } (\mathbf{w} \cdot \phi(x))y \leq 0 \\ (1 - (\mathbf{w} \cdot \phi(x))y)^2 & \text{if } 0 < (\mathbf{w} \cdot \phi(x))y \leq 1 \\ 0 & \text{if } (\mathbf{w} \cdot \phi(x))y > 1 \end{cases}$$

Solution Yes, and the gradient is:

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = \begin{cases} -2\phi(x)y & \text{if } (\mathbf{w} \cdot \phi(x))y \leq 0 \\ -2(1 - (\mathbf{w} \cdot \phi(x))y)\phi(x)y & \text{if } 0 < (\mathbf{w} \cdot \phi(x))y \leq 1 \\ 0 & \text{if } (\mathbf{w} \cdot \phi(x))y > 1 \end{cases}$$

Note: this loss function is a "smoothed" hinge loss, which is continuously differentiable everywhere (notice that the constants are chosen so that the gradients match at the different pieces), but still behaves like the hinge loss when the magnitude of the margin is large.

(iv) [3 points] $\text{Loss}(x, y, \mathbf{w}) = \begin{cases} \max(1 - (\mathbf{w} \cdot \phi(x)), 0) + 10 & \text{if } y = +1 \\ \max(1 + (\mathbf{w} \cdot \phi(x)), 0) - 10 & \text{if } y = -1 \end{cases}$.

Solution Yes, it turns out that this loss function has the exact same gradient as the hinge loss! The fact that the two labels have a different constant offset doesn't matter.

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & \text{if } (\mathbf{w} \cdot \phi(x))y \leq 1 \\ 0 & \text{otherwise.} \end{cases}.$$

Note that this is the simplified form of

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = \begin{cases} -\phi(x) & \text{if } (\mathbf{w} \cdot \phi(x)) \leq 1 \text{ and } y = +1 \\ \phi(x) & \text{if } (\mathbf{w} \cdot \phi(x)) \leq -1 \text{ and } y = -1 \\ 0 & \text{if } (\mathbf{w} \cdot \phi(x)) > 1 \text{ and } y = +1 \\ 0 & \text{if } (\mathbf{w} \cdot \phi(x)) > -1 \text{ and } y = -1. \end{cases}.$$

(v) [3 points] $\text{Loss}(x, y, \mathbf{w}) = \sigma(-(\mathbf{w} \cdot \phi(x))y)$, where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic function.

Solution Both answers are acceptable. One could argue that the loss is non-convex so that it is difficult to optimize (even if the gradient exists and is non-zero everywhere). Of course, non-convexity never stopped anyone, and given that this loss function is a more faithful approximation to the zero-one loss, one could make a case that this objective is reasonable. It's gradient is:

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = -\sigma(-(\mathbf{w} \cdot \phi(x))y)(1 - \sigma(-(\mathbf{w} \cdot \phi(x))y)\phi(x)y.$$

b. (*10 points*)

Your next job is to decide which features to use in order to solve the classification problem. Assume you have a set D of real English words.

(i) [5 points] Suppose you have a sentence x which is a string; e.g., $x =$ `"erath is falt"`. Write the Python code for taking a sentence x and producing a `dict` representing the following two feature templates:

1. x contains word _____
2. number of words in x that are *not* in D

Assume that words in x are separated by spaces; e.g., the words in x above are `"erath"`, `"is"`, `"falt"`.

```
def featureExtractor(x, D):
    phi = defaultdict(float)
```

```
    return phi
```

Solution The key names are arbitrarily chosen. The return value of "erath is falt" should be: {"num dict": 2, "contains erath": 1, "contains is": 1, "contains falt": 1}. One possible solution is shown below.

```
def featureExtractor(x, D):
   phi = defaultdict(float)
   for word in x.split(‘ ’):
      x[‘contains ’ + word] = 1
      if x not in D:
         x[‘num dict’] += 1
```

```
return phi
```

Due to ambiguity of the first feature template, solutions that add words in D with value 0 will also be accepted. Solutions that assumed the words in x are unique were also accepted.

Solutions that were not accepted:

- Solutions that assume a "given" word is being passed in
- Solutions that only consider words in D and not words like "erath"

(ii) [2 points] If k is the number of unique words that occur in the training set and $|D|$ is the number of words in the given set of real English words, what is the number of features that the linear classifier will have?

Solution There is one feature for every possible word in the training set, but only one feature that captures the number words in x that are in the dictionary. So the total is $\boxed{k+1}$.

Due to ambiguity of the first feature template, this part will be correct if it is consistent with the code written on the previous page. If the solution to part (i) adds words in D with value 0, then the answer $\boxed{n+|D|+1}$ will also be accepted where n is the number of unique words that occur in the training set but not in D.

Solutions that are not accepted:

- $k+|D|+1$ is incorrect because there can be words in the training set that are double counted in $|D|$
- |D| + 1 is incorrect (if the previous part is correct) because it only consider words in D and not words like "erath"

(iii) [3 points] Suppose that an insider leaks Sanymtec's classification strategy to the attackers. The classifier itself was not leaked, just the classification strategy behind it, which reveals that Sanymtec is using a dataset of adversarial sentences to train a classifier with the features defined in part (i).

The attackers then use this information to try modifying any fixed sentence (e.g., `"climate change is a hoax"`) into something readable by humans (e.g., `"clmaite canhge is a haox"`) but classified (incorrectly) as **non-adversarial** by Sanymtec. How can the attackers achieve this?

Solution The resulting classifier only knows about adversarial words that it has seen in the training set. Any words that don't appear in the training set will not contribute to any of the "x contains word _____" weights. However, the space of possible adversarial words is exponentially large, so given any sentence, the adversary can find a perturbation of each word that has not appeared in the training set. If the adversary then just ensures that x has a non-adversarial word (e.g., `"the"`) which likely has negative weight, then only this word will contribute to the classifier score to result in a non-adversarial prediction. Having a smaller ratio of unseen words to words in D can also help ensure that x has a negative weight.

Solutions that add an arbitrary type of obfuscation other than permutation are also accepted (e.g. adding characters: "climate" -> "cloimate").

Due to ambiguity of the question, solutions that mention the problem that attackers would need to know which adversarial permutations are used in Sanymtec's training set are accepted. However, solutions that do not take into account that some of the adversarial sentences could be weighted very positively by Sanymtec's classifier are penalized.

c. (*10 points*)

Having built a supervised classifier, you find it extremely hard to collect enough examples of adversarial sentences. On the other hand, you have a lot of non-adversarial text lying around.

(i) [3 points] Suppose you have a total of 100,000 training examples that consists of 100 adversarial sentences and 99,900 non-adversarial sentences. You train a classifier and get 99.9% accuracy. Is this a good, meaningful result? Explain why or why not.

Solution No, because you can get at least that level of accuracy by always predicting non-adversarial ($y = -1$), which is a trivial solution.

(ii) [3 points] You decide to fit a generative model to the non-adversarial text, which is a distribution $p(x)$ that assigns a probability to each sentence x. For simplicity, let's use a unigram model:

$$p(x) = \prod_{i=1}^{n} p_{\text{u}}(w_i),$$

where $w_1, w_2, ..., w_n$ are the words in sentence x, and $p_{\text{u}}(w)$ is a probability distribution over possible w's.

Suppose you are given a single sentence `"the cat in the hat"` as training data. Compute the maximum likelihood estimate of the unigram model p_{u}:

w	$p_{\text{u}}(w)$

Solution

w	$p_{\text{u}}(w)$
the	2/5
cat	1/5
in	1/5
hat	1/5

(iii) [4 points] Given an unseen sentence, your goal is to be able to predict whether that sentence is adversarial or not. You have a labeled dataset $\mathcal{D}_{\text{train}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and would like to use the unigram model to train your predictor.

How could you use $p(x)$ (from the previous problem) and $\mathcal{D}_{\text{train}}$ to obtain a predictor $f(x)$ that outputs whether a sentence x is adversarial ($y = 1$) or not ($y = -1$)? Be precise in defining $f(x)$. Hint: define a feature vector $\phi(x)$.

$$f(x) =$$

Solution Ideally, $p(x)$ should assign high probability to non-adversarial text and low probability to adversarial text. The only thing that's missing is the threshold, which can be tuned on $\mathcal{D}_{\text{train}}$. One way to do this is to define two features $\phi(x) = [1, p(x)]$, and fit a corresponding weight vector $\mathbf{w}$ using standard techniques (hinge loss + SGD). Then we simply predict

$$f(x) = \text{sign}(\mathbf{w} \cdot \phi(x)).$$

d. (*15 points*)

You notice that the adversarial words are often close to real English words. For example, you might see `"erath"` or `"eatrh"` as misspellings of `"earth"`. Furthermore, the actual number of adversarial words is rather small (it seems like the attacker just wants to reinforce the same messages). This makes you think of another unsupervised approach to try.

Let D be the set of real English words as before and $a_1, \ldots, a_n$ be the list of adversarial words you've found, and let $\text{dist}(a, e)$ be the number of edits to transform some adversarial word a to the English word e (how exactly distance is defined is unimportant).

We wish to choose K English words $e_1, \ldots, e_K \in D$ and assign each adversarial word a_i to one of the chosen English words ($z_i \in \{1, \ldots, K\}$). Each English word $e \in D$ incurs a cost $c(e)$ if we choose it as one of the K words. Our goal is to minimize the total cost of choosing $e_1, \ldots, e_K$ plus the total number of edits from adversarial words $a_1, \ldots, a_n$ to their assigned English words $e_{z_1}, \ldots, e_{z_n}$.

As an example, let $D = \{$`"earth"`, `"flat"`, `"scientists"`$\}$ with $c($`"earth"`$) = 1$, $c($`"flat"`$) = 1, c($`"scientists"`$) = 2$, and $a_1 =$ `"erath"`, $a_2 =$ `"falt"`, $a_3 =$ `"eatrh"`. Then with $K = 2$, one possible assignment (presumably the best one) is $e_1 =$ `"earth"`, $e_2 =$ `"flat"`, $z_1 = 1, z_2 = 2, z_3 = 1$.

(i) [3 points] Define a loss function that captures the optimization problem above:

$$\text{Loss}(e_1, \ldots, e_K, z_1, \ldots, z_n) =$$

Solution As the flavortext mentions, "our goal is to minimize the total cost of choosing $e_1, \ldots, e_K$ plus the total number of edits from adversarial words $a_1, \ldots, a_n$ to their assigned English words $e_{z_1}, \ldots, e_{z_n}$." The total cost of choosing $e_1, \ldots, e_K$ is $\sum_{j=1}^{K} c(e_j)$ and the total number of edits from adversarial words to their assigned English words is $\sum_{i=1}^{n} \text{dist}(a_i, e_{z_i})$.

$$\text{Loss}(e_1, \ldots, e_K, z_1, \ldots, z_n) = \sum_{j=1}^{K} c(e_j) + \sum_{i=1}^{n} \text{dist}(a_i, e_{z_i}).$$

(ii) [5 points] Derive an alternating minimization algorithm for optimizing the above objective. We alternate between two steps. In step 1, we optimize $z_1, \ldots, z_n$. Formally write down this update rule as an equation for each z_i where $1 \leq i \leq n$. What is the runtime? You should specify runtime with big-Oh notation in terms of n, K and/or $|D|$.

Solution For each adversarial word a_i, the goal is to find the cluster assignment, which can be done by finding the index of the English word with the closest edit distance to a_i. This can be found by iterating through all j and comparing every English word e_j to the current a_i. The index that minimizes this value is the desired cluster assignment.

$$z_i = \arg \min_{1 \leq j \leq K} \text{dist}(a_i, e_j).$$

For each of the n adversarial words, we loop over K clusters, so the runtime is $O(nK)$.

(iii) [5 points] In step 2, we optimize $e_1, \ldots, e_K$. Formally write down this update rule as an equation for each e_j where $1 \leq j \leq K$. What is the runtime? You should specify runtime with big-Oh notation in terms of n, K and/or $|D|$.

Solution For each cluster j, we want to find the best English word e_j to be the "centroid." To do this, we can iterate through every English word e and calculate the "cost" of that particular assignment. The "cost" of that particular assignment is defined as the cost of choosing e plus the sum of the edit distance between e with every adversarial word assigned j. The e with the minimum "cost" will be the new "centroid."

$$e_j = \arg \min_{e \in D} \left(c(e) + \sum_{i: z_i = j} \text{dist}(a_i, e) \right).$$

For each English word $e \in D$ and cluster j, we keep a running sum $S_{e,j}$ which is initialized to $c(e)$. Then for each adversarial word a_i and English word e, we increment S_{e,z_i}. Finally, for each j, we simply choose the e that minimizes $S_{e,j}$. The runtime is $O(n|D| + K|D|)$.

(iv) [2 points] Is the above procedure guaranteed to converge to the minimum cost solution? Explain why or why not. If not, what method what algorithm could you use with such guarantees?

Solution No, as this is a variant of k-means, it is only guaranteed to converge to a local optimum. To obtain the optimal solution, we could cast the problem as a factor graph and use backtracking search.

2. Maze (*50 points*)

One day, you wake up to find yourself in the middle of a corn field holding an axe and a map (Figure 1). The corn field consists of an $n \times n$ grid of cells, where some adjacent cells are blocked by walls of corn stalks; specifically, for any two adjacent cells (i, j) and (i', j'), let $W((i, j), (i', j')) = 1$ if there is a wall between the two cells and 0 otherwise. For example, in Figure 1, $W((1, 1), (1, 2)) = 0$ and $W((1, 2), (1, 3)) = 1$.

You can either move to an adjacent cell if there's no intervening wall with cost 1, or you can use the axe to cut down a wall with cost c without changing your position. Your axe can be used to break down at most b_0 walls, and your goal is to get from your starting point (i_0, j_0) to the exit at (n, n) with the minimum cost.

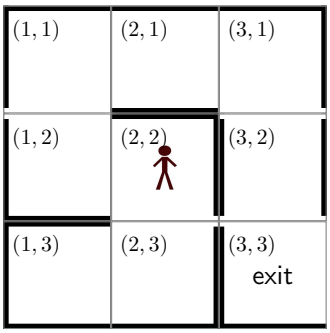

Figure 1: An example of a corn maze. The goal is to go from the initial location $(i_0, j_0) = (2, 2)$ to the exit $(n, n) = (3, 3)$ with the minimum cost.

a. (*15 points*)

(i) [10 points] Fill out the components of the search problem corresponding to the above maze.

- $s_{\text{start}} = ((i_0, j_0), b_0)$.
- $\text{Actions}(((i,j),b)) = \{a \in \{(-1,0),(+1,0),(0,-1),(0,+1)\} :$ $(i,j)+a$ is in bounds and $(W((i,j),(i,j)+a) = 0$ or $b > 0)\}$.
- $\text{IsEnd}(((i,j),b)) =$

  Solution $\text{IsEnd}(((i,j),b)) = [i = n \text{ and } j = n]$.

- $\text{Succ}(((i,j),b),a) =$

  Solution The new location is $(i,j)+a$, and we need to decrement our axe budget b whenever $W((i,j),(i,j)+a) = 1$.

  $$\text{Succ}(((i,j),b),a) = ((i,j)+a, b - W((i,j),(i,j)+a)) \tag{1}$$

  During exam we made clarifications that $c > 0$ and the cost c should be rephrased as the cost to break down a wall in a move (if there is a wall during a move, we take it down with extra cost c). Some students might not get our clarification during exam, so we also accept this solution in this and following questions:

  $$\text{Succ}(((i,j),b),a) = ((i,j)+(1-W((i,j),(i,j)+a))a, b - W((i,j),(i,j)+a))$$
  $$W((i,j),(i,j)+a) = 0$$

- $\text{Cost}(((i,j),b),a) =$

  Solution We always pay cost 1 for moving one cell and additionally, we pay c if we have to break down a wall.

  $$\text{Cost}(((i,j),n),a) = 1 + cW((i,j),(i,j)+a). \tag{2}$$

  We also accept this solution:

  $$\text{Cost}(((i,j),n),a) = (1 - W((i,j),(i,j)+a)) + cW((i,j),(i,j)+a). \tag{3}$$

(ii) [5 points] When you use an axe to take down a wall, the wall stays down but the set of walls which have been taken down are not tracked in the state. Why does our choice of state still guarantee the minimum cost solution to the problem?

Solution Under the current formulation, one needs to pay each time we go through a wall. However, the minimum cost path will never visit the same location twice (otherwise, there's a cycle which can be cut out to reduce the cost of the path). Therefore, there is no difference between paying only for the first time and paying for each time.

b. (*10 points*)

Solving the search problem above is taking forever and you don't want to be stuck in the corn maze all day long. So you decide to use A*.

(i) [5 points] Define a consistent heuristic function $h(((i,j),b))$ based on finding the minimum cost path using the relaxed state (i,j) where we assume we have an infinite axe budget and therefore do not need to track it. Show why your choice of h is consistent and what you would precompute so that evaluating any $h(((i,j),b))$ takes $O(1)$ time and precomputation takes $O(n^2 \log n)$ time.

Solution Define $h(((i,j),b))$ to be the minimum cost path from (i,j) to (n,n), where we have no budget constraint on the number of times we can use the axe. We can solve the relaxed search problem from the exit (n,n) to all locations (i,j). This can be done using UCS in $O(n^2 \log n)$ time. Then, we just store the lookup table for each of the $O(n^2)$ states to be queried during the search.

This problem only relaxes axe budget constraint, so any heuristic relaxing wall or cost related constraint (such as Manhattan distance to the end, which just remove all walls or set $c = 0$) will not be accepted.

(ii) [5 points] Noticing that sometimes h is the true future cost of the original search problem, you wonder when this holds more generally. For what ranges of b_0 and c would this hold? Assume for this part that there is a path that doesn't require breaking down any walls.

$\_\_\_\_\_ \leq b_0$ $\_\_\_\_\_ \leq c$

Your lower bounds need not be tight, but you need to formally justify why they hold.

Solution The heuristic is exactly the future cost when the number of axe uses of the original and relaxed search problems are identical. This happens in two cases:

1. When the budget constraint b_0 is large enough, then there's effectively no constraint on the number of axe uses. This happens when b_0 is at least the largest number of walls that need to be broken down. In worse case we only need to break down $2n$ walls (otherwise we can directly head to the exit with less cost).

2. When c is large enough, then it is not worth breaking down any walls at all. This happens when $c \geq n^2$, an upper bound on the length of the minimum cost path without breaking down walls (which exists by assumption).

c. (*15 points*)

Having solved the search problem above, you are eager to set out on your journey through the maze, but you realize that breaking down corn stalks is harder than you thought. Suppose that each attempt to break down a wall has an $\epsilon > 0$ probability of failing. Recall that b_0 is the maximum number of walls you can break down, not the number of attempts, and each attempt to break down a wall has cost c.

(i) [5 points] Suppose that each attempt to break down a wall is independent (e.g., if you fail once, the next attempt at the same wall also has probability ϵ of failing regardless of your previous failures). You are interested in minimizing the expected cost of exiting the maze. While the natural solution is to treat this as an MDP, it turns out you can still cast this problem as a search problem. In particular, define a modified $\text{Cost}(((i,j),b),a)$ function, and write one sentence about why this choice gives you the optimal policy.

Solution Since a failed attempt to bring down a wall leaves you in the same state, the optimal policy will either keep on trying to break down a wall or not try at all (recall the dice game from lecture). Therefore, we can interpret an action a as saying: repeatedly break down any wall in direction a and move in that direction. The expected total cost T of such an action is given by the recurrence $T = c + \epsilon T$, which has a solution $T = \frac{c}{1-\epsilon}$, so:

$$\text{Cost}(((i,j),b),a) = 1 + \left(\frac{c}{1-\epsilon}\right) W((i,j),(i,j)+a). \tag{4}$$

We also accept:

$$\text{Cost}(((i,j),b),a) = \left(\frac{c}{1-\epsilon}\right) W((i,j),(i,j)+a). \tag{5}$$

(ii) [5 points] Suppose instead that each attempt to break down a wall is perfectly dependent (e.g., if you fail once, you will always fail to break down that wall). Let us model this problem as an MDP. What should the states of the MDP be? What is the number of states in the worst case as a function of b_0 and n (use big-Oh notation)? In this problem suppose $b_0 << n$.

Solution Now when you fail to break down a wall, you have to remember that fact so you don't try again. Therefore your state must include all the walls that you've tried to break down that have failed (F) and all the walls where you have succeeded (S), since you might need to revisit locations. You have at most $2n^2 - 2n$ total walls and you need to keep track of at most b_0 in S, and possibly all $2n^2 - 2n$ in F; the number of such S is $\sum_{i=0}^{b_0} \binom{2n^2-2n}{i}$ and that of F is 2^{2n^2-2n}. There are still n^2 possible locations. We don't need to store the budget left since this can be tracked by the size of set S. The number of states is therefore $O\left(n^2 \left(\sum_{i=0}^{b_0} \binom{2n^2-2n}{i}\right) 2^{2n^2-2n}\right)$.

Another solution is to label each wall as one of succeeded, failed or not attempted. This gives $O\left(n^2 3^{2n^2-2n}\right)$.

We give full credit for both solutions.

(iii) [5 points] If the probability of successfully breaking down a wall is $(1-\epsilon)/k$, where $k > 0$ is the number of times you've tried to break down a wall. What should the states of the MDP be now?

Solution Now you have to keep track of how many times you've tried to break down each wall, so that at any point in time, you know the failure probability of each wall. The state should be same as (ii) but F is now a counter instead of a set.

Another interpretation is to take k as shared for all walls. In this context the state should be current location, budget of axe, k and set of broken walls S. Both interpretations are acceptable.

d. (*10 points*)

Let's actually solve the maze! In this specific 3×3 maze as shown in Figure 1, the initial location is $(2, 2)$ and the exit is $(3, 3)$. For simplicity assume that $b_0 = 1$ and that your axe always succeeds ($\epsilon = 0$).

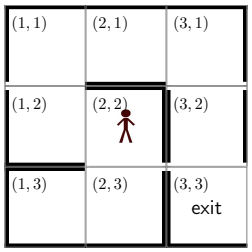

Figure 2: Same corn maze from Figure 1, repeated for convenience.

(i) [5 points] Compute the minimum achievable cost as a function of c.

Solution There are three solutions:

1. Go around the walls: (2,2)–(1,2)–(1,1)–(2,1)–(3,1)–(3,2)–(3,3), which has cost 6.
2. Break down one wall: (2,2)–(3,2)–(3,3), which both have cost $2 + c$.
3. Break down the other wall: (2,2)–(2,3)–(3,3), which both have cost $2 + c$.

Therefore, the minimum cost is $\min(2 + c, 6)$. We also accept solutions that consistent with their definitions in **a**.

(ii) [5 points] Let's look at the optimal policy at the initial location. For each value of c, what are corresponding optimal actions? If there is a tie between optimal actions state all of them. Your answer should consist of statements of the form: if $c \in$ _____, then the optimal actions are _____.

Solution Note that the three solutions above correspond to action 1 (west), action 2 (east), and action 4 (south), respectively. There are three cases:

- If $2 + c < 6$ or $c \in (0, 4)$, then optimal actions are 2 and 4.
- If $2 + c = 6$ or $c = 4$, then optimal actions are 1, 2 and 4.
- If $2 + c > 6$ or $c \in (4, \infty)$, then optimal action is 1.

We also accept solutions that consistent with their definitions in **a**.

3. Faulty Accumulator (*50 points*)

You decide to try your hand at building hardware. Specifically, you will build a simple circuit that takes n numbers and incrementally computes their sum. However, it turns out hardware is hard, and in your first attempt, the accumulator occasionally gets zeroed out randomly.

To capture this precisely, we can define the following generative model whose Bayesian network is shown in Figure 3. Let $Y_0 = 0$ be the initial sum. For each time step $i = 1, \ldots, n$, the circuit:

1. Receives an input number X_i chosen uniformly from $\{1, 2, 3, 4\}$.
2. Decides to remember ($R_i = 1$) with probability $1-\epsilon$ or forget ($R_i = 0$) with probability ϵ.
3. Computes the running sum: $Y_i = R_i Y_{i-1} + X_i$, where Y_{i-1} is added depending on R_i.

As an example:

1. $X_1 = 3$, $R_1 = 1$, $Y_1 = 3$ (remember)
2. $X_2 = 2$, $R_2 = 0$, $Y_2 = 2$ (forget)
3. $X_3 = 4$, $R_3 = 1$, $Y_3 = 6$ (remember)
4. $X_4 = 4$, $R_4 = 1$, $Y_4 = 10$ (remember)

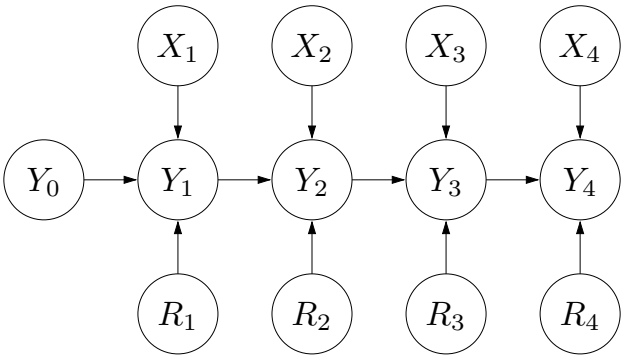

Figure 3: Bayesian network corresponding to the faulty accumulator.

a. (*10 points*)

To speed things up, you want to first prune the domains of variables. Recall that when we enforce arc consistency on a variable A with respect to a factor f, we keep a value v in the domain of A if and only if there exist values for other variables in the scope of f such that f evaluates to a non-zero number.

(i) [5 points] What is the domain of Y_n as a function of n?

Solution The value Y_n is the sum over n numbers, each of which can be up to 4, so the domain of Y_n is $\boxed{\{1, \ldots, 4n\}}$.

(ii) [5 points] Consider the following factor, where we have marginalized out R_2:

$$p(y_2 \mid y_1, x_2) = \epsilon p(y_2 \mid y_1, x_2, r_2 = 0) + (1 - \epsilon) p(y_2 \mid y_1, x_2, r_2 = 1). \tag{6}$$

Suppose $Y_1 \in \{1, 2\}$ and $Y_2 = 3$. What is the domain of X_2 after enforcing arc consistency on X_2?

Solution If we remember $(R_2 = 1)$, then X_2 must be $Y_2 - Y_1 \in \{1, 2\}$. If we forget $(R_2 = 0)$, X_2 must be $Y_2 = 3$. So the resulting domain of X_2 is $\boxed{\{1, 2, 3\}}$.

b. (*15 points*)

Now, disregarding what was done during part a, let us explore how conditioning on evidence changes our beliefs about X_2.

(i) [5 points] Compute:

x_2	$\mathbb{P}(X_2 = x_2)$
1	
2	
3	
4	

Solution We marginalize all other variables, which are non-ancestors of X_2, to get the prior distribution, which is uniform as defined:

x_2	$\mathbb{P}(X_2 = x_2)$
1	1/4
2	1/4
3	1/4
4	1/4

(ii) [5 points] Suppose we observe that $Y_2 = 3$. Now what do we believe about X_2?

x_2	$\mathbb{P}(X_2 = x_2 \mid Y_2 = 3)$
1	
2	
3	
4	

Solution First note that $Y_1 = X_1$ deterministically, which has a uniform distribution over $\{1, 2, 3, 4\}$. Next, let us write out the conditional distribution:

$$\mathbb{P}(X_2 = x_2 \mid Y_2 = 3) \propto \mathbb{P}(X_2 = x_2, Y_2 = 3) = \sum_{y_1, r_2} p(y_1)p(r_2)p(x_2)p(y_2 = 3 \mid y_1, x_2, r_2). \quad (7)$$

Now we compute the RHS for each value of x_2.

x_2	$\mathbb{P}(X_2 = x_2, Y_2 = 3)$	$\mathbb{P}(X_2 = x_2 \mid Y_2 = 3)$
1	$\frac{1}{4} \cdot (1 - \epsilon) \cdot \frac{1}{4} \cdot 1$	$\frac{1-\epsilon}{2+2\epsilon}$
2	$\frac{1}{4} \cdot (1 - \epsilon) \cdot \frac{1}{4} \cdot 1$	$\frac{1-\epsilon}{2+2\epsilon}$
3	$1 \cdot \epsilon \cdot \frac{1}{4} \cdot 1$	$\frac{4\epsilon}{2+2\epsilon}$
4	0	0

Note that for $X_2 \in \{1, 2\}$, we must have had remembered ($R_2 = 1$), which forces $Y_1 = Y_2 - X_2$ (has probability $\frac{1}{4}$). For $X_2 = 3$, we must have forgotten ($R_2 = 0$), and Y_1 is free to be anything (has probability 1). Then normalize the probabilities.

(iii) [5 points] Suppose we observe $Y_2 = 3$ and $Y_1 = 2$. Compute

x_2	$\mathbb{P}(X_2 \mid Y_2 = 3, Y_1 = 2)$
1	
2	
3	
4	

Solution The solution follows the same calculation as in the previous part, but where we zero out terms where $Y_1 \neq 2$.

x_2	$\mathbb{P}(X_2 = x_2, Y_2 = 3, Y_1 = 2)$	$\mathbb{P}(X_2 = x_2 \mid Y_2 = 3, Y_1 = 2)$
1	$\frac{1}{4} \cdot (1 - \epsilon) \cdot \frac{1}{4} \cdot 1$	$1 - \epsilon$
2	0	0
3	$\frac{1}{4} \cdot \epsilon \cdot \frac{1}{4} \cdot 1$	ϵ
4	0	0

c. (*10 points*)

Suppose you wish to compute the posterior distribution over all other variables given $Y_1 = 3, Y_2 = 2, Y_3 = 6, Y_4 = 10$. You're getting tired of doing probabilistic inference by hand, so you decide to implement Gibbs sampling to do it. Suppose you start out with the following configuration:

i	1	2	3	4
X_i	3	2	4	4
Y_i	3	2	6	10
R_i	1	0	1	1

(i) [3 points] Compute the Gibbs sampling update for

$$\mathbb{P}(X_2 \mid \text{everything else}) = \mathbb{P}(X_2 \mid X_1, X_3, X_4, Y_1, \ldots, Y_4, R_1, \ldots, R_4) = \tag{8}$$

Solution Given all other variables, X_2 is deterministic. In other words,

$$\mathbb{P}(X_2 = 2 \mid \text{everything else}) = 1.$$

(ii) [3 points] Compute the Gibbs sampling update for

$$\mathbb{P}(Y_2 \mid \text{everything else}) = \mathbb{P}(Y_2 \mid X_1, \ldots, X_4, Y_1, Y_3, Y_4, R_1, \ldots, R_4) = \tag{9}$$

Solution Given all other variables, Y_2 is deterministic. In other words,

$$\mathbb{P}(Y_2 = 2 \mid \text{everything else}) = 1.$$

(iii) [4 points] What is the problem with running Gibbs sampling on this Bayesian network? What alternative would you suggest?

Solution In order for Gibbs sampling to work, we must be able to reach any setting of variables from any other one. Because the Gibbs updates are mostly deterministic, we are stuck in this setting. One solution would be first marginalize out the variables $R_1, \ldots, R_n$, in which case the posterior over $X_1, \ldots, X_n$ would factor in to n separate distributions can be sampled independently and exactly. Particle filtering would work too as a default because of the sequential structure of the problem, but would be less efficient.

d. (*15 points*)

You are embarrassed to realize that not only is the circuit faulty, but also you have no clue how faulty it is (what ϵ is). You decide to estimate ϵ from data. Suppose you observe the following variables:

i	1	2	3	4
X_i	3	2	4	4
Y_i	3	2	6	10

In particular, you *do not* observe $R_1, \ldots, R_4$. Your goal is to find the ϵ which maximizes the marginal likelihood of the observed data. Let's use the EM algorithm.

(i) [5 points] Initialize $\epsilon = \epsilon_0$. For the E-step, compute the posterior:

$$\mathbb{P}(R_1, R_2, R_3, R_4 \mid X_1 = 3, X_2 = 2, X_3 = 4, X_4 = 4, Y_1 = 3, Y_2 = 2, Y_3 = 6, Y_4 = 10)$$

Solution The data provides no information about R_1, so the posterior is identical to the prior distribution over R_1, which is 0 with probability ϵ_0 and 1 with probability $1 - \epsilon_0$. The other three variables are completely determined from the data: $R_2 = 0$, $R_3 = 1$, $R_4 = 1$.

(ii) [5 points] For the M-step, use the posterior above to compute the updated value of ϵ (which should be a function of ϵ_0).

Solution There are $\epsilon_0 + 1$ fractional counts towards forgetting (0) and $(1 - \epsilon_0) + 2$ fractional counts towards remembering (1). Normalizing, we get $\epsilon = \boxed{\dfrac{\epsilon_0 + 1}{4}}$.

(iii) [5 points] Compute what ϵ converges to as you run more iterations of EM. Justify your answer mathematically.

Solution In each iteration, we take ϵ and return $\frac{\epsilon+1}{4}$. We can compute the fixed point (what EM converges) by solving the equation $\epsilon = \frac{\epsilon+1}{4}$, which yields $\epsilon = \boxed{\dfrac{1}{3}}$.

(page left blank for scratch work)

(page left blank for scratch work)