The report must be delivered in PDF format and must include the following:

• Practical and Answering questions, written in the form of small 10-15 pages and a

description of the process developed

• **Description of the features that you have included in your system: DESCRIBE AS MUCH AS POSSIBLE THE WORK DONE**

• **Conclusion: summary of what they learned**, identifying initial shortcomings that have

been (or not) covered by this study, acquired new knowledge, new skills (ie, **things**

**that have been made during the activity that had never been made indicating its**

**level of acquisition (from 0 to 3, to use all the same scale)**, already had skills but have

been worked and reinforced this activity, **assessment of individual work (hours, job**

**performance, planning and implementation of task) and collaboration with other**

**peers or the teacher**. References, modeled standard citation, with each critic

reviews.

FIRST PART (Practical Part)

**Reinforcement Learning:**

* What is RL?
* Model-Free ML
* Explorations VS Exploitation dilemma

**Algorithms:**

* Monte Carlo
* TD (n)
* Q-Learning [1] [2] [3] [4] [5] [6] [7] [8] [9]

Q-Learning is an algorithm in Reinforcement Learning for the purpose of strategy learning.

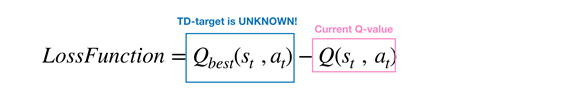
Mathematically, the strategy in Q-Learning is modeled as an action-value function Q(s, a), which represents how much is the Agent willing to take an action ‘a’ in given state ‘s’.

In addition, the value of Q(s, a) returned is called Q-value. The larger Q-value, the larger possibility of the action a will be taken since the Agent expects to get more rewards if it has large Q-value.

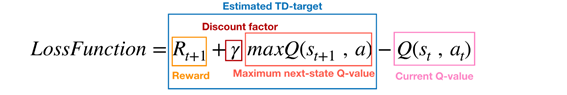
We use Q(s, a) as the policy of the Agent, so if the best Q(s, a) is found, then the Agent has learned the policy perfectly. Therefore, our goal here is to find the best Q(s, a).

The Cost Function (or Loss Function) is used for measuring the quality in Supervised Learning and it can be calculated precisely since the label (i.e. right answer) is known.

It is also used in Reinforcement Learning, but the right answer is always unknown. Then, how do we estimate the quality of the policy?



The way we apply to estimate the best Q(s, a) is Bellman Equation



We want to minimize Loss Function using, for example, Gradient Descent:



Learning Rate is an hyper-parameter for controlling the convergent speed of updating procedure and Discount Factor is another hyper-parameter for weighting the importance of estimated future reward. The value of discount factor is between 0 and 1. The closer to 1, the more important the future reward is.

We will end up with a matrix to store what we know after some random values.

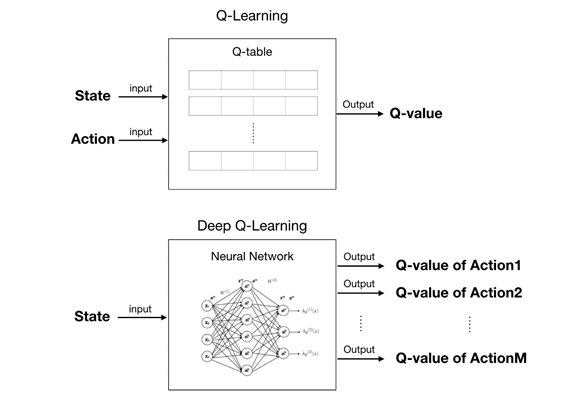
The problem with this is that the AI will never explore new actions because it knows the rewards of the already known ones.

The way to solve this problem is with random exploration. The IA will take eventually a random action so in the long term it will know more possibilities.

The explorations vs exploitation dilemma is solved then with an Epsilon variable that goes from 1 to 0.

Now we have another problem: with the matrix method, the space we need is n states by m actions. When these numbers are large such as 1000 states and 1000 actions per state, we would need a matrix of 1 million elements. Thus, the matrix method doesn’t work in most interesting problems such as chess, which needs 10¹²⁰ states space. Instead of recording Q-value precisely with matrix, the approximately way is more feasible such as Neural Network (NN) framework

That is why we go deep:



We only must initialize hyper-parameters and the NN Model as we saw in this subject and, then, make it start training.

This might seem enough, but the basic Q-Learning algorithm has some disadvantages like overestimations of action value: Q-value, in basic (Deep) Q-Learning and introduce one popular solution, called Double Deep Q Network (Double DQN or DDQN), a Double Q-Learning with Neural Network architecture.

* Double Q-Learning

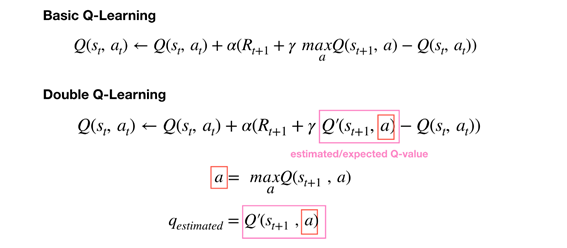
Briefly, the problem of overestimations is that the agent always chooses the non-optimal action in any given state only because it has the maximum Q-value.

The assumption behind the idea is that the best action has the maximum expected/estimated Q-value.

However, the Agent knows nothing about the environment in the beginning, it needs to estimate Q(s, a) at first and update them at each iteration. Such Q-values have lots of noises and we are never sure whether the action with maximum expected/estimated Q-value is really the best one. Therefore, the learning process will be very complicated and messy.

Double Q-Learning uses two different action-value functions, Q and Q’, as estimators. Even if Q and Q’ are noisy, these noises can be viewed as uniform distribution.

The update procedure is slightly different from the basic version:



Q function is for selecting the best action ‘a’ with maximum Q-value of next state.

Q’ function is for calculating expected Q-value by using the action ‘a’ selected above.

Update Q function by using the expected Q-value of Q’ function and we have it.

[1]<https://www.youtube.com/watch?v=qhRNvCVVJaA&start=438s>

[2]<https://www.youtube.com/watch?v=wrBUkpiRvCA&t=517s>

[3]<https://www.youtube.com/watch?v=mo96Nqlo1L8&start=484s>

[4]<https://www.youtube.com/watch?v=0bt0SjbS3xc>

[5]<https://www.youtube.com/watch?v=ILDLT97FsNM>

[6]<https://www.youtube.com/watch?v=CXwvOMJujZk>

[7]<https://medium.com/@qempsil0914/zero-to-one-deep-q-learning-part1-basic-introduction-and-implementation-bb7602b55a2c>

[8]<https://medium.com/@qempsil0914/deep-q-learning-part2-double-deep-q-network-double-dqn-b8fc9212bbb2>

[9]<https://towardsdatascience.com/deep-reinforcement-learning-tutorial-with-open-ai-gym-c0de4471f368>

SECOND PART: Search and Planning

Describe the TABU, GENETIC and A\* Search algorithm

1. In which category do you classify it. Indicates advantages and disadvantages with the algorithms that have been studied in the classes.

2. Which advantages and disadvantages do you discover in this algorithm to solve the traveling salesman problem.

TABU [1]:

It is a type of metaheuristic search that uses local or neighbourhood search methods. These iterations change the current solution to an improved one, which is on the neighbourhood of the current solution.

This may create some problems, like getting stuck in a zone of poor scoring, but this can be avoided tabu explores other possibilities, other solutions. To solve this problem tabu has a memory structure that determines all the possible neighbours that could be accepted so now he can improve the current solution.

This memory structures are called tabu list, that is form by a couple of rules used to choose the solutions that will be admitted as part of the neighbourhood. It is common to see solutions that change after you iterate in this algorithm. In the end there is a stop criterion that will determine when the search is done.

Related to the salesman problem, one of the advantages that tabu searches offers is that it can avoid getting stuck when a set of cities are commonly repeated. On the other hand, if you don’t understand the solution it will be difficult to set your list and your solution won’t be optimal.

[1] <https://en.wikipedia.org/wiki/Tabu_search>

Genetic [2]:

It is also a type of metaheuristic search; it is all based in the natural selection an how is evolution conceived now a days. The algorithm hide under it relays on bio-inspired operations like mutations, crossover and natural selection.

It basically works as follows, there is a population of candidate solutions that will evolve to a better solution than the current, and in order to do that they have a set of properties that they can mutate and change.

At the beginning everyone from the population has been randomly created and once the iterations start, also called generations, the result gets better and better, they have an individual value that try to maximize, commonly called fitness. Those that got the best result will be used to get their genomes and create a new generation that will iterate and iterate.

Related to the salesman problem, it is usually an expensive algorithm. Also, the bigger the problem is the more expensive the algorithm will be due to the high number of mutations that can be created each generation. This algorithm has no stop criterion. It can get stuck in local optimal solutions.

[2] <https://en.wikipedia.org/wiki/Genetic_algorithm>

A\* [3]:

Known as A-star is a graph traversal and path search algorithm. Used in computer science because it is complete and optimal. It makes use of weighted graphs. Basically, what it tries is to get to a certain node by having the smallest cost possible.

In order to accomplish this task, it is based in tree paths. Each node has a weight and each node also may have sons. On each iteration, it determines the path that is going to follow, and this path will be the result of adding the next node and the total cost that has already gone through.

The typical implementation is the following one: *“At each step of the algorithm, the node with the lowest f(x) value is removed from the queue, the f and g values of its neighbours are updated accordingly, and these neighbours are added to the queue. The algorithm continues until a goal node has a lower f value than any node in the queue (or until the queue is empty).”* Where f for the goal is the shortest path possible to be created.

Once you know the path you can storage the information of all the nodes of that path by keeping track of the predecessor node in the next one.

Related to the salesman problem, this algorithm seems like the one made for it, it seems like it was design as a pathfinding algorithm. If you set the distances as weight the only thing you can do is wait for the result. Depending on the complexity of the problem this algorithm can handle better than the TABU algorithm. And this one will never get stuck like the genetic one or the TABU if you don’t know how to set the rules for the list.

[3] [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

What is overfitting training problem and how to solve it in neural networks and specially in Keras.

Overfitting [4][5][6]:

When we talk about overfitting, we are talking about a common mistake done while training an AI. The main problem occurs when the result of that training is too exact and corresponds closely to a unique set of data. This will certainly cause a fail when trying to fit a new set of data with the knowledge learned with the previous data. Any way as you may have this problem you can also have the underfitting problem, which is less common but still possible if you miss the parameters that should be in a correct model.

[4] <https://elitedatascience.com/overfitting-in-machine-learning>

[5] <https://en.wikipedia.org/wiki/Overfitting#Remedy>

In order to solve this problem in keras there are different approaches, but the best ones are first of all try to train as much as possible your AI, the more it learns by training the more knowledge it will have and overfitting or underfitting won't appear. It this first approach isn’t possible or if you want to be more accurate you can implement constraints on the quality and type of information you obtain from your model.

There are different ways to implement constraints, one of them is weight regulation, that considers the fact that the explanation most likely to be correct is the “simplest” one.

It works as follows: “*A “simple model” in this context is a model where the distribution of parameter values has less entropy (or a model with fewer parameters altogether, as we saw in the section above). Thus a common way to mitigate overfitting is to put constraints on the complexity of a network by forcing its weights to only take on small values, which makes the distribution of weight values more “regular”.* “ [6]

[6] <https://keras.rstudio.com/articles/tutorial_overfit_underfit.html>

Give an original example of reasoning based on backward chaining.

Backward chaining is used in games like chest, where you have the goal you want to achieve and you can go backwards to the current state of the game. In chest this method is called [retrograde analysis](https://en.wikipedia.org/wiki/Retrograde_analysis) and it creates table bases.

Also, backward chaining is used in real life, for example when babies learn how to eat, they know that the goal is to have the food in the mouth and from there they go back one step at the time.

We want to solve the problem of travelling Sales problem (https://en.wikipedia.org/wiki/Travelling\_salesman\_problem), how would you solve it? What algorithm would you use and why? How should the problem be coded to solve it with the "tabu search" algorithm?

Based on what we have learned in this second part the optimal solution probably will be the A\* algorithm witch is critical for path finding, and is the perfect approach for this problem since each travel has his own weight so the A\* tree can be created really easy.

The other possible answer is to use tabu, as this question suggests but it will be a bad choice because the bigger the problem is the better A\* is compared to tabu. Also, tabu may be stuck in suboptimal solutions.

In order to be implemented tabu should have a couple restrictions related to looping between cities, also it will have a restriction related to go to the lowest weight possible. Having some sort of exploration restriction to try to find new solutions or sub solutions that may be more optimal in long term.

QUESTIONS DONE ON THE ORAL EXPOSITION:

**What does ReLu mean:**

In neural networks it is used to define those units that use the rectifier. A rectifier is a function define as f(x)=x+=max(0, x), where x is the input that the neuron receives.[8]

[8] <https://en.wikipedia.org/wiki/Rectifier_(neural_networks)>

**Why do we use these numbers of layers:**

As everybody should know the less layers you use on a neural network the better performance you will get, but your result won’t be satisfactory enough. To know the number of layers we should use we have to consider that just one layer won’t be enough to solve this problem with a proper solution. [9]

We have 5 layers for each neural network in the double deep Q algorithm, 3 convolutional in order to process the game image, then the result is flattened to be able to be fed to the next 2 regular perceptron layers that analyze the data.

For the simple deep Q algorithm only one neural network is used but with 7 layers

[9] <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

**Sequential model:**

The sequential API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs[10], but it is much more simpler than functional while performing correctly the purpose for this work at learning how to play PacMan. As we don’t want a complex model, we went for the sequential one. as the Sequential model API is enough for developing deep learning models in our situation[11].

[10] <https://jovianlin.io/keras-models-sequential-vs-functional/>

[11] <https://keras.io/getting-started/sequential-model-guide/>