Calculators may be used in this examination provided they are not capable of being used to store alphabetical information other than hexadecimal numbers

UNIVERSITY OF BIRMINGHAM

School of Computer Science

MSc Introduction into Artificial Intelligence

Main Summer Examinations 2019

Time allowed: 1:30

[Answer all questions]

Note

Answer ALL questions. Each question will be marked out of 20. The paper will be marked out of 60, which will be rescaled to a mark out of 100.

Question 1 General Knowledge

(a) Consider that you would like to install a set of cameras to monitor human activity in an amusement park. You will start with a small number of cameras, but would like to acquire more cameras in the future. The company that is providing you with the cameras offers a service where they can check the cameras' working order once every 4 months for a fixed yearly fee. If a camera breaks between these checkup points and you would like to have it fixed before the next checkup point, you will have to pay an additional fee which you wish to avoid. In addition, your cameras need to have their clocks synchronised to facilitate tracking a person if necessary.

Explain **two reasons** why firefly synchronisation would be an adequate algorithm for synchronising the cameras **in this specific context**.

[4 marks]

(b) Explain the term "overfitting" in the context of machine learning and discuss how each of "regularization" and "model complexity" impact this phenomenon. Be sure to include the following:

(i) What do the terms "regularization" and "model complexity" mean and how does the increase and decrease of each impact model overfitting.

(ii) How is regularization related to the magnitude of weights.

(iii) How is model complexity related to the presence of non-linear terms.

(iv) Provide an illustration of what a non-linear decision boundary looks like.

[8 marks]

(c) Recall the three following elements, which are common to several learning algorithms: a) Hypothesis function, b) Cost function, and c) the (partial) derivative of the cost function. Clearly explain what each of them are, how they relate to each other and their purpose in learning algorithms.

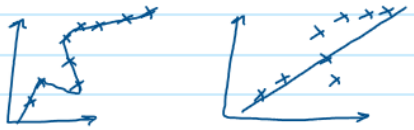
NOTE: You are **NOT** expected to provide the actual equations for any of these. You are only required to provide an overview of what they are.

[8 marks]

Phoebe

16) "overfitting" is when, in the training process when we train a model based off the training set, we are too specific with the model and this can lead to, in ML, a learned model $h(x)$ which which inaccurately predicts the output label of unseen examples.

e.g



overfit

not overfitting

- "regularisation" is like normalisation in that it generalises data that may have an influence in the final model, making more significant data points (greater size/density/outliers) less significant, s.t. the final model better characterises the overall dataset.

- "model complexity" is dependent on the dataset, e.g. a linear model of form $y = w_0 + w_1 x$ is less complex than quartic model such as $y = w_0 + w_1 x_1 + w_2 x_1^2 + w_3 x_1^3 + w_4 x_1^4$ (since this is more wiggly)
Basically, it is how much time and space requirements a model takes (we can measure this with benchmarking/ amortised complexity).

Sid

(b) Overfitting in ML refers to the phenomenon wherein a model cannot generalise the data distribution of a training dataset and instead, fits too closely to the training points.

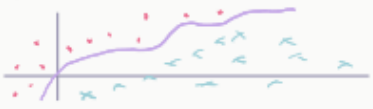
This results in the model learning to capture potential noise in the data, resulting in the model performing poorly on unseen data.

Regularisation combats overfitting by penalising large values of free parameters by introducing an new hyperparameter, which ultimately decreases model complexity and reduces the likelihood of overfitting.

Here, model complexity refers to how well a trained model can capture patterns, trends & nuances in the training data.

As regularisation is linked to free params, namely, weights of polynomial terms (including degree 0), regularisation primarily penalises non-linear polynomial terms of a high degree, which would otherwise increase complexity, leading to potential overfitting.

Non-linear decision boundaries — which are not straight lines — illustrate this high complexity, as shown:



(c) Recall the three following elements, which are common to several learning algorithms: a) Hypothesis function, b) Cost function, and c) the (partial) derivative of the cost function. Clearly explain what each of them are, how they relate to each other and their purpose in learning algorithms.

NOTE: You are **NOT** expected to provide the actual equations for any of these. You are only required to provide an overview of what they are.

[8 marks]

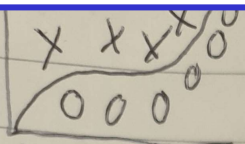
Shay

c) The hypothesis function is what we are using to make our model's predictions. It takes in an input (or several inputs, named "independent variables") and calculates an output. The goal of a learning algorithm is to make this hypothesis function resemble the data as closely as possible.

We formulate this "resemblance" with the use of a cost function. The cost function is a numerical measure of how "wrong" our model is, and we want to minimize this.

We can calculate the partial derivative of the cost function in regards to our independent variables, and we can use this to decide how we change the values of weights and biases to better model our data. The partial derivative tells us the direction of steepest ascent, so we can negate it to know what direction will take us closer to a local minimum.

Adwit



1c) • A hypothesis function is the equation of your trained model/predictor with weights adjusted.

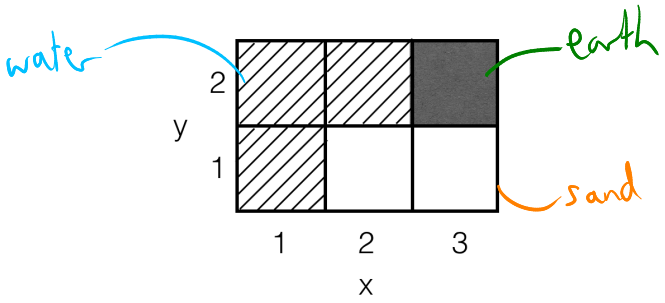
• A cost function is a measure of inaccuracy of a model (the further from labels the predictions are, the higher the cost). Its ~~par~~ free-parameters are the weights.

• The derivative of a cost function is a vector containing the partial derivatives of the function with respect to each free-parameter (weight).

- The cost function can be differentiated and we can then substitute in ~~a~~ set of actual weight values to get the gradient vector. By moving ~~by~~ ~~a~~ in the opposite direction iteratively, we optimally decrease the cost function until a minimum point is reached. Those become the weight values for the hypothesis function.

Question 2 Search and Optimisation

Assume that you are developing an algorithm to find the lowest cost path to move an army from a starting to a goal position in a strategy game. The field is organised as a grid, where the position whose coordinates (x, y) are (3,2) has an earth terrain; positions (2,1) and (3,1) have sand terrain; and positions (1,1), (1,2) and (2,2) have water terrain:



Each move can take the army one position up, down, left or right on the grid, so long as this does not move the army outside the grid. Independent of the type of terrain of the current position of the army, the cost of moving to an earth, sand and water position is 1, 3 and 10, respectively. For example, the cost of moving up from (2,1) is 10.

(a) Your army is in position (1,2) and you wish to reach goal position (3,1). An A* search algorithm is available to solve this problem, respecting the following rules:

- A state in the state space graph is identified by the (x, y) coordinates of the current position of the army, meaning that there are 6 possible states.
- The heuristic is the Manhattan distance between the state of the current node and the goal state, defined as follows:

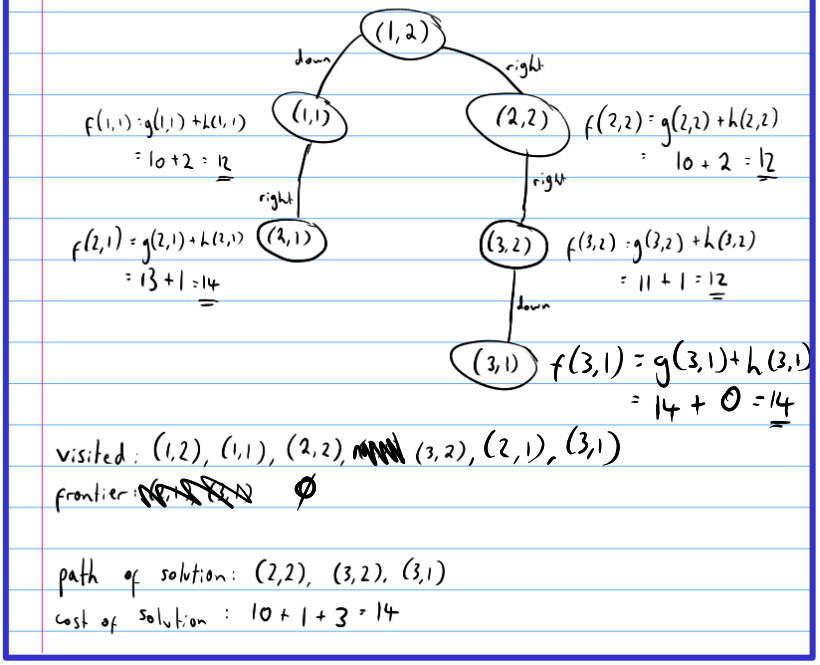
$$distance((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|,$$

  where (x_1, y_1) and (x_2, y_2) are the state of the current node and the goal state, respectively, and the operation $|k|$ represents the absolute value of k.
- Do not place children in the frontier if their corresponding state is already in the frontier or list of visited nodes with a smaller or equal $g(n)$, where $g(n)$ is the true cost from the origin to node n.
- Stop when you visit a node which contains the goal state.
- When deciding which node to visit, if there is a draw, choose to visit the node with the smallest x-coordinate first. If this still results in a draw, choose to visit the node with the smallest y-coordinate first. For example, if there is a draw between nodes whose states are (1,2) and (2,2), visit (1,2) first. If there is a draw between (1,2) and (1,1), visit (1,1) first.

Question 2 continued over the page

Shay

2) a)

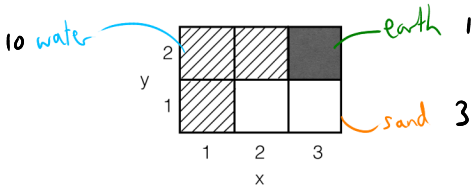


visited: (1,2), (1,1), (2,2), (3,2), (2,1), (3,1)

frontier: ~~(2,1), (3,1)~~ ∅

path of solution: (2,2), (3,2), (3,1)

cost of solution: 10 + 1 + 3 = 14



Write down the following information:

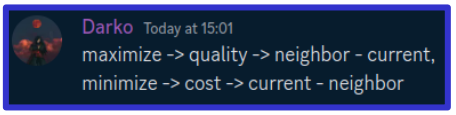
- Search tree produced by A*, indicating the following: $f(n)$, $g(n)$ and $h(n)$ values of each node n; which nodes are visited nodes; and which nodes are in the frontier when the algorithm terminates.
- Sequence of nodes *visited* by A*. Note: you can identify a node through its state.
- Sequence of states that compose the path retrieved as a solution by A*.
- The cost of the solution retrieved by A*.

[8 marks]

(b) Consider that you would like to solve this problem using Simulated Annealing instead of A*. The pseudocode of Simulated Annealing assuming maximisation is shown below. How would you change the pseudocode to minimise rather than maximise the objective function, so that you can apply it to this problem? Refer to the line number(s) that you would change, and how you would change it(them).

Simulated Annealing (assuming maximisation)

Input: initial temperature Ti, minimum temperature Tf



1. current_solution = generate initial solution randomly
2. T = Ti
3. Repeat until a minimum temperature Tf is reached or until the current solution "stops changing":
   - 3.1 generate neighbour solutions (differ from current solution by a single element)
   - 3.2 rand_neighbour = get random neighbour of current_solution
   - 3.3 If quality(rand_neighbour) <= quality(current_solution) {
     - 3.3.1 With probability $e^{(quality(rand\_neighbour) - quality(current\_solution))/T}$,
       - 3.3.1.1 current_solution = rand_neighbour

     }
   - 3.4 Else current_solution = rand_neighbour
   - 3.5 T = schedule(T)

cost(rand_neighbour) >= cost(current_solution)

$e^{(cost(current\_solution) - cost(rand\_neighbour))/T}$

[6 marks]

Question 2 continued over the page

(c) Consider that you would like to use Hill Climbing for a variation of this problem where your grid is much larger (with size $m \times m$, where $m > 100$) and you would like to move from $(1, m)$ to $(m, 1)$. Moves are forbidden if they take your army outside the grid or into any of the positions known to be well defended by the enemy. Your implementation of Hill Climbing is prepared to deal with minimisation problems, and your problem formulation has an objective function corresponding to the sum of the costs of the moves in your candidate solution, i.e.:

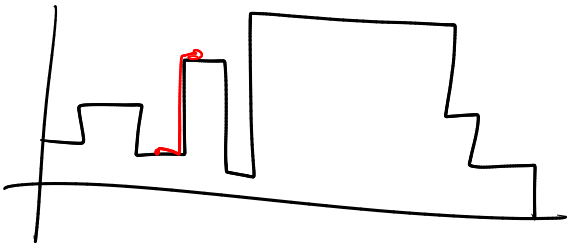
$$objective(solution) = \sum_{move \in solution} cost(move).$$

You decide to deal with the constraint related to forbidden moves by changing the objective function. For feasible solutions, the objective function is calculated as above. However, for infeasible solutions, you set the objective function to the value $C = 100$, i.e.:

$$objective(infeasible\_solution) = C.$$

Explain two problems of using this strategy to deal with the constraint.

[6 marks]



Shay

c) For larger-sized boards, even if the cost of all of the moves exceeds 100, this may still be a feasible solution. Using the death penalty method of constraint handling requires us to set the value to a constant that is significantly large that any feasible solution will always be preferred over it. 100 is not significantly large for C.

Since we only know $m > 100$ we can't safely set a constant C, as if $m = 1$ then any feasible solution in the search space will exceed C.

Phoebe

- since $m > 100$, it might be that m is MUCH larger than $C = 100$, meaning the obj func for infeasible solutions doesn't have any much impact, so we may still pick an infeasible solution.
- also this constraint doesn't prevent us from moving outside the grid of $m \times m$ so we
- even if this is a good value for C, it may increase the amount of time taken to find the solution (path to goal) since we have to compare all possible infeasible solutions, when choosing the next move.
- hill climbing is greedy, if all neighbours are infeasible, we will stop at a local not global maximum.

Question 3 Machine Learning

(c) Consider the scenario wherein we wish to train a model to drive a virtual car in a virtual world. Describe how you would set up a reinforced learning algorithm to learn this objective given that the car has sensors providing its distance from various objects around it and has control over the the angle of the steering, the accelerator and brakes. Can this problem remain a reinforced learning problem if we were interested in training a model to control a car in the real world? Why? How would you modify it? **[8 marks]**

Shay