Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Autumn Semester 2023 | Prof. R. Wattenhofer

Computational Thinking

Saturday, January 27, 2024, 11:00-13:00

Do not turn over until instructed to do so

This examination lasts for 120 minutes and comprises 120 points. There is one block of questions for each lecture chapter.

You may answer in German, English, or combine German and English.

Unless explicitly stated, you do **not** have to justify your answers. Writing down your thoughts (math, text, or annotated sketches), however, might help with the understanding of your approach. This may then result in points being awarded even if your answer is not correct. Please write legibly. Unreadable answers will not be graded.

Some questions will ask you to fill in answers in a template. If you decide to start over you will find fill-in replacements at the end of the examination booklet.

Please write your name and student number on every additional sheet. Please write your name and student number in the following fields on this cover sheet.

Family Name	**First Name**	**Student Number**

Task	**Achieved Points**	**Maximum Points**
1 - Algorithms		19
2 - Complexity		21
3 - Cryptography		17
4 - Databases		15
5 - Machine Learning		18
6 - Neural Networks		16
7 - Computability		14
Total		**120**

1 Algorithms (19 points)

You are on a road trip with your sibling that will take you from Zurich to Stockholm, a distance of 2000 km. You want to optimize the total cost of charging your electric car for this trip, so you plan to determine how much charge to get at the stations you pass. All locations are specified as the distance traveled from Zurich.
There are $n + 1$ stations along the route. You have compiled a list of stations available at positions $\mathbf{p} = (p_0, p_1, \ldots, p_n)$ (the list is sorted in increasing order with $p_0 = 0$ and $p_n = 2000$), and you have the charging costs for each station $\mathbf{c} = (c_0, c_1, \ldots, c_n)$ in CHF/kWh. The charging stations are right next to the road, so it is not necessary to take a detour to reach them. Let the list $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ denote the amount the battery is charged at the stations in Wh, with the x_i being non-negative real numbers.
You start the journey with an empty battery at the first charging station and end with an empty battery at the end of the journey, i.e., after driving 2000 km. Your car uses 150 Wh/km and the battery has a capacity of 60 kWh.

a) `[3 points]` Your sibling argues that the most cost-efficient strategy always either skips a station or completely charges the battery. Is this true? Explain why.

b) `[8 points]` Write a **linear program** that given $\mathbf{p}$ and $\mathbf{c}$ minimizes the total cost over the variables $\mathbf{x} = (x_0, \ldots, x_n)$. You can assume that the stations are close enough for a solution to exist. Please formally define any additional variables you introduce.

c) `[2 points]` Does there exist a polynomial-time algorithm that solves the problem in question **b)**? Explain your answer.

d) `[2 points]` In question **b)** we assumed that the stations are close enough for a solution to exist. How will your LP solver tell you if this is not the case?

- ○ "Unbounded"
- ○ "Infeasible"
- ○ The solver will not terminate
- ○ (None of the above)

e) `[4 points]` Oh no! You accidentally passed by station i and forgot to charge x_i of electricity. Luckily, you made it to station $i + 1$ anyway.

Your sibling argues that simply adding $x_i = 0$ to your linear program and then running the linear program again will still produce the best possible solution given the situation. Say why this is incorrect (in one sentence) and propose a real fix.

Solution

a) Station $i + 1$ could be super cheap, but you need to charge at the more expensive station i to get there, but you certainly don't want to charge fully.

b) Recall that x_i is the amount of charge you get at the station i.

$$
\begin{aligned}
\min \sum_{i=0}^{n} c_i x_i \\
\forall i \in \{0, 1, \ldots, n\} : x_i \geq 0 \\
x_n = 0 \\
\forall k \in \{0, 1, \ldots, n\} : \sum_{i=0}^{k} x_i - 150 p_k \leq 60000 \\
\forall k \in \{0, 1, \ldots, n-1\} : \frac{1}{150} \sum_{i=0}^{k} x_i \geq p_{k+1}
\end{aligned}
$$

The second to last line ensures that the charging levels cannot exceed the battery capacity, while the last line ensures that we can always drive to the next station.

c) Yes, the LP defined in the previous question is such a polynomial-time algorithm.

d) If the stations are too far apart, then we will not be able to drive from one station to the next. The LP solver explicitly states that the problem is *infeasible.*

e) No, this would not give the correct solution, as it might change the variables $x_j, j = 0, 1, \ldots, i - 1$. We need to fix x_j for all stations we have passed, that is, $j \leq i$.

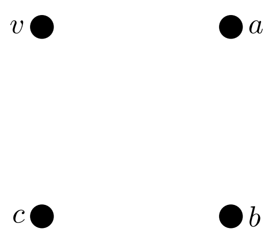
2 Complexity (21 points)

The Traveling Salesperson Problem with distances 1 and 2 (TSP12) is TSP restricted to complete graphs with edge lengths in $\{1, 2\}$, i.e., where there is an edge between every pair of nodes with a length of either 1 or 2. Consider the following pseudocode for solving TSP12, which returns a list of nodes in the order they should be visited, repeating the starting node to complete the cycle.

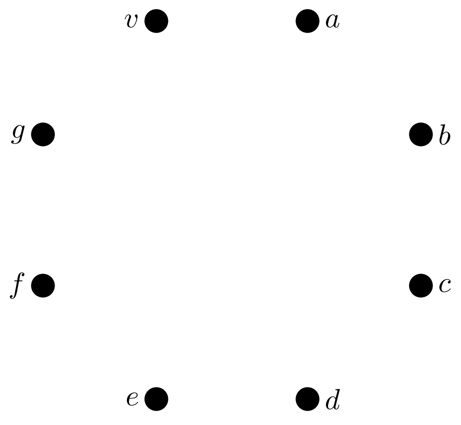
```
1  def greedyTSP(G, v):     # Graph G, Starting node v
2      V = V(G)-{v}         # V(G) = set of nodes of G, size(V) > 0
3      TSP = [v]            # TSP = solution
4      x = v
5      while V is nonempty:
6          Pick an arbitrary node u in V that is closest to x
7          TSP = TSP.append(u)
8          V = V-{u}
9          _______________
10     TSP = TSP.append(v)
11     return TSP
```

a) [4 points] Fill in the missing line 9 so that greedyTSP returns a valid tour and explain why greedyTSP returns a valid tour.

b) [4 points] Complete the graph below to give an example where greedyTSP gives an approximation ratio of 5/4. Provide the optimum tour and the output of greedyTSP. The starting node for greedyTSP is indicated by the v. (Hint: Notice the word "arbitrary" in Line 6 of the algorithm.)



c) `[4 points]` Complete the graph below to give an example where greedyTSP gives an approximation ratio of 11/8. You should only draw edges of length 1; omitted edges are presumed to be of length 2. Provide the optimum tour and the output of greedyTSP.



The examples in parts **b)** and **c)** can be further improved to show that greedyTSP gives an approximation ratio of $\frac{3}{2}$ at best. You will now prove the other side — that greedyTSP always achieves this approximation ratio. To help you, we split the proof into two steps.

d) `[5 points]` Suppose the optimal tour contains k edges of length 1. Prove that greedyTSP always returns a tour that uses at least $k/2$ edges of length 1. For simplicity, you may assume that n and k are even.

e) `[4 points]` Assuming part **d)**, complete the proof to show that the approximation ratio of greedyTSP is at most 3/2. Again, you may assume that n and k are even.

Solution

a) $x = u$. The while loop ensures that all nodes will be listed without duplicates (visited nodes are removed in line 8). Line 10 ensures the tour is completed by returning to the starting node.

b) See Figure 1. The approximation ratio is $ALG/OPT = 5/4$.

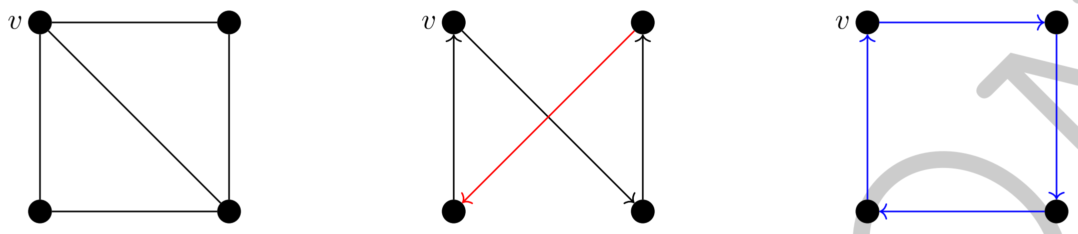


Figure 1: (left) Solution. Drawn edges indicate edges of length 1. Edges of length 2 are not drawn. (middle) A possible tour returned by greedyTSP of length $ALG = 5$. Red edges indicate edges of length 2. (right) Optimal tour of length $OPT = 4$.

c) See Figure 2. The approximation ratio is $ALG/OPT = 11/8$.

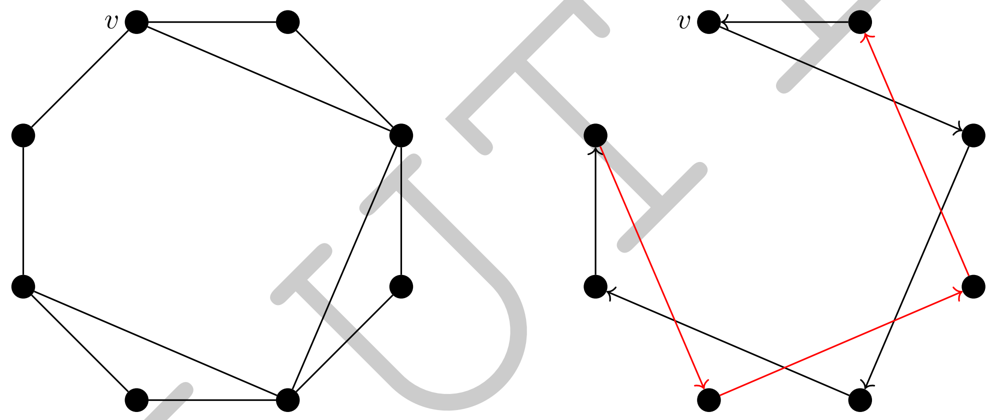


Figure 2: (left) Possible solution. Edges indicate edges of length 1. Edges of length 2 are not drawn. (right) A possible tour returned by greedyTSP of length $ALG = 11$. Red edges indicate edges of length 2. The optimal tour goes around the perimeter and has length $OPT = 8$.

(Note that 12/8 is also possible!)

d) Call an edge of length 1 a 1-edge. Notice that for each 1-edge (u, v), the algorithm visits the nodes u and v in some order; without loss of generality, suppose the algorithm visits u first and then v. Then, at least one 1-edge will be available when the algorithm reaches u, and a 1-edge will (have to) be taken.

The optimum tour uses k 1-edges, which are traversed in some order. Note that taking every second 1-edge gives a set of pairwise-disjoint edges (no common vertices) since no vertices in a tour repeat. Since k is assumed to be even, there are at least $\frac{k}{2}$ disjoint 1-edges.

e) Taken together with the previous argument, this means that the algorithm will use at least $\frac{k}{2}$ 1-edges giving a total cost of $ALG \leq 2n - \frac{k}{2}$, and for the optimum, we have $OPT = 2n - k$. Thus we have the approximation ratio:

$$\frac{ALG}{OPT} \leq \frac{2n - k/2}{2n - k}$$

This ratio is increasing in k for $0 \leq k \leq n$ since we have $2n$ as both the numerator and the denominator and we subtract more from the denominator than from the numerator. It is sufficient to observe this, but it can be shown formally by differentiation. The maximum is attained when $k = n$, giving an approximation ratio of $3/2$.

3 Cryptography (17 points)

Authentication protocol

Consider the following simplified authentication protocol for a cash withdrawal system, where account owner Alice authenticates to bank Bob in order to authorize a withdrawal from her account:

Setup:

- Select public parameters $p, g \in \mathbb{N}$.
- Alice and Bob sample uniformly random secret keys $a, b \in \mathbb{Z}_p$ and calculate their public keys $A := g^a \mod p$ and $B := g^b \mod p$ respectively.
- Alice receives Bob's public key B from a trusted source.

Authentication:
In the following f and g are yet to be instantiated functions.

- Alice sends her public key A along with the amount $v \in \mathbb{Z}_p$ she wants to withdraw.
- Bob picks and sends random value $r_B \in \mathbb{Z}_p$ along with $x := f(r_B, b)$.
- If $g(x, r_B, B) = 0$, Alice aborts.
- Alice calculates $c := v + r_B \mod p$ and sends $y := f(c, a)$.
- If $g(y, c, A) = 1$, Bob is convinced Alice authorized the withdrawal of amount v.

a) `[4 points]` Which of the following are reasonable instantiations for functions f and g?

○ True ○ False f is ElGamal encryption and g is ElGamal decryption.

○ True ○ False f is ElGamal decryption and g is ElGamal encryption.

○ True ○ False f is ElGamal signing and g is ElGamal verification.

○ True ○ False f is ElGamal verification and g is ElGamal signing.

b) [8 points] Which of the following attacks would become possible for an eavesdropper, Eve, if r_B was a fixed secret value Alice and Bob agreed on beforehand, instead of being randomly picked by Bob in every interaction? For each option, write one sentence to justify your answer.

○ True ○ False Eve could resubmit a previous authorization for a withdrawal Alice made to Bob, thus withdrawing the same amount again.

○ True ○ False Eve could alter value v and withdrawal authorization y before they reach Bob, such that Alice withdraws a different amount for which Eve previously observed a withdrawal.

○ True ○ False Eve could alter value v and withdrawal authorization y before they reach Bob, such that Alice withdraws an arbitrarily chosen different amount.

○ True ○ False Eve could authorize arbitrary new withdrawals in Alice's name, even after Alice went offline.

c) [5 points] Alice proposes changing the calculation of c to be $c := v + r_A + r_B \mod p$, where r_A is an additional value randomly picked by Alice and then sent alongside y. This change would inadvertently break the protocol. Show how an eavesdropper, Eve, could impersonate Alice, after observing just a single interaction between Alice and Bob.

Solution

a) False, False, True, False.

We can instantiate f with ElGamal signing and g with the corresponding signature verification as this provides authentication and non-repudiation with respect to Alice's key.

b) True, c is always the same for the same v, thus previous y remains valid.

True, in principle works the same as the first case.

False, in general this would require Eve to create a valid y for a c she has not seen.

False, in general this would require Eve to create a valid y for a c she has not seen.

c) When receiving a new challenge c'_B from Bob, Eve can calculate $c'_A := c - c'_B \mod p$ and send it and y to Bob, where c and y are the values observed by Eve in the honest interaction between Alice and Bob. The authentication succeeds as the value for c is the same and thus the authentication proof y remains valid.

4 Databases (15 points)

Key-value storage

a) Consider 30 key-value pairs with distinct integer keys that are supposed to be stored. For each of the following, write down the smallest and largest possible value.

- `[2 points]` The **maximum** depth of a key in a binary search tree (storing this data).

  smallest: _______ largest: _______

- `[2 points]` The size of a bucket with the **most** elements in a hash table.

  smallest: _______ largest: _______

- `[5 points]` The size of a bucket with the **most** elements in a hash table that uses the hash function $h(x) = x^2 \mod 10$.

  smallest: _______ largest: _______

JOIN types

b) `[6 points]` Imagine two tables A and B (which are both non-empty and do not contain any missing values) being joined using different SQL join types. Which of the following statements are true for all tables A and B? For each option, write one sentence to justify your answer.

○ True ○ False `FULL OUTER JOIN` returns strictly more rows than `RIGHT OUTER JOIN`.

○ True ○ False If `LEFT OUTER JOIN` returns strictly more rows than `RIGHT OUTER JOIN`, then `INNER JOIN` returns strictly fewer rows than `FULL OUTER JOIN`.

○ True ○ False The sum of the numbers of rows returned by `INNER JOIN` and `FULL OUTER JOIN` equals the sum of the number of rows returned by `LEFT OUTER JOIN` and `RIGHT OUTER JOIN`.

Solution

a) smallest: 4, largest: 29
smallest: 1, largest: 30
smallest: 5, largest: 30 (There are 6 buckets that this hash function can fill: 0,1,4,5,6,9)

b) **False:** If all rows in A have a matching row in B, then `FULL OUTER JOIN` and `RIGHT OUTER JOIN` return the same number of rows.

True: There must exists rows in A without matching rows in B, and these will be included in `FULL OUTER JOIN` but not in `INNER JOIN`.
Alternatively: `INNER JOIN` $\leq$ `RIGHT OUTER JOIN` $<$ `LEFT OUTER JOIN` $\leq$ `OUTER JOIN`

True: While all matched row pairs appear in all four joins, all unmatched rows of A appear in `FULL OUTER JOIN` and `LEFT OUTER JOIN`, and all unmatched rows of B appear in `FULL OUTER JOIN` and `RIGHT OUTER JOIN`.

5 Machine Learning (18 points)

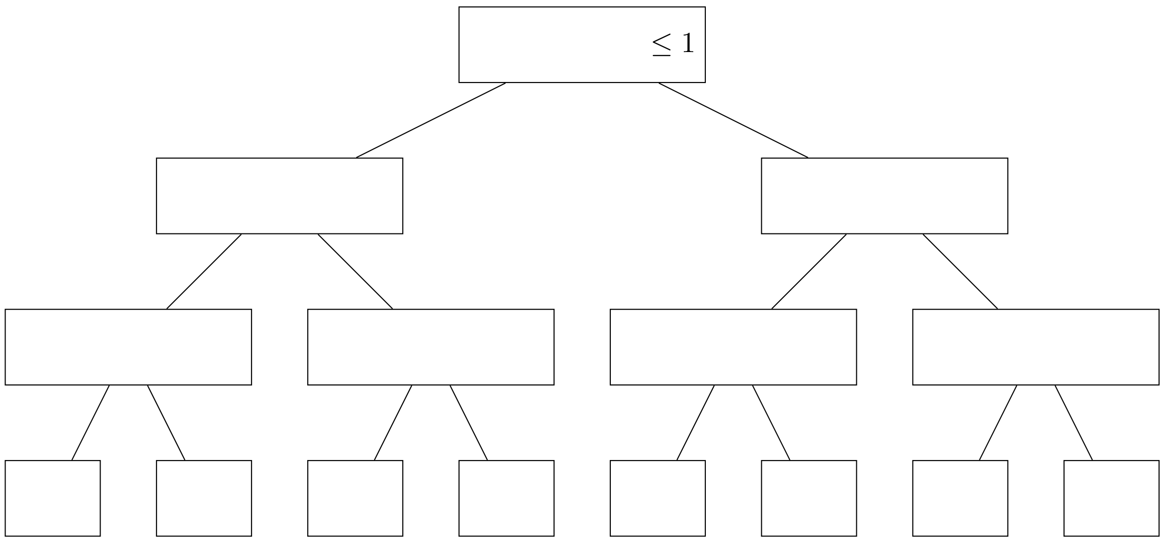
a) `[5 points]` You are given a non-empty set of points of class 0 and a non-empty set of points of class 1 with three-dimensional input features and you want to classify them correctly. Your goal is to achieve this by using linear regression instead of logistic regression. What constraint(s) need to hold so that your **linear regression** model outputs **exactly** the correct value 0 or 1 for each data point? Mark all necessary conditions.

- ○ All points need to be equidistant to each other.
- ○ The feature in one dimension needs to be the same for all points.
- ○ Two points with the same features need to belong to the same class.
- ○ The points in class 0 and the points in class 1 each form a plane, and the planes need to be distinct and parallel to each other.
- ○ All points need to lie in a plane that is parallel to at least one coordinate axis.
- ○ Either class 0 or class 1 can only contain one point.

b) `[7 points]` You are now given some data in the form of $\boldsymbol{p}_i = [x_1, x_2, x_3, y]^T$.

$$\boldsymbol{p}_1 = [1,1,4,0]^T, \boldsymbol{p}_2 = [4,4,3,0]^T$$
$$\boldsymbol{p}_3 = [2,1,3,1]^T, \boldsymbol{p}_4 = [4,2,1,1]^T$$
$$\boldsymbol{p}_5 = [2,4,4,1]^T, \boldsymbol{p}_6 = [2,3,1,0]^T$$

Fill in splitting rules for the following decision tree of our data found by minimizing the CART loss with mean absolute error (MAE) for impurity. If a node becomes a leaf, just write the corresponding class. Cross out unneeded parts of the tree. Write your condition in the form *feature* $\leq$ *threshold*, where the left child will be taken if the condition holds. Note that the threshold for the first split is given.



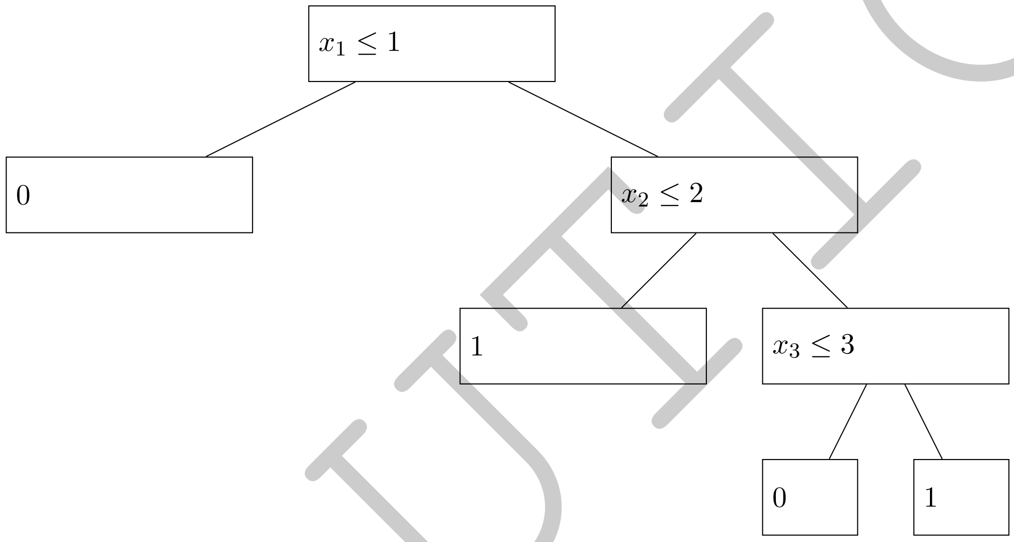
c) [6 points] Feature modeling is a method to enable a linear model to learn a non-linear relationships between $\boldsymbol{x}$ and y. For our data in **b)**, what would be a feature $\hat{f}$ of the input that enables a logistic regression model to correctly separate the classes, given that the function $\hat{f}$ needs to be continuous?

Solution

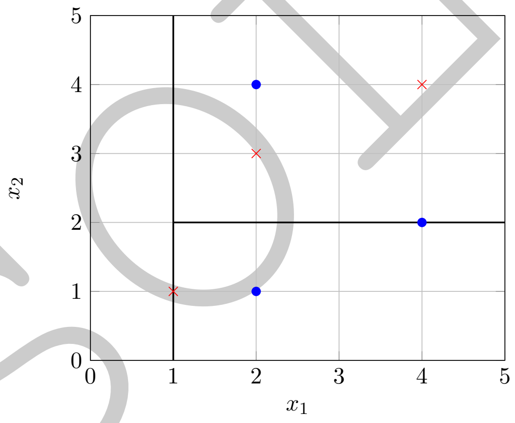
a)

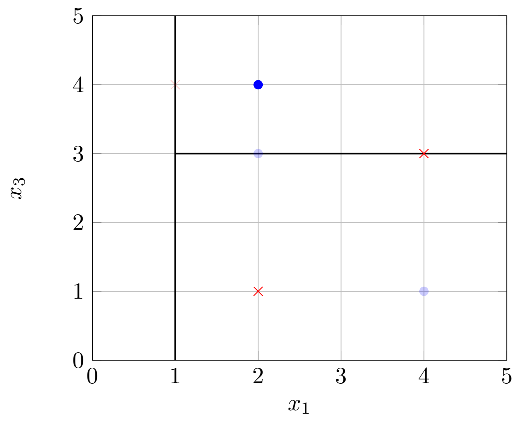
• All points need to be equidistant to each other	False
• The feature in one dimension needs to be the same for all points.	False
• Two points with the same features need to belong to the same class.	True
• The points in class 0 and the points in class 1 each form a plane, and the planes need to be distinct and parallel to each other.	True
• All points need to lie in a plane that is parallel to at least one coordinate axis	False
• Either class 0 or class 1 can only contain one point.	False

b) We first show the optimal splitting rules:



Here is a plot of these splits in two separate 2-D plots.





c) $x_4 = |x_1 - x_2 + 0.5|$, which allows all points to be classified correctly with the splitting rule $x_4 \leq 0.5$.

6 Neural Networks (16 points)

Binary classification

Consider a binary classifier that predicts whether there is a hot dog in an image. The last linear layer of the classifier has a softmax activation function σ with two output neurons: $\hat{f}_{softmax}(\boldsymbol{x})_i = \sigma\left(\boldsymbol{w}_0^T\boldsymbol{x}, \boldsymbol{w}_1^T\boldsymbol{x}\right)$, where $\boldsymbol{x}$ are input features from the previous layer.

a) `[6 points]` You want to reduce the model computational cost by converting the last layer into a linear layer with a sigmoid activation function ψ and only one neuron: $\hat{f}_{sigmoid}(\boldsymbol{x}) = \psi\left(\boldsymbol{w}^T\boldsymbol{x}\right)$. Derive the formula for the weights $\boldsymbol{w}$ given $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ so that $\hat{f}_{sigmoid}(\boldsymbol{x}) = \hat{f}_{softmax}(\boldsymbol{x})_1$ holds true for all $\boldsymbol{x}$.

Coin toss

You are offered to play a coin toss game. Before each coin toss, you can decide to finish the game or toss one more coin. If you throw a second head or tail, the game automatically stops, thus the maximum number of coin tosses is 3. At the end of the game, you need to pay as many points as coins you tossed, but if you managed to toss 2 heads (in total) you win 5 points back.

b) `[5 points]` Draw the Markov decision process (MDP), there is space on the next page.

c) `[5 points]` State the optimal policy π^* and calculate the expected reward achievable by the policy π^*. You can annotate your MDP of **b)** to help compute your solution.

Space for your MDP:

Solution

Binary classification

a) Sigmoid:

$$\begin{aligned}
\Pr(Y=0) &= \frac{e^{-\boldsymbol{w}\cdot\mathbf{x}}}{1+e^{-\boldsymbol{w}\cdot\mathbf{x}}} \\
\Pr(Y=1) &= \frac{1}{1+e^{-\boldsymbol{w}\cdot\mathbf{x}}}
\end{aligned}$$

Softmax:

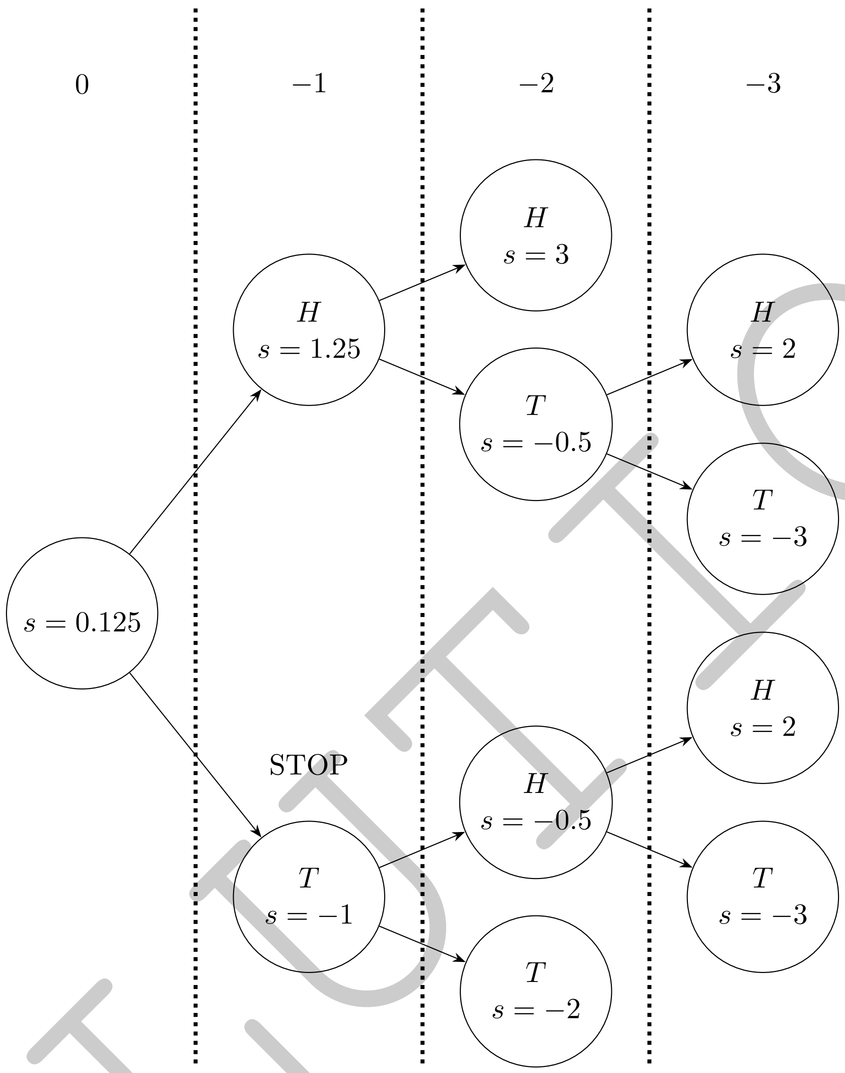
$$\Pr(Y=k) = \frac{e^{\boldsymbol{w}_k\cdot\mathbf{x}}}{\sum_{0\le c\le K} e^{\boldsymbol{w}_c\cdot\mathbf{x}}}$$

$$\begin{aligned}
\Pr(Y=0) &= \frac{e^{\boldsymbol{w}_0\cdot\mathbf{x}}}{e^{\boldsymbol{w}_0\cdot\mathbf{x}}+e^{\boldsymbol{w}_1\cdot\mathbf{x}}} = \frac{e^{-(\boldsymbol{w}_1-\boldsymbol{w}_0)\cdot\mathbf{x}}}{e^{-(\boldsymbol{w}_1-\boldsymbol{w}_0)\cdot\mathbf{x}}+1} = \frac{e^{-\boldsymbol{w}\cdot\mathbf{x}}}{1+e^{-\boldsymbol{w}\cdot\mathbf{x}}} \\
\Pr(Y=1) &= \frac{e^{\boldsymbol{w}_1\cdot\mathbf{x}}}{e^{\boldsymbol{w}_0\cdot\mathbf{x}}+e^{\boldsymbol{w}_1\cdot\mathbf{x}}} = \frac{1}{e^{-(\boldsymbol{w}_1-\boldsymbol{w}_0)\cdot\mathbf{x}}+1} = \frac{1}{1+e^{-\boldsymbol{w}\cdot\mathbf{x}}}
\end{aligned}$$

By setting $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_0$ the softmax can be reduced to the sigmoid activation function for binary classification.

Coin toss

b) MDP



c) Stop after the first throw if it is a tail, else do not stop. The expected win of a game is 0.125 with the optimal strategy.

7 Computability (14 points)

A cellular automaton generates a pattern row by row. A rule defines the state of a cell in the next generation (= next row) based on its current state and the current state of its two immediate neighbors. Note that each row is infinite in both directions.
In particular, the state of the cell at position (row i, column j) is specified by the states of the cells at positions $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$. For example, the following sample rule sets the cell's state to 1 in the next generation, when the current state of its left neighbor is 0, its current state is 0, and the current state of its right neighbor is 1.

$$(0, 0, 1) \rightarrow 1$$

You may omit ($\star$) parts of the rule if you want the rule to happen regardless of the state of the omitted cell. If multiple rules can be applied, the first rule that fits will be applied. For the following two ordered rules, the first rule sets the cell's state in the next generation to 1 if the cell's current state is 1; independent of the state of the neighbors ($\star$). The second rule will never be applied, because the first rule is always evaluated before.

$$(\star, 1, \star) \rightarrow 1$$

$$(0, 1, \star) \rightarrow 0$$

For the following two questions, the first row of the cellular automaton consists of $N > 0$ continuous cells in state 1 and all other cells are in state 0 (shown below). The following rows will be generated according to the rules you will write.

$$\cdots\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,\underbrace{1\,|\,1\,|\,1\,|\,1\,|\cdots|\,1\,|\,1\,|\,1}_{N}\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,\cdots$$

a) `[5 points]` Design an automaton with three states 0, 1, 2 and at most eight rules (the sample solution uses three states and four rules) that creates, at some point in a future generation, a stable (i.e., does not change anymore in future generations) continuous sequence of cells in state 2 of length $N + 1$ surrounded by cells in state 0.
Note: You will be awarded partial points for an automaton with too many states or rules.

b) `[9 points]` Design an automaton with at most four states 0, 1, 2, 3 and at most 14 rules (the sample solution uses three states and seven rules) that creates, at some point in a future generation, a stable continuous sequence of cells in state 2 of length $2N$ surrounded by cells in state 0.

Note: You will be awarded partial points for an automaton with too many states or rules.

Solution

Cellular Automata

a) The following automaton creates a stable sequence of cells in state 2 of length $N + 1$.

$$\begin{aligned}
(0,0,1) &\to 2\\
(\star,0,\star) &\to 0\\
(\star,1,\star) &\to 2\\
(\star,2,\star) &\to 2
\end{aligned}$$

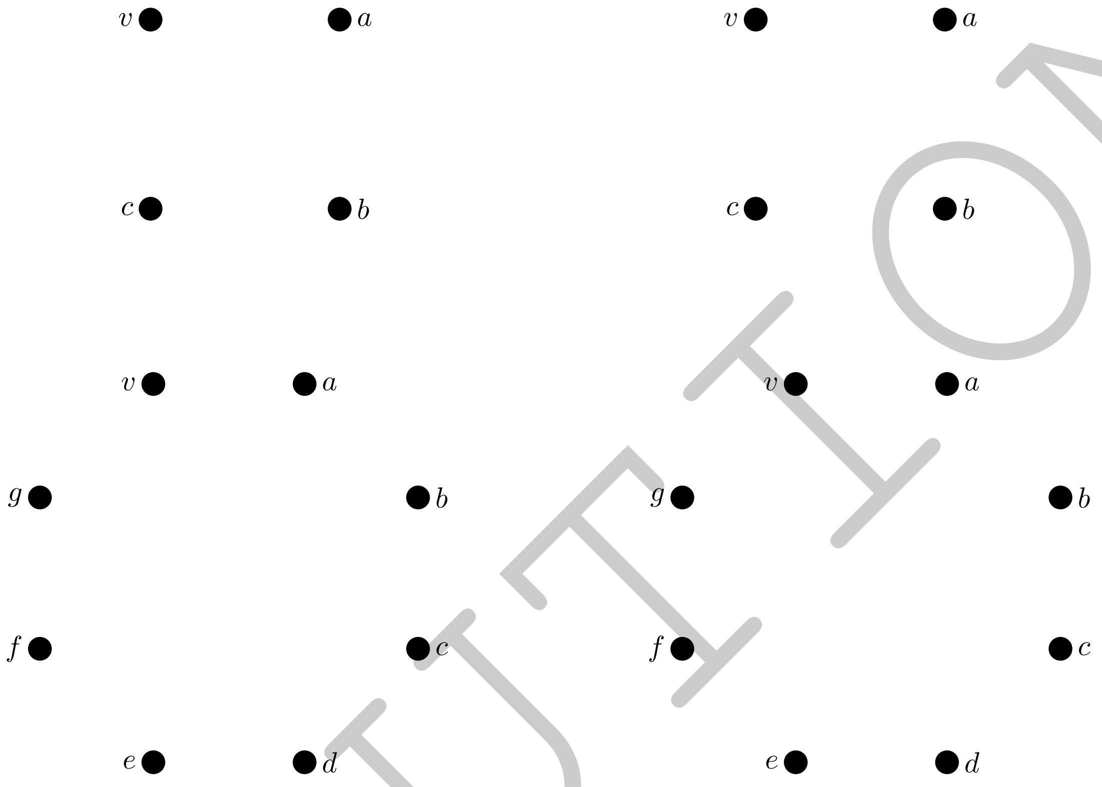
b) The following automaton creates a stable sequence of cells in state 2 of length $2N$.

$$\begin{aligned}
(0,1,\star) &\to 2\\
(0,0,1) &\to 2\\
(2,1,\star) &\to 2\\
(\star,2,1) &\to 1\\
(\star,0,\star) &\to 0\\
(\star,1,\star) &\to 1\\
(\star,2,\star) &\to 2
\end{aligned}$$

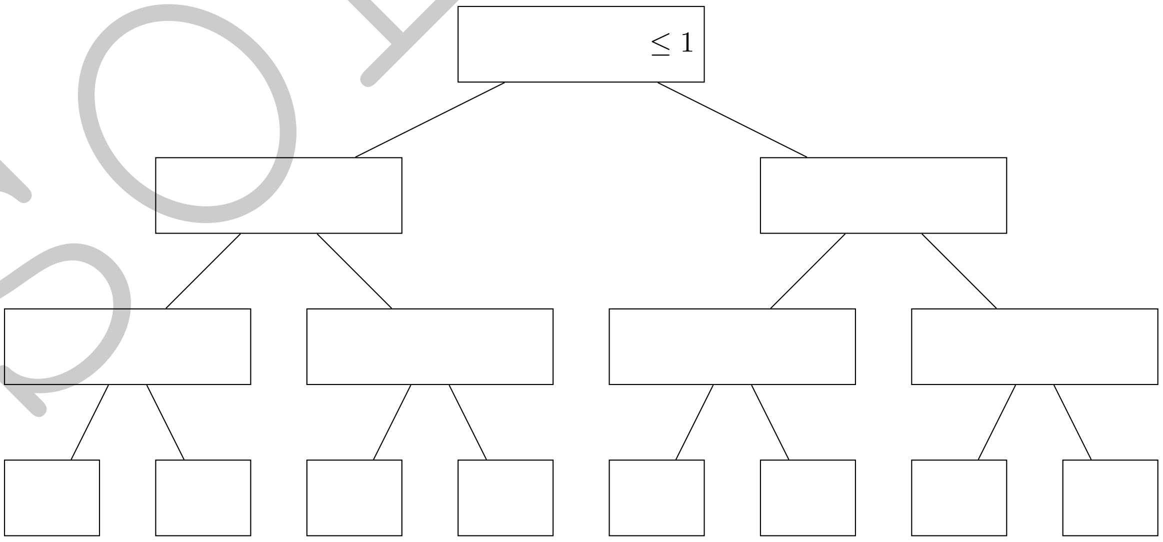
Replacements

Here you can find replacement templates for the fill-in tasks, in case your original solution becomes too messy. If you use these, please **clearly** indicate which one we should consider.

Question 2



Question 5



Use this page if you need extra space.