Deep reinforcement learning agent tuning framework

Aleksei Ivanov

University of Applied Sciences Osnabrück
Faculty of Engineernig and Computer Science
Barbarastr. 16, D-49076 Osnabrück
aleksei.ivanov@hs-osnabrueck.de

'Advanced Seminar' Summer 2020

June 25, 2020

Abstract

The topic of this paper is an attempt to build a systematic approach to reinforcement learning agent tuning by conducting a number of experiments in a custom grid world environment. This environment implements agent vision as a state representation instead of absolute position making it more complex and similar to computer vision state representation while still keeping it simple enough to be able to adjust it. A series of experiments is conducted with various hyperparameter configurations and results are evaluated. In the end, a proposal for the new adaptive parameter method regarding memory sample size and capacity is discussed.

Keywords – reinforcement learning, deep learning, neural networks, tuning, parameters

1 Introduction

Even though the majority of classic reinforcement learning (RL) algorithms, such as k-armed bandits and table Q-learning [Sut18] are somewhat deterministic and do not have many parameters, with recent improvements in machine learning algorithms and successful applications of deep neural networks RL algorithms got the second wind in the light of much more complex function approximators which neural networks can successfully applied as.

One of the most complex aspects of applying machine learning algorithms based on neural networks in general and deep RL algorithms in particular is the fact that these algorithms have vast amount of hyperparameters – parameters, which affect either neural network architecture or the algorithm learning process. This makes it hard to get these parameters right from the start without trial and error which in turn adds a great deal of overhead between algorithm installation or implementation and its usage phases.

Therefore, the there is an emerging need for methodologies which would allow to reduce the time that is needed for algorithm tuning instead focusing more on implementation details of the task at hand. The broader the scope of the given methodologies will help make them more general, in turn providing a framework for RL algorithm tuning as a system of method which could be applied to specific classes of algorithms.

Hence the aim of this work is to try and systemize already existing methods of machine learning hyperparameter tuning in the realm of reinforcement learning in particular, mainly because of the current active developments in this field in comparison to classification algorithms and computer vision. The aim of this work is going to be achieved through the review of already existing state of the art RL algorithms in discrete state and action spaces [Sut18] to be able to lay down the groundwork for this kind of methodology, because this is the simplest type of action space available in reinforcement learning.

Presented solution in this paper consists of a custom grid world environment, which implements RL agent area of vision of nearby grids, area of which can be customized, providing adjustable algorithm hyperparameter. In the end there are conclusions regarding the effects of hyperparameter adjustments and an effort to systemize these effects by concluding them into a methodological framework.

The reinforcement learning agent is implemented using a simple deep Q-learning algorithm in order to make it easier to observe and analyze changes which occur with the change of algorithm hyperparameters.

In order to systematically conduct experiments and record their results a custom software implementation is used, which allows to run multiple experiments for result aggregation and averaging which in turn provides a statistically more robust representation of experimental results.

Before running experiments, a methodology developed, which provides a systematic approach to attribute changes and monitoring of the effects of those changes.

In the last part of the paper experiments are conducted and results are analyzed. After that there is a discussion, concluding experiment results and highlighting further scope of the work.

2 Concepts and Methods

Since its resurgence in popularity with the advances of deep learning, reinforcement learning algorithms have seen a number of improvements over the last years, such as approaches to learning without exploration [Fuj19], multi-agent learning [Foe16], [Doy02], self-learning [Eva17] and knowledge transfer [Geo03]. In this part of the paper basic description of reinforcement is given along with the highlights of current state of the art techniques in the area.

2.1 The concept of reinforcement learning

In very broad terms reinforcement learning problems involve learning what to do in context of how to map situations to actions so as to maximize a numerical reward signal [Sut18]. In can be represented as a closed-loop system because by the nature of the algorithm current alterations in its strategy affect how it will act down the line (fig. 1).

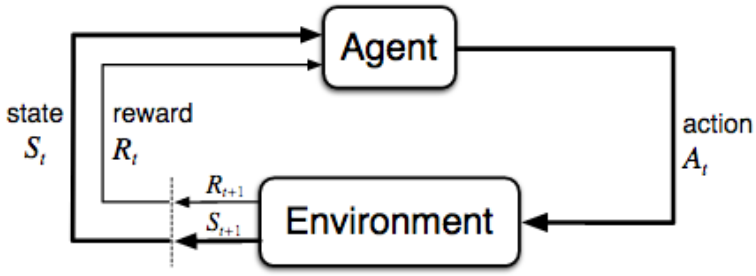


Figure 1: Typical agent–environment interaction in reinforcement learning [Sut18].

As a research direction, reinforcement learning provides a branch of machine learning alongside supervised and unsupervised learning and on surface they might seem similar, especially when neural networks are applied. However, it is worth mentioning that reinforcement learning is much less related to traditional machine learning tasks of classification, instead using advances in those fields as building blocks for higher efficiency and applicability.

For example, reinforcement learning algorithms enjoy advances in neural network stochastic gradient descend algorithm modification such as Adam [Kin15] and more recently Adagrad (adaptive gradients) [Duc11] which can provide higher convergence speed.

More impressively advances in deep learning have found their application in reinforcement learning with the use of convolutional neural network architectures [Cou16] which give artificial intelligence agents to learn from image state representations, such as camera feeds or computer game screenshots [Mni13].

2.2 Current advancements

Current advancements in reinforcement learning are trying to solve newly occurring problems which will allow to use them in a wider range of applications, the most prominent being robotics. New developments are focusing on continuous state and action space representations while also making efforts in improving learning algorithms using approaches specific to this type of machine learning.

One of the earliest attempts at improving learning performance were experimented with by employing multi-agent learning [Doy02], which is still a somewhat relevant research topic [Foe16]. This approach focuses on the usage of multiple reinforcement learning agents which could be connected through a common knowledge basic (usually a neural network) or some other means. Such approach facilitates knowledge transfer, which has been studied in such works as [Geo03] and more recently in [Che15], which provides a framework for the usage of pre-trained neural network as a starting point and extending it with a higher amount of neurons.

To reinforcement learning specifically there is a problem of policy acquisition, which can be affected in a number of ways. Works such as [YNg03] lay the groundwork for research in the field of reinforcement learning agent reward shaping and policy search algorithms. In their work [YNg00], authors propose an algorithm called Pegasus which allows search for an optimal policy via transformation of a partially observable Markov decision process (POMDP) to the one, where all transitions are deterministic. After that they provide a way of approximating newly formed POMDP's initial state distribution.

More recently with advances in robotics a problem of learning without exploration [Fuj19] emerged, coming from the high cost and dangers of stochastic policy search in real environments new methods have been developed, which allow agent to learn without exploration or by watching an a human [Gab19] do a number of examples first, which can guide the algorithm to converge faster to an optimal policy without the risks of costly damage [Abb10]. Another approach to this type of learning could be implemented by integrating human feedback instead of doing a number of specific examples [Gri13].

Alongside all those methods an interesting approach was proposed in [Eva17], where authors show how reinforcement learning agent can use self-supervision in order to receive more feedback from its actions through frequent rewards which reflect additional losses which reflect current agent performance avoiding the problem of sparse rewards which in turn aids in representation learning (state-action pairs).

2.3 Problem area

Current machine learning algorithms become more and more complex, however there is a lack of research in how state representation and memory capacity affect the learning process along with classic hyperparameters, such as neural network learning rate (which is usually not specifically linked to reinforcement learning) and discount rate – instead there is a notion that modern algorithms "just work" without much systematic approach.

Recently there have already been attempts to stand back and rehearse what is already known about machine learning algorithms [Jin19] where authors try to reflect on why simple algorithms work well. This work is an attempt to continue to widen systematic approach to reinforcement learning algorithm tuning.

2.4 Methodology

In this work several experiments are conducted in a specially built grid world environment which provides reinforcement learning agent with a state representation in a form of vision around them in contrast to classic state representation in such world which is usually an absolute position. The grid world is used instead of complex computer vision task because it is a much easily controllable environment, hence it provides an opportunity to get better insights. The aim of the experiments is to verify and try build a systematic approach to agent hyperparameter choice, such as learning rate, future reward discount factor and state space dimensionality through the tweaking of agent vision. The computer program will run the same experiment multiple times, averaging historic values of model training losses and agent rewards, which improves statistical robustness of gathered data.

3 Experimental Study

3.1 Proof-of Concept Implementation

Grid world configuration is stored in a separate text file which gives the flexibility of changing it without the need to change program code. Each world has an accompanying JSON file, which describes idle reward for agent standing on a free tile and the type of bounding tile. Each tile is represented as an ASCII symbol, making it easy to render in terminal. In the world description file, there is also a list of entity descriptions which list entity symbols and their associated name and reward, when agent steps on it.

```
EPISODE: 68 (EXPERIMENT: 1)
CUMULATIVE REWARD: -4950
EXPLORATION RATE: 0.04473351
L.....  ←↑↓↑↑↑
.W.W.W  ↑ ↑ ↑
.W.W.W  ← ↑ ↑
.W.W.W  ↑ ↑ ↑
WW.W.W    ↑ ↑
...W.G  ↑←↑ ↑↑

WWWWW
WWWWW
WWL..
WW.W.
WW.W.

PREFERRED ACTION: left

. - Empty tile
L - Lizard
W - Wall
G - Goal
```

a) View during policy learning

```
EPISODE: 143 (EXPERIMENT: 2)
CUMULATIVE REWARD: -2605
EXPLORATION RATE: 0.01081685
L.....  →→→→↓→
.W.W.W  ↑ ↓ ↓
.W.W.W  ↑ ↓ ↓
.W.W.W  ↑ ↓ ↓
WW.W.W    ↓ ↓
...W.G  →→↓ →→

WWWWW
WWWWW
WWL..
WW.W.
WW.W.

PREFERRED ACTION: right

. - Empty tile
L - Lizard
W - Wall
G - Goal
```

b) View when a policy has been learned

Figure 2: Developed program view, showing grid world, agent policy and its vision in two cases: during policy learning (a) and when a policy has been learned (b).

Fig. 2 shows program view which renders given grid world, current agent policy (on the right) and agent vision around them (under the grid world). Agent implements a simple deep Q-learning (DQN) algorithm with additional target neural network [Her19] which provides higher stability during learning process.

Environment is read from a text file and each non-empty grid is transformed into an entity – an object abstraction, which allows other objects to interact with this grid. For example an entity which is represented with "W" is a wall and agent will not be able to move through it, while entity marked as "H" is a hole in a similar way as in a classic frozen lake grid world [Bro16].

Agent state representation is a two-dimensional array where the first dimension corresponds to a grid in agent's field of view while the second dimension is one-hot encoded representation of a given grid. This array is further flattened in order to match the input of a fully connected neural network. For instance, with a field of view of 5x5 and 4 distinct grid types, the state representation size is a vector with the length of 100, making it impractical to use table, hence why in this work a DQN approach is used.

3.2 Experiments

All experiments were carried out in a maze grid world (Fig. 2) where agent is not be provided any aid in the form of intermediate rewards (i.e. crumbs along the path). This will ensure long-term reward prioritization of the agent. In fact, as it will later be shown, providing such intermediate rewards manually might even make learning process worse, essentially confusing the agent about correct reward.

Change of parameters was done by starting with a first set of parameters (experiment 1 in table 1) and then changing one parameter until performance (average reward) improves. After a new value of parameter is found another parameter is tweaked. If the new value of parameter doesn't improve performance, the next set of experiments can use previous values of the parameters.

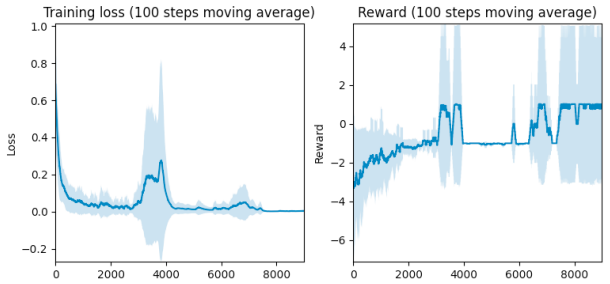
First set of experiments was changing of memory sample size – the number of experiences that agent's neural network is learned on, effectively acting as a learning data set. After an optimal memory sample size is found – the one after agent is not able to find a better reward, then other hyperparameters are tuned – such as future rewards discount rate [How15], making an attempt to further improve learning performance.

Because the environment is deterministic, agent learning rate of 1 is optimal, hence it is not changed during the experiments. In the same fashion the learning rate of neural network does not affect the perception of the agent, hence it is left unchanged at the value of 0,001. Over the course of the experiments parameters which affect agent's perception and memory are tuned, such as memory replay sample size and reward discount factor.

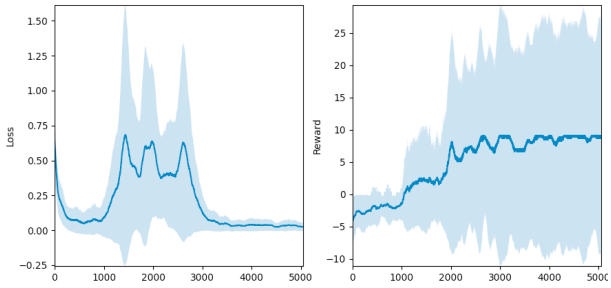
3.3 Experiment 1

In the first experiment the value of replay memory sample size is changed while memory capacity, vision range and discount rate stay the same. Fig. 3 shows how different values of memory sample size affect reinforcement learning agent learning process, especially judging by the reward graph.

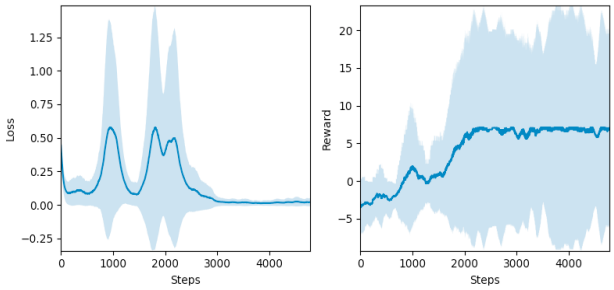
Replay memory serves the role of neural network training data set where the data is sampled from the environment as state representations, taken actions, reward and the next state at a given time. One such sample is usually represented as a vector (s, a, r, s') while memory contains a number of the most recent observations.



a) Memory sample size: 32

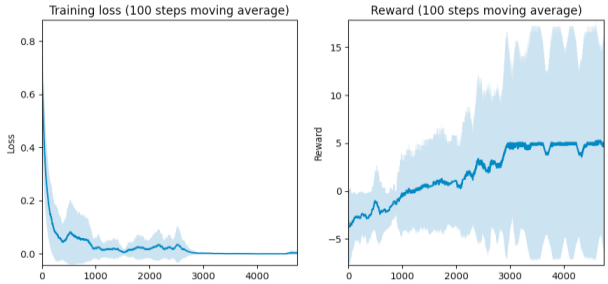


b) Memory sample size: 64

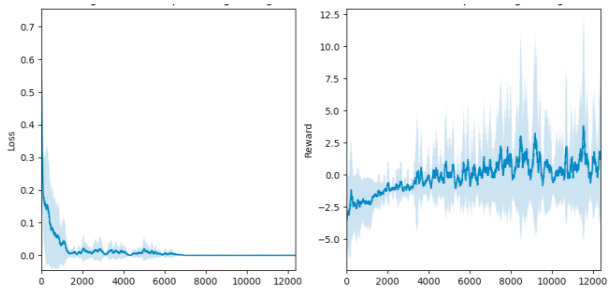


c) Memory sample size: 128

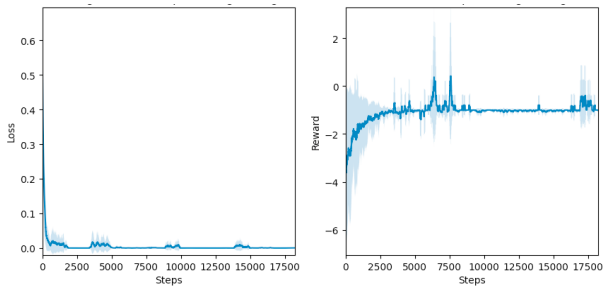
Figure 3: Differences in loss and average reward depending on memory sample size



a) Discount factor: 0.50



b) Discount factor: 0.25



c) Discount factor: 0.10

Figure 4: Differences in loss and average reward depending on discount factor

Judging by the Fig. 3 there is a visible change in learning performance of the agent with regard to the obtained reward. It also makes the learning process unstable, making big jumps in reward over time. This suggests that there is a minimum required sample size for the agent, which enables a more stable learning process. The downside of picking high memory sample size is slower learning due to the higher computational demand. Because of this it is not trivial to pick the correct sample size, since the typical rule of 70/30 from supervised learning does not work in reinforcement learning. In this experiment the optimal sample size is 64, not only because it is faster, but the average reward is even higher with this sample size.

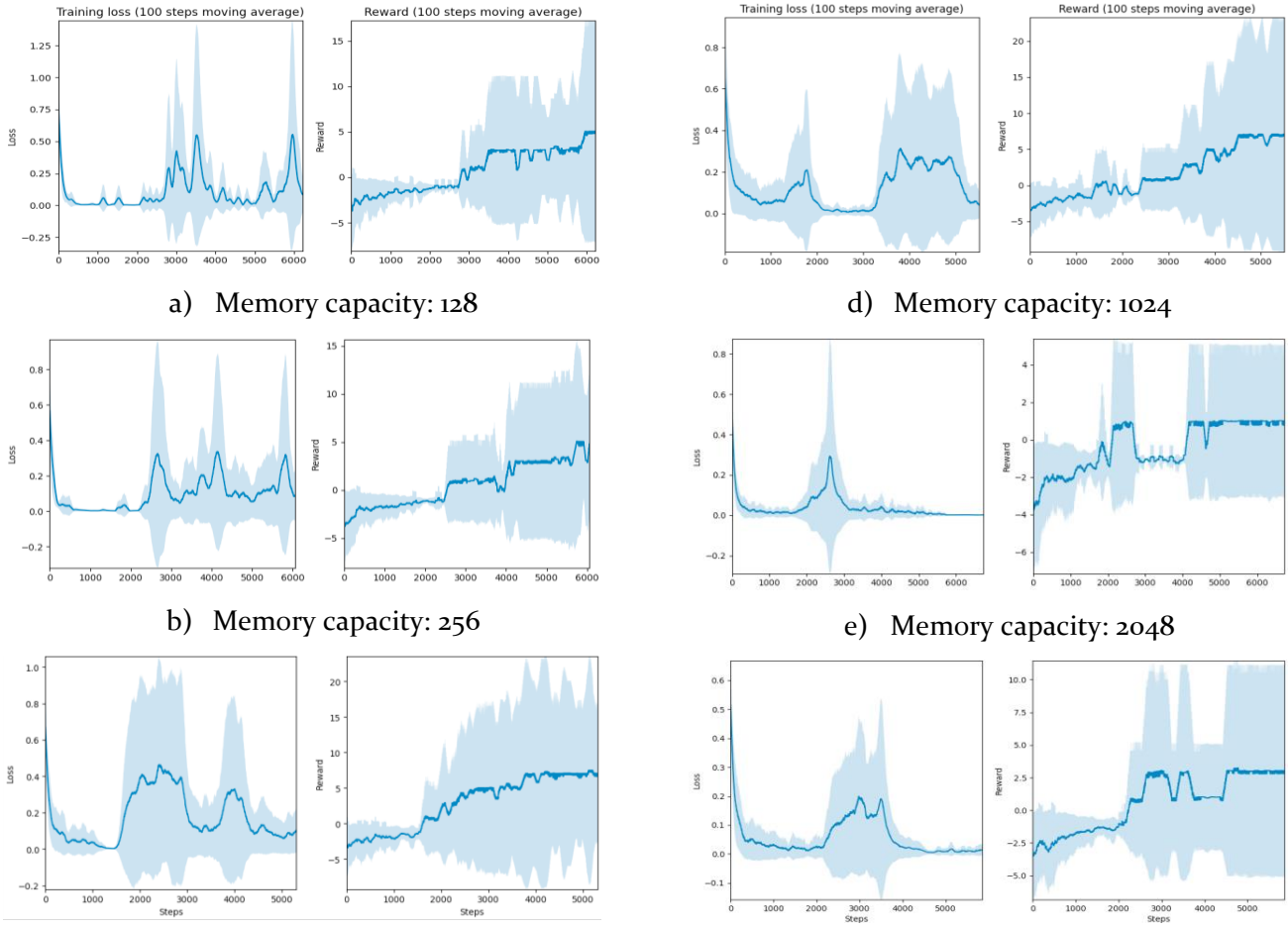
3.4 Experiment 2

After the optimal sample size value is found it is chosen as a baseline for the second experiment, where discount rate is changed. On Fig. 4 it is visible that the less the discount factor becomes, the harder it is for the agent to learn an optimal policy. In case when discount factor is 0.10, the agent reaches plateau in reward rather quickly without further improvement. This further reinforces the fact that in most cases the value of discount factor should be nearing 1. It is worth noting that even when the discount factor is twice as smaller at the value of 0.50 (Fig. 4a) agent almost manages to get to the highest reward (comparing to

Fig. 3b). However, it takes more time to get to the plateau, which suggests that because of the lower discount factor agent rejects useful information during training.

3.5 Experiment 3

The next experiment involves changing memory capacity, instead of sample size. This affects agent "long-term" memory , which in turn affects how many previous experiences the agent "remembers", however too large of memory capacity not only consumes more computer memory, but can also potentially affect the learning performance if the random batch of memory samples will be picked from older experiences – focusing agent's learning on previous experiences which it might has already learned, but still are present in memory. On Fig. 5 six experiments are shown with varying memory capacity.



a) Memory capacity: 128

d) Memory capacity: 1024

b) Memory capacity: 256

e) Memory capacity: 2048

c) Memory capacity: 512

f) Memory capacity: 4096

Figure 5: Differences in loss convergence and average reward at different memory capacities

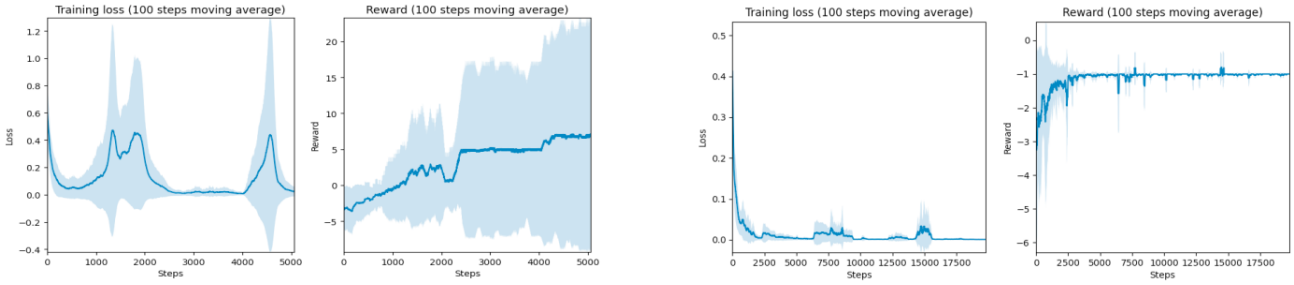
There is a noticeable learning performance degradation (in terms of average reward), when memory capacity is increased past the value of 1024, which suggests that the increase in memory capacity makes it difficult for the agent to pick more recent experiences, in turn slowing down the learning process. Table 1 shows relation between memory size, loss convergence time in steps and average reward achieved by the time of convergence.

Run No.	*Memory size*	*Loss first convergence (in steps)*	*Average reward at convergence point*	*Final approx. average reward*
3a	128	300	-2.50	2.50
3b	256	500	-2.00	3.00
3c	512	1500	-2.00	6.00
3d	1024	**2500**	**0.00**	**7.00**
3e	2048	1500	-1.00	1.00
3f	4096	2000	-1.25	2.50

Table 1: Number of steps until model error first convergence and respective average reward

3.6 Experiment 4

In this experiment the vision range of the agent was changed first to 3 units (fig. 6a) and then to the smallest possible range of 1 unit (fig. 6b). While giving extra information around the agent did not improve its learning performance, restricting visible area made it impossible for the agent to learn in the second case.



a) Vision range: 3 units b) Vision range: 1 unit

Figure 6: Differences in loss and average reward with various agent vision ranges

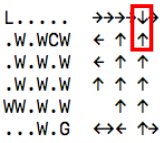
This could have been the case because of the identical state representations due to the limited range of agent vision. Fig. 7 shows a case when two different states have identical state representations, even though agent must distinguish between them in order to reach the goal. By decreasing the dimensionality of the state agent loses ability to adequately judge their surroundings and therefore are incapable of choosing correct actions.

```
L.....      L.....
.W.W.W      .W.W.W
.W.W.W      .W.W.W
.W.W.W      .W.W.W
WW.W.W      WW.W.W
...W.G      ...W.G
```

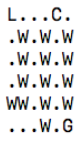
Figure 7: Two different states with identical state representations

3.7 Experiment 5

In the final experiment the optimal values of memory capacity and sample size along with discount rate are chosen. In such configuration a crumb is added along the path to the goal (Fig. 8), which gives additional reward to the agent along the path and in theory should help it find a better policy faster.

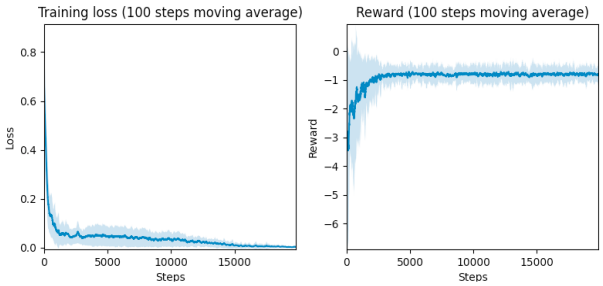


a) Crumb inside the corridor

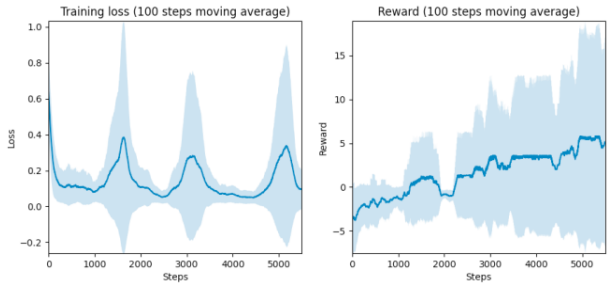


b) Crumb outside the corridor

Figure 8: Crumb added on a path to the goal. Marked area highlights learned policy inability to determine correct path



a) Performance with a crumb placed inside the corridor



b) Performance with a crumb placed outside the corridor

Figure 9: Loss and average reward in an environment with an added crumb

However, as can be seen on Fig. 9a this is not the case. Instead agent quickly learns the wrong policy and is unable to recover from it. This could have happened because of the change in state representation of the agent. Since each entity surrounding the agent is one-hot encoded the change of entity type (to empty tile) might have had a drastic effect on the agent policy. Curiously enough on Fig. 9b the agent is able to learn the policy, but the reward gain is much less steep, which suggests that agent is struggling to learn an optimal policy.

3.8 Results

The results are presented in Table 2 with all combinations of hyperparameters used in experiments alongside final average reward from previous experiments. Numbers in bold are the best for the given subset in an experiment. Judging by the table, the best approximate average reward is achieved when memory sample size and memory capacity were in the middle – both not too high and too low. This shows that an optimal value for these parameters can be found empirically.

Experiment No.	*Vision range*	*Discount factor*	*Memory size*	*Memory sample size*	*Final approx. average reward*
1a	2	0.99	1024	32	0.75
1b				**64**	**7.00**
1c				128	6.50
2a		**0.50**		64	**5.00**
2b		0.25			0.50
2c		0.10			-1.00
3a			128		2.50
3b			256		3.00
3c			512		6.00
3d			**1024**		**7.00**
3e			2048		1.00
3f			4096		2.50

4a	1		1024		-1.00
4b	**3**				**6.00**
5a	2				-1.00
5b					**3.00**

Table 2: Configurations of experiment parameters and final rewards for each set

4 Discussion

Conducted experiments in this paper have shown that configuration of parameters specific to reinforcement learning algorithms affect their performance significantly. In experiment 2 (Fig. 4) it was shown that decreasing the value of discount factor makes learning process significantly worse and adds visible noise to the reward graph, which suggests agent acts are more random and its inability to learn a stable policy.

Another notable effect on performance was observed with the change of memory sample size (experiment shown on Fig. 3). Decreasing memory sample size effectively makes machine learning algorithm training dataset very sparse and in turn learning highly biased which affects optimal policy search. On the other hand, increasing memory sample size significantly increases computational cost, because this batch of training data has to be applied on each training step of the agent, hence just choosing a high value for the batch size is not practical and therefore development of methods in this direction is relevant.

Experiment 3 changes memory capacity instead of sample size, which gives an effect of long-term or short-term for the agent. As can be seen in Table 2, either decreasing or increasing memory capacity does not necessarily makes agent learn better. Counter-intuitively, increasing the capacity of memory makes learning process worse than when capacity is decreased, which suggests that the agent learns better with shorter-term memory. This fact might find reflection in the maximum number of steps that agent is allowed to make during the episode until environment resets.

Changes of agent vision range in experiment 4 (Fig. 6) shown that state representation must be large enough to be able to represent current state **absolutely** which means it should either take into account previous experiences (like number of steps) or a higher dimensionality of state representation. Fig. 7 shows that with a small vision range, agent is unable to differentiate between two separate states which are different relative to the environment (absolute), however **relative** to the agent they are the same.

The last experiment involved environment augmentation in a form of adding a crumb with gave the agent additional reward on the path to the goal. Interestingly enough this did not improve learning performance of the agent while on the opposite made it impossible for it to learn a policy (Fig. 9a). Even though reaching the goal would give the agent much bigger reward, agent instead focusing on the crumb. This shows that adding intermediate rewards manually not always can benefit the learning process, therefore other methods against sparse rewards should be used.

5 Summary and Outlook

Experiments conducted in this work have shown that there is an empirical way to tune machine learning parameters specific to reinforcement learning mainly focusing on memory. Topic of state representation and environment augmentation were touched and showed that there always must be enough of fidelity in state representation for the agent to distinguish current state. Environment augmentation took place

in the form of placing a crumb on the path to the goal in hope to reinforce the agent to follow the path, however experiments have shown that in practice this is not the case. Instead manually created artificial intermediate reward interfered with agent exploration of the environment and in one case did not allow agent to fully explore the it.

Based on the experiments memory sample size affects learning process significantly and therefore it is crucial to have an ability to choose the value of sample size using systematic approach. Hence, one of further directions of research could be the development of an adaptive memory sample size algorithm similar to the one existing for neural network learning rate (Adagrad [Duc11]). Such algorithm could take into account average step count during each episode or some other metric. Another approach to memory adaptation might involve experience weighting based on correlation with given situation, giving context to certain memories in certain situations.

The idea of adaptive memory has been touched upon in the works such as [Rui18], however presently there are no widely adopted methods for this, therefore further research is needed.

References

[Abb10] Abbeel P.; Coates A. und Ng A.Y. (2010): Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29, 13, 1608-1639.

[Bro16] Brockman G.; Cheung V.; Pettersson L. et al. (2016): *OpenAI Gym*. arXiv preprint arXiv:1606.01540.

[Che15] Chen T.; Goodfellow I. und Shlens J. (2015): Net2Net: Accelerating Learning via Knowledge Transfer. In: *ICLR 2016*..

[Cou16] Courville I.G.a.Y.B.a.A. (2016): Chapter 9, Convolutional Networks. In *Deep Learning*. MIT Press.

[Doy02] Doya K.; Samejima K.; Katagiri K. et al. (2002): Multiple model-based reinforcement learning. *Neural Computation*, 14, 6, 1347-1369.

[Duc11] Duchi J.; Hazan E. und Singer Y. (2011): Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In: *Journal of Machine Learning Research 12*.. S. 2121-2159.

[Eva17] Evan S.; Mahmoudieh P.; Argus M. et al. (2017): *Loss is its own Reward: Self-Supervision for Reinforcement Learning*. arXiv preprint arXiv:1612.07307.

[Foe16] Foerster J.N.; Assael Y.M.; de Freitas N. et al. (2016): Learning to Communicate withDeep Multi-Agent Reinforcement Learning. In: *30th Conference on Neural Information Processing Systems (NIPS 2016)*. Barcelona, Spain.

[Fuj19] Fujimoto S.; Meger D. und Precup D. (2019): Off-Policy Deep Reinforcement Learning without Exploration. In: *Proceedings of the 36th International Conference on Machine Learning*. California.

[Gab19] Cruz G.V.d.l.; Du Y. und Taylor M.E. (2019): Pre-training with non-expert human demonstration for deep reinforcement learning. *Knowledge Engineering Review*, 34.

[Geo03] Konidaris G. und Barto A. (2003): Autonomous shaping: Knowledge transfer in reinforcement learning. In: *Proceedings of the 23rd international conference on Machine learning*.. S. 489-496.

[Gri13] Griffith S.; Subramanian K.; Scholz J. et al. (2013): Policy Shaping: Integrating Human Feedback with Reinforcement Learning. *Advances in neural information processing systems*, 2625-2633.

[Her19] Hernandez-Garcia J.F. und Sutton R.S. (2019): *Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target*. arXiv preprint arXiv:1901.07510.

[How15] (2015): *How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies*. NIPS 2015 Deep Reinforcement Learning Workshop.

[Jin19] Jin C. (2019): *Machine Learning: Why Do Simple Algorithms Work So Well?* (PhD Thesis) EECS Department, University of California, Berkeley.

[Kin15] Kingma D.P. und Lei Ba J. (2015): Adam: A Method for Stochastic Optimization. In: *3rd International Conference for Learning Representations*. San Diego.

[Mni13] Mnih V.; Kavukcuoglu K.; Silver D. et al. (2013): *Playing Atari with Deep Reinforcement Learning*. arXiv preprint arXiv:1312.5602.

[Rui18] Ruishan L. und Zou J. (2018): The Effects of Memory Replay in Reinforcement Learning. In: *56th Annual Allerton Conference on Communication, Control, and Computing*. IEEE.

[Sut18] Sutton R.S. und Barto A.G. (2018): *Reinforcement Learning: An Introduction*. Cambridge: MIT Press.

[YNg00] Y. Ng A. und Jordan M. (2000): *PEGASUS: A policy search method for large MDPs and POMDPs*.

[YNg03] Y. Ng A. (2003): *Shaping and Policy Search in Reinforcement Learning*. (PhD Thesis) University of California, Berkeley.