The Exploration-Exploitation Dilemma when using the A3C-Algorithm

Erik von Brömssen
Robin Nilsson

April 14, 2021

1 Motivation

Perhaps one of the most well-known and important concepts in the fields of decision making and machine learning is the trade-off between exploration and exploitation. Exploration is necessary in order to sufficiently explore the available state space; without this, a good policy, let alone an optimal one, may never be found. Exploitation is however also required for an agent during its training in order to reinforce good choices and make learning possible. One very common strategy to include both of these during training is to use what is known as an ε-greedy policy; at each time step, choose a random action with probability ε (exploration), otherwise choose the greedy action with respect to the current (usually estimated) action-value function or policy (exploitation).

In some implementations, an ε value that starts relatively large and then monotonically decreases in time is used, in an attempt to make agents explore more in the early stages of its training, then tending to exploit more than explore in the later stages. This approach is not unreasonable as you could expect an agent to find a somewhat feasible policy after some initial exploration, and then letting it exploit more (although still with a nonzero ε) can result in it refining its policy further.

If ε is chosen too large, too little exploitation happens and our agent may never have time to reinforce a good policy; if it is chosen too small, our agent may never explore more than a small part of the state space, which may well be a subset containing only low-value states. One has to decide on a balance of this trade-off when designing a single agent. It is possible, however, that a constant or monotonic ε is suboptimal in many training situations, regardless of how the exploration-exploitation is balanced. More sophisticated mixes of a greedy policy and random exploration may be advantageous in some situations.

There have also been attempts to construct adaptive ε algorithms such that ε may increase (more exploration) if, for instance, the results lately have been poor. Such attempts can for instance be found in [5, 6]

The usual exploration-exploitation algorithms are only time-dependent, i.e. as the training session goes on the value of ε is changed. There are other versions as well, for example in [2] a state-time dependent ε is used. As stated in the paper,
"The desired behavior is to have the agent more explorative in situations when the knowledge about the environment is uncertain"
but also
"... with methods that are impractical in large state spaces because of their memory and computation time requirements."
Saving information for each state in other words might prove to be difficult for large or continuous state spaces.

Lastly, there are other ways to counter the exploration-exploitation trade-off problem other than ε-methods; one such solution be found in [3]. For an interesting read about the topic one can check out [4].

2 Problem formulation and restrictions

There are two main problems that we address in the report. Firstly we explore whether we are able to construct a simple adaptive ε greedy method with the goal of improving performance. Secondly we explore the possibilities for a grid-based ε method which aims at solving or reinforcing the idea of having a state-dependent ε for state spaces that are very large and/or continuous.

While we recognise the attempts of other methods for solving the exploration-exploitation dilemma we will focus on improving the ε-decay algorithm. That is gaining higher rewards, better stability and lower risk at getting "stuck" in low rewards than the usual ε-decay method. With "getting stuck in low rewards" we do not necessarily mean converging to bad local minima, but rather doing so much exploration that a good policy is never acted upon, or doing too much exploitation, hindering the agent from exploring and finding a good policy to begin with. This might happen if for example some parameters (e.g. the learning rate or ε decay factor) are not properly tuned, and as such it is encouraged to find an algorithm which escapes such scenarios more frequently.

As the name suggests, the grid-ε method will be an extension of the normal case, where ε is a scalar, to one where it is defined on some grid; specifically a grid in the state space, thus making ε a function of state. The problem then becomes to fine-tune the size and number of nodes in the grid, since having too many nodes will be computationally heavy and runs the risk of having many nodes not being explored enough or at all. Neither do we want a grid where the nodes are too sparsely placed, since too many states will be accompanied to one node.

3 Methodology

The algorithms are tested in an asynchronous advantage actor-critic (A3C) setting in which there are multiple local agents, or *workers*, which separately explore the state space, each having their own ε-algorithm attached to it. In the original article [1] the authors mentioned that having different exploration-exploitation methods for the agents adds to the robustness. We did not find a clear advantage of using different configurations of exploration-exploitation for the different workers, other than what is mentioned in section 3.2.

We test our grid-ε algorithm against a couple of different implementations of ε-methods; the ordinary ε-greedy decay, the adaptive ε-method found in [5] and lastly the *follow-then-forage*-method introduced in [6] for when the learning rate is decreased a little. This is to display more clearly some of their shortcomings and how the grid-ε copes with them. The *follow-then-forage* is an algorithm specially designed for the A3C algorithm.

3.1 Standard ε-greediness with decay

We implemented a standard A3C agent using ε-greedy exploration with exponential decay, in order to compare its performance to our novel algorithms. The results from training this agent was used as a baseline; in particular, we consider the moving average score during training to be a measure of performance.

3.2 Adaptive ε using rectangular pulses

Running the standard A3C agent (from section 3.2) but periodically setting $\varepsilon = 1$ for the duration of a few episodes resulted in faster training in some cases, but poor training in others. If it is possible to induce such pulses in ε only at beneficial times, this could be a viable adaptive algorithm. Given a measure of performance that such an algorithm would respond to, it can be meticulously designed for optimal performance; the width and height of these ε-pulses can be adaptively changed during training, in response to the performance measure among other things. Measuring performance can be done every episode or time step, eagerly finding opportunities to send a pulse; or we can choose to only check if a pulse is needed at some specific frequency, giving the agent more time to follow the regular ε decay behaviour in between pulses.

In this report we consider a primitive version of such an algorithm, not utilising much of the presented flex-

ibility in order to, hopefully, get some lower bound on how an algorithm of this type can perform. Specifically, we implemented an algorithm which uses a normal ε decay, but periodically (with fixed frequency) checks if the average score of the last N episodes is worse than it was at the last check; if so, it pauses the ε decay and sets $\varepsilon = \varepsilon_{\max}$ for a fixed number of episodes, before continuing from where the decay left off. Furthermore, using the multiple workers in a single A3C algorithm we offset these periodic checks across each agent to lower the risk of a period with low scores going unnoticed.

Algorithm 1 (found in the appendix) shows the layout of our adaptive algorithm. Here, ε is a usual value in $(0, 1)$ which decays exponentially. If the agent is exploiting, it takes random actions with probability `current_epsilon` $= \varepsilon$; it if it exploring, it takes random actions with the higher probability `current_epsilon` $= \min(1, \max(\varepsilon_{\max}, \varepsilon + 0.1))$ where $\varepsilon_{\max}$ is a constant. The variable c_e is the "exploration counter", which is nonzero only when the agent is exploring and decrements every episode. The constant C_e is the length of such an exploration period, in episodes. The variable c_{ec}, the "exploration check counter", increments every non-explorative episode. Once this counter is a multiple of f_{ec} (the "exploration check frequency"), we calculate the mean score of the last N episodes, μ, and compare this to the mean from the last check μ_{prev}; if the mean score has not increased by at least G since the previous check, we set $c_e = C_e$ and thus start exploring.

3.3 ε-greedy exploration on a grid on a continuous state space

A simple idea is to use some state dependent $\varepsilon = \varepsilon(s)$ in the hopes that this may take into account which parts of the state space have been explored and which have not, hopefully for better performance. In the case of a discrete and finite state space, a value table might be enough, however this is of course impossible if the state space is either infinite or continuous. Thus we propose to define a mesh on the state space, such that we only keep a finite number of nodes in this mesh at which we can change the value of ε during training; $\varepsilon(s)$ can then be defined as some interpolation of nearby values on grid nodes. Each node in the mesh is then "responsible" for an area of states and has an epsilon attached to it, which is updated as usual or in some novel way. We will specifically use n-linear interpolation of grid values for evaluation.

The mesh is fine-tuned for its environment and by experience as mentioned earlier. For instance if the state space includes infinity most likely it is enough to limit the range of the mesh to a finite subset of the state space. This is of course to both have enough nodes and have them close enough to have the desired effect. If the mesh is too sparse there may be too many states which belong to one node and the generalisation will be too grand, while on the other hand if there are too many nodes limits in memory and computational power will come into play.

In our case we simply determined, by experiment, a small subset of the state space which contained all states ever visited in practice, and defined our mesh on this subset. Any state s outside of this subset was projected onto the grid before evaluating $\varepsilon(s)$.

4 Results

All tests are performed on the OpenAI Gym environment *Cartpole-v1*. The A3C agents we have trained are using four worker agents each.

For the simulations, if not otherwise specified, we have chosen to set the discount factor $\gamma = 0.99$, the learning rate $\alpha = 0.0025$ and, if used, the decay factor when updating the ε to 0.99.

4.1 Adaptive ε using rectangular pulses

We trained an agent using the adaptive ε algorithm five times for 700 episodes each time, and for comparison we did the same with an agent using a standard, non-adaptive ε decay. In these episodes, the number of steps per episode was capped to 500. The moving average (of length 50) of the training scores for all five training runs can be seen in figure 1, together with the mean of the five moving averages.

Configuring the adaptive algorithm to check if exploration is needed every 25 episodes, by computing the average of the last $N = 5$ episodes, and if so, exploring for 10 episodes at a time with an $\varepsilon_max = 0.5$ and

target score increase of $G = 20$, we got the results shown in figure 2.

A comparison of the mean of moving averages between the adaptive and non-adaptive algorithms is shown in figure 3.

4.2 grid-ε on a continuous state space

We used a homogeneous grid with on the state subset $[-4.8, 4.8] \times [-0.5, 3.0] \times [-0.42, 0.42] \times [-0.5, 1.0]$, where 5, 10, 20 and 50 nodes in each dimension respectively. As mentioned, this subset is in practice the only visited subset of the CartPole environment, with a few rare exceptions.

When testing the algorithm the learning rate was decreased to 0.001 such that the results would fluctuate more and be easily comparable. Results are shown in figure 5. Both the standard ε-greedy decay algorithm and the adaptive ε-greedy in [5] as well as follow hen forage from [6], as shown in fig 5a), tend to stay in the lower regions. While the adaptive ε-greedy in [5] is more stable, the standard ε-greedy decay "explodes" one or two times out of ten, and reaches scores higher than the rest of the runs. As said this rarely happens, and usually the average scores stays in the lower regions. Figure 5b) shows results from the grid-ε algorithm when using 5 respectively 10 nodes in each dimension of the grid. While the algorithms fluctuate more, especially in the beginning, there is a steady increase and as they successfully "escape" the lower regions. The fluctuations also seem to become smaller as the algorithms progress. While the grid-ε didn't reach as high as the standard ε-greedy decay at best did, it performed better in general.

Figure 5c) shows results from the grid-ε algorithm when using 20 respectively 50 nodes in each dimension. Here the results fluctuate much more than the previous case and the average results are also lower. The lowest training run with 20 nodes is an outlier, similar but opposite to standard ε-greedy decay high results and shows that the grid-ε algorithm also may get stuck in the lower regions.

Fig 6 represents a change in the decay rate for the grid-ε by 0.01 (0.99 to 0.98) when using 5 respectively 10 nodes. This change resulted in a somewhat more stable increase when n = 5 and in general it reached higher scores then before.

5 Discussion

5.1 Adaptive ε using rectangular pulses

Observing the mean of all five training runs in figure 3 does not indicate that the adaptive version performs better than the standard ε decay. Arguably, it does not perform strictly worse either, as the two curves are relatively similar, both reaching the same average score after 700 episodes; further testing could however reveal a significant performance loss, as five training runs is a small amount of samples. Of more interest might be the fact that the adaptive algorithm seems to have an almost monotonic moving average, while the standard algorithm observes a local maximum followed by a significant loss of score at 500 to 600 episodes. Comparing further the individual training runs in figures 1 and 2 we see an indication of different behaviour between the two algorithms; once it reached above the 300 average score mark, the non-adaptive variant sometimes fell down in score for a long period, and some of the training runs that continued to increase and approach the maximum score lost this progress and fell down significantly. In contrast, the adaptive variant seems to lack this behaviour; the few training runs that did fall down in score recovered quicker and they all seem to share the same, almost monotonic, lower bound.

Thus we conclude that the adaptive algorithm appears to take slightly longer than the non-adaptive to reach a certain score average, but it seems to be more stable; the spread of average scores towards the end of training seems to be smaller for the adaptive variant than for the non-adaptive.

The algorithm could be improved in a number of ways; apart from more thorough tuning, using a constant target score increase G is not reasonable if we expect the moving average score to converge; in this case, G should approach zero, or another system should be used entirely. Further, the height of pulses could be determined adaptively, such as being a function of how much the average score has decreased over the last two checks.

5.2 grid-ε on a continuous state space

The results of the grid-ε algorithm seem promising. The chances of "escaping" the lower regions which the standard ε-greedy decay had problems with seem to have increased (although not in its entirety) which induces more reliability in the algorithm since it is more likely to produce better results. While in general having a higher score than the standard ε-greedy decay, the algorithm is not "exploding" and reaching heights which the standard ε-greedy decay algorithm at certain times may do. This is because grid-ε in general does more exploring (having a standard ε-greedy decay behaviour at each node) and is constructed to be more sure of the environment before exploiting. Even if one episode finds a path which results in a high score, the algorithm will still be set to explore more frequently than the standard ε-greedy decay.

For this reason a faster decay rate for the grid-ε, especially for meshes with a high number of nodes, may be desired. Figure 6 displays the results of a small change in the decay rate. We got higher scores and for $n = 5$ a more stable increase. For $n = 10$ we still have highly fluctuating results until about the last 100 episodes where the algorithm imposed next to always exploiting.

For the standard ε-greedy decay there is also a fine line for when the decay rate is too fast and too slow, and in the grid-ε case this also depends on the number of nodes in the mesh. Thus an adaptive algorithm which takes into account the number of nodes and the results of an episode may be desirable. One could also argue that an adaptive mesh, such that regions where the algorithm frequently visits has more nodes than others, is also desired. This supposed algorithm could be interpreted as the algorithm trying to understand the regions of the environment where the exploiting leads it, i.e. the path which (hopefully) leads to the optimal/highest score should have more nodes (better understanding of that region) surrounding it. But as mentioned below it might also have drawbacks, where local nodes rarely get visited and when visited, it is more likely to explore when it instead should exploit.

The fluctuations which are presented in figures 5b-c) of the grid-ε algorithm can be partially explained by the problem which [3] also addressed, that if the number of states is too high there is an increased risk that some nodes very rarely get visited. The choice of mesh clearly affects the results which is seen by comparing the figures. In figure 5c) we notice that having too many nodes in the mesh results in higher fluctuations (more exploring) even after many episodes have passed when the algorithm should (in an optimal setting) be exploiting more. In 5b) the results are better and the algorithm exploits more for higher episode counts.

References

[1] Google DeepMind. *Asynchronous Methods for Deep Reinforcement Learning.* https://arxiv.org/pdf/1602.01783.pdf

[2] Michel Tokic *Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences* http://www.tokic.com/www/tokicm/publikationen/papers/AdaptiveEpsilonGreedyExploration.pdf

[3] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman Filip De Turck, Pieter Abbeel *#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning* https://arxiv.org/pdf/1611.04717.pdf

[4] *Exploration and Exploitation Strategies, Profit Performance, and the Mediating Role of Strategic Learning: Escaping the Exploitation Trap* https://www.researchgate.net/publication/256717980_Exploration_and_Exploitation_Strategies_Profit_Performance_and_the_Mediating_Role_of_Strategic_Learning_Escaping_the_Exploitation_Trap

[5] Alexandredos Santos Mignona, Ricardo Luisde Azevedo da Rochaa *An Adaptive Implementation of ε-Greedy in Reinforcement Learning* https://www.sciencedirect.com/science/article/pii/S1877050917311134

[6] James B. Holliday *Improving Asynchronous Advantage Actor Critic with a More Intelligent Exploration Strategy* https://www.mobt3ath.com/uplode/book/book-66871.pdf

A Algorithms

Algorithm 1: The training loop of the adaptive ε using periodic checks and rectangular pulses, for a single local A3C agent.

for *each episode* **do**
  if $c_e > 0$ **then**
    `current_epsilon` $\leftarrow \min(1, \max(\varepsilon_{\max}, \varepsilon + 0.1))$
    $c_e \leftarrow c_e - 1$
  else
    `current_epsilon` $\leftarrow \varepsilon$
    $\varepsilon \leftarrow \max(\varepsilon \cdot d_\varepsilon, \varepsilon_{\min})$
    $c_{ec} \leftarrow c_{ec} + 1$
  end
  if $c_{ec} \bmod f_{ec} = 0$ **then**
    $\mu \leftarrow$ mean score of last N episodes
    if $\mu - \mu_{prev} - G \leq 0$ **then**
      $c_e \leftarrow C_e$
    end
    $\mu_{\text{prev}} \leftarrow \mu$
  end
  while *not done* **do**
    Take random action with probability `current_epsilon`
    Train agent according to A3C
  end
end

B Figures

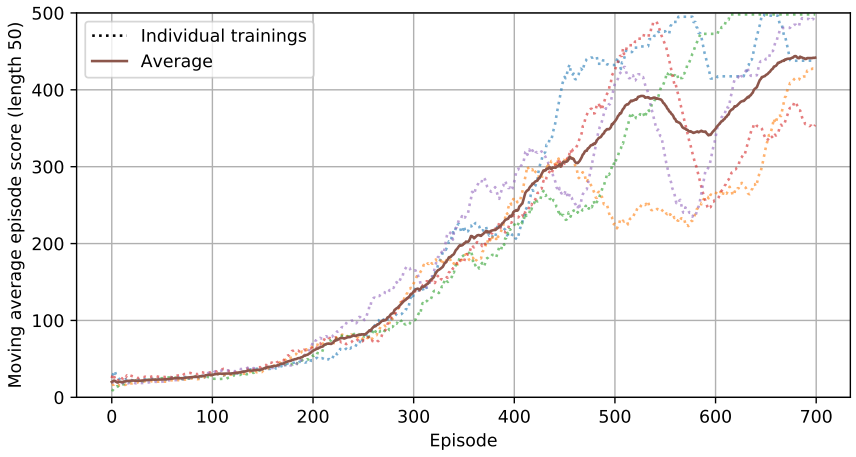


Figure 1: Moving average episode scores from five training runs of the standard ε decay version (see section 3.1). Solid line is the mean of all five; this appears in figure 3. Episodes ended after max 500 steps.

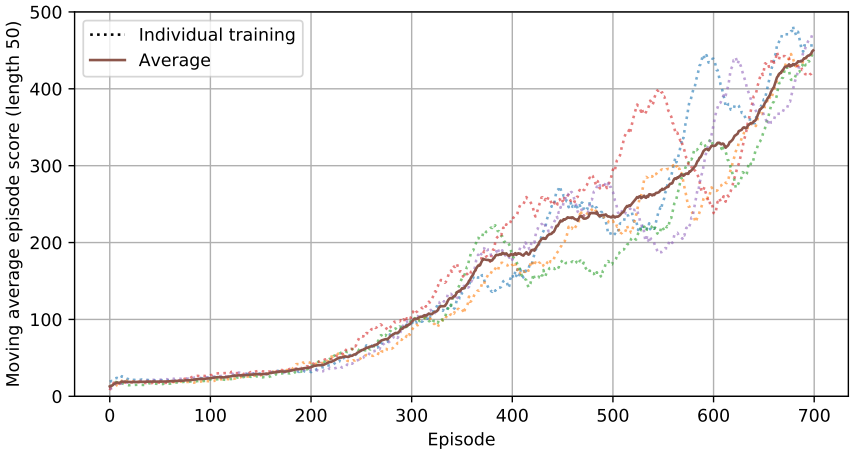


Figure 2: Moving average episode scores from five training runs of the adaptive ε version (see section 3.2). Solid line is the mean of all five; this appears in figure 3. Episodes ended after max 500 steps.

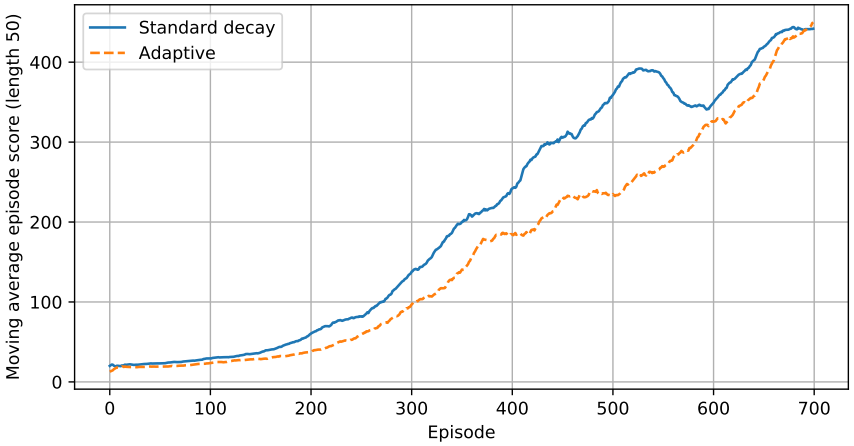


Figure 3: Comparison of the mean of moving average episode scores of the standard ε decay algorithm and the adaptive version, from sections 3.1 and 3.2. Mean is taken over five training runs for both graphs. Episodes ended after max 500 steps.

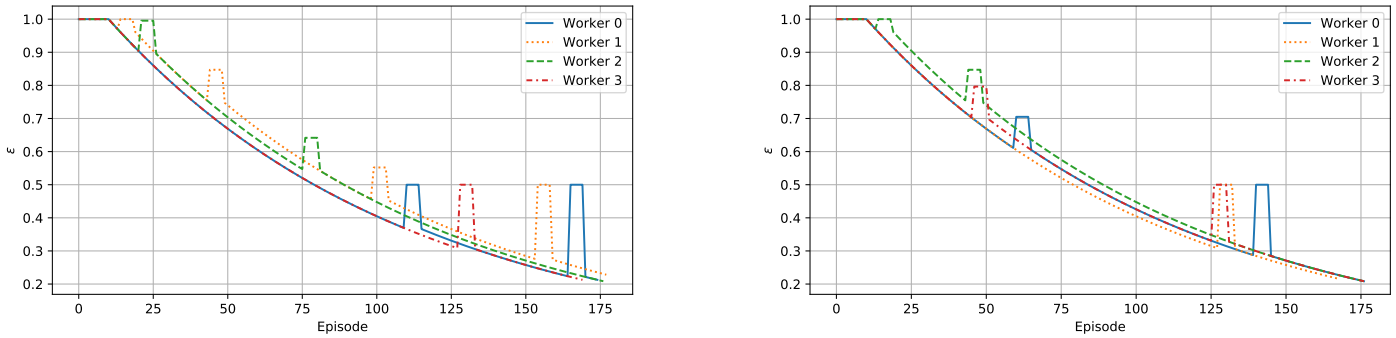
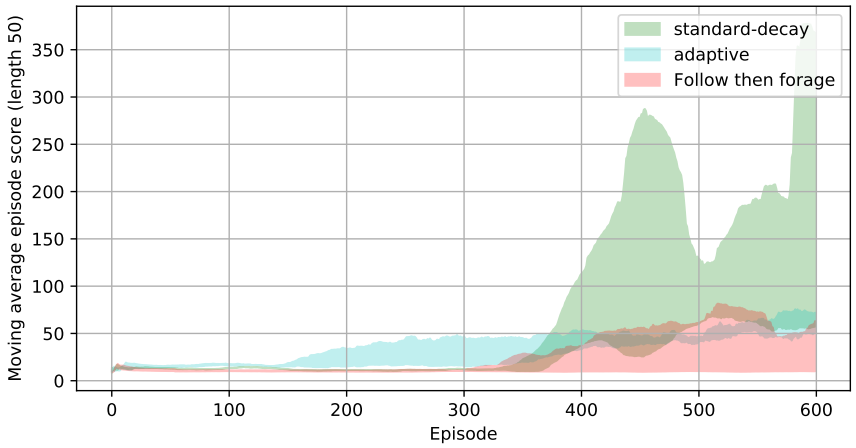
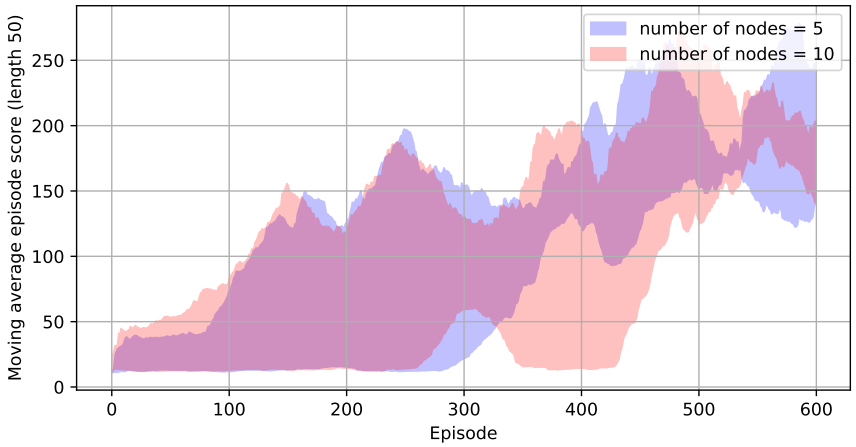


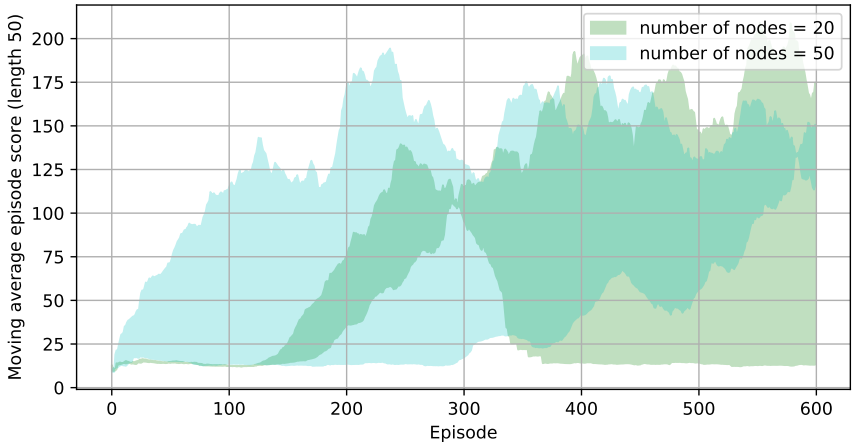
Figure 4: Two examples of how the ε pulses appear during training of the adaptive algorithm described in 3.2.



(a) Standard ε-greedy decay, the adaptive method introduced in [5] and lastly follow then forage from [6]. While all three in general stay in the region of moving average scores between 10-60, one episode "blew up" for the standard ε-greedy decay algorithm.



(b) Grid-ε with 20 respectively 50 nodes. Both algorithms steadily increase and fluctuate less for higher episodes.



(c) Grid-ε with 20 respectively 50 nodes. The grid with n = 50 fluctuates more when fewer episodes have passed and increasingly becomes more stable. While the lowest values reached when n = 20 is an outlier, there are still more fluctuations compared to when n = 50.

Figure 5: Grid-ε for the last two figures where each node has a standard ε-greedy algorithm accompanied with it. The intervals display the highest and lowest averages of scores of 10 training rounds and here the learning rate has been set to 0.001 to display unstable behaviour. Figure a) is for comparing purpose.

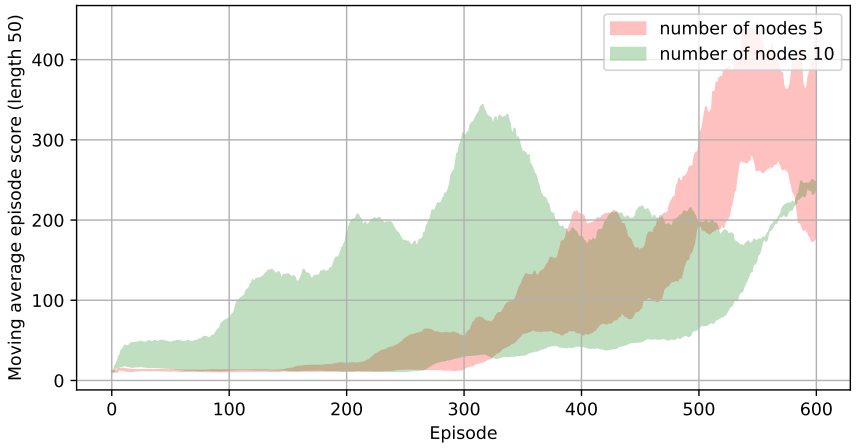


Figure 6: Moving average episode scores from ten training runs of the grid-ε with standard ε-greedy decay. The decay rate of an ε in the grid has been set to 0.98 instead of 0.99.