Advanced Machine Learning (WS 2020/21)
Programming Project

Author 1
Last name: Hector
First name: Villeda
c-Number: 7031297

Author 2
Last name: Binder
First name: Michael
c-Number: 11843166

1 Introduction

The Acrobot-v1 environment is a continuous state, discrete action environment of the gym library from OpenAI. It represents two joints and two links and the goal is to swing the end of the lower link up to a given height. The environment has six continuous states and 4 possible, discrete actions. Since multiple reinforcement learning (RL) algorithms are able to learn and perform well in the Acrobot environment, it was the goal of this work to provide a concise comparison of selected approaches and point out important settings, which lead to a successful training of the agents. Like expected, classic Q-learning or SARSA agents could not reach a sufficient level of performance, because of the continuous state space, but adjustments to these algorithms in terms of a DQN and variations like a policy network led to more than satisfying results.

2 Methods

Tackling an environment with an infinite number of states (continuous) requires a function approximator to at least represent the value or action-value function within the RL algorithms. Neural networks (NNs) work great in this context and become even more powerful, when approximating the policy of their respective agent directly. Both approaches are described and analyzed in the following sections.

2.1 Agents with Q-network

Based on the temporal difference TD(0) prediction, two control algorithms evolve. Since the TD target is a biased estimate of the true return (in other words, the target depends on the weights of the network, which are updated every step), control algorithms including the TD target do not result in a true gradient-descent method and are called semi-gradient algorithms. Both semi-gradient algorithms, the off-policy Q-learning and on-policy SARSA algorithm, have been analyzed for this report under consideration of the so-called Deadly Triad, a combination of three properties (function approximation, bootstrapping and off-policy learning) that is supposed to significantly reduce convergence guarantees.

A by now common solution to deal with the Deadly Triad and improve convergence properties of the Q-learning algorithm was first presented by Google's Deepmind team in 2015 (Mnih et al., 2015). Their Deep Q-network (DQN) uses experience replay and a target network to avoid forgetting and improve the learning process. Here experience replay refers to a memory accumulation of state-action pairs and corresponding rewards, so that the agent can sample mini-batches out of this collection and learn from multiple past experiences at once. This helps to avoid learning from a sequence of highly correlated state-action pairs like in a classic semi-gradient approach, whereas a target network is supposed to significantly stabilize the learning process. The target network represents a copy of the Q-network. It is held constant for a specified number of steps and is used for estimating the return

based on the TD prediction. It allows the agent to improve its policy towards a constant target for some steps, instead of changing the target with every update to the policy, too.
Section 4.2 shows the results of a Q-learning agent modified with the ideas of a DQN. A DQN version of the SARSA algorithm has also been implemented and tested, but experience replay motivates the choice of off-policy learning (Q-learning), because the network's parameters change during the learning process and are different to those that generated the samples of the mini-batch.

2.2 Agents with policy-network

An alternative to learning the action-value function is learning the policy of the agent directly. In many cases it might actually be easier to learn, for example, if the agent has to move left or right instead of learning a complicated value of a certain state or state-action pair. Furthermore, these so-called policy gradient algorithms usually show better convergence properties in continuous environments, because, generally, small changes to the policy parameters Θ do not result in large changes in the actions to be taken, a known issue of semi-gradient agents.
Actor-Critic methods expand the idea of policy gradients by introducing an action-independent baseline to the evaluation process to address the high variance of the classic Monte-Carlo (MC) policy gradient algorithm. The following sections further describe and compare the episodic MC Advantage Actor-Critic (MC-A2C) and the one-step Advantage Actor-Critic (TD(0)-A2C) algorithm.

3 Setup

Object-based programming with separate scripts for agents, networks, training and evaluation provides an effective and convenient setup to train multiple agents. Seeds for the environment and weight initialisation guarantee total reproducibility, while an exponential moving average (EMA) is constantly calculated during training for a clear visualization of the training progress.
The agents and networks from the course assignments created the basis for the implemented agents. All use the default Adam optimizer of Pytorch to update the weights of all networks and a Xavier-normal weight initialization. Though a Kaiming initialisation is recommended when using rectified linear unit (ReLU) activation functions, it is probably considered negligible for networks with two or less hidden layers (He et al., 2015). A Re-initialisation of the network weights is performed after 150 episodes if nothing has been learned yet (the EMA is below -490). Agent specific settings are described in the respective subsections below, whereas an overview of the used hyperparameters is given in table 1.

	Semi-gradient Q and SARSA	**DQN**	**Policy-gradient (MC, [TD(0)-]A2C)**
α (learning rate)	0.0001	0.0001	0.001
γ_α (α decay rate)	0.999	-	-
γ (discount factor)	0.99	0.99	0.99
ϵ (greedy-factor)	0.3	0.3	-
γ_ϵ (ϵ decay rate after 400 episodes)	0.995	0.995 (& 0.999)	-
τ (softmax-temperature)	1	1	1
Dropout	-	-	0 (& 0.4)
Mini-batch size	-	32	-
Memory size	-	10000	-
C (steps with const target net)	-	200	-
Number of hidden layers	1	2	2
Hidden layer dimension	32 (& 128)	256 (& 128)	512 (& 128)

Table 1: Hyperparameter-overview for all agents

3.1 Semi-gradient Q-learning and SARSA

Semi-gradient control algorithms tend to work better with smaller and less complicated NNs, so a single, fully-connected, hidden layer with a ReLU activation function and no biases in the linear parts approximates the action-value function of these agents. Since fine tuning of the classic hyperparameters in table 1 does not significantly improve our understanding of the agents and also their performance, the upcoming experiments focused on varying the hidden layer dimension and the action selection rule (policy).

3.2 Q-learning with DQN

Experience replay and target networks allow for deeper and larger NNs to approximate the action-value function. In section 4.2 a Q-learning agent with a DQN of 2 hidden layers with ReLU activation functions is presented. Again, biases are set to zero, but the hidden layers have a higher dimension compared to the semi-gradient methods. Though dropout is usually a good idea for larger networks to prevent overfitting, it introduces a higher variance and would counteract the usage of target network, which tries to stabilize the learning process. Furthermore, overfitting is less a problem in RL, because datasets are usually not limited.
The implementation of the experience replay part was inspired by an online tutorial for DQNs (Roth, 2015), but had to be adapted to Pytorch and the agent's framework significantly.

3.3 Agents with policy-network

In Advantage Actor-Critic (A2C) algorithms the Critic represents the Value-function $\hat{V}_\phi^\pi(S)$. Therefore, it is possible to implement the function approximator in two different ways. The first configuration consists of two independent networks for the Critic (Value-function) and the Actor (the policy $\pi_\theta(a|s)$) resulting in independent parameters and loss functions ($L(\hat{V}_\phi^\pi(S))$, $L(\pi_\theta(a|s))$). An alternative is the design of a single NN with two outputs for the Value-function and the policy. In this setup parameters are shared and the loss functions add up to a single loss $L_T = L(\hat{V}_\phi^\pi(S)) + L(\pi_\theta(a|s))$. Since both versions appeared to produce very similar results, the following discussion is solely based on the version with two independent NNs and mainly focuses on variations of the network size and the dropout factor.
Again the network consists of two hidden layers of equal size with ReLU activation functions. The output is now represented with a softmax layer to decide on actions based on a probability distribution.

4 Experiments and Results

Two approaches were used to compare the performance of different agents. For the most part, each agent was trained multiple times for 1000 episodes resulting in a mean and standard deviation of the EMA for each agent. Multiple training runs are important to compare different algorithms, because agents with function approximators can settle for a local optimum which can depend on initial weights and random actions at training start. Furthermore, constantly training for 1000 episodes allows the observation if an agent can keep its performance or if it is subject to forgetting.
In a later stage, the performance of an agent was evaluated by adjusted reward thresholds which stop the training of a single agent close to its best performance (Figure 5).

4.1 Semi-gradient Q-learning and SARSA

Considering the previously mentioned Deadly Triad off-policy Q-learning could be expected to perform worse than the on-policy SARSA algorithm. On average, this thesis was met when training both agents with identical parameters (Table 1), but their high variances aggravate a clear statement. What can be stated though is that both agents performed much better than a random agent (compare figure 1 for the SARSA agent), but did not manage to regularly solve the environment with respect to the reward threshold provided by the environment of -100.

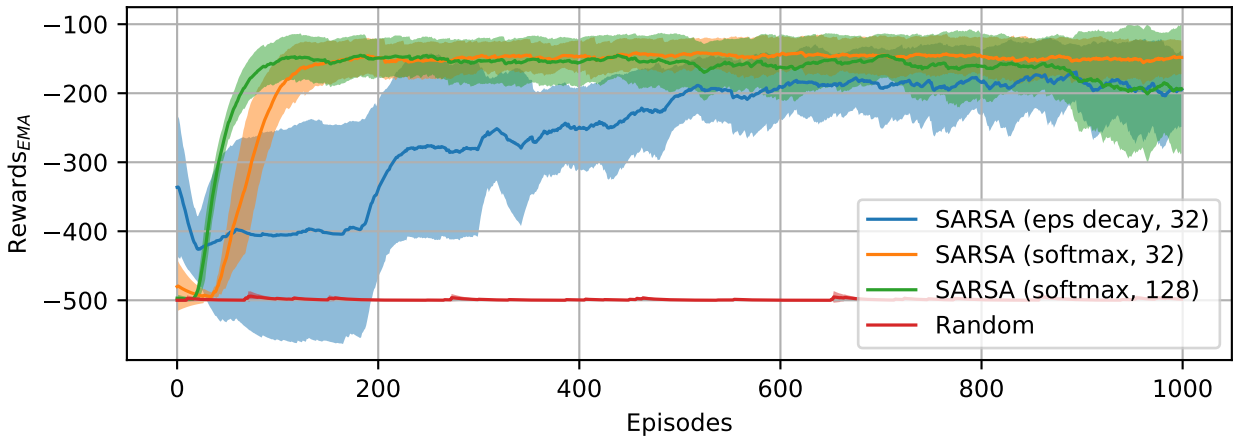


Figure 1: A comparison of semi-gradient SARSA agents with varying policy and hidden dimension of the NN (number in brackets). Visualized are the mean and standard deviation for all agents based on four training iterations.

More interesting was the sensitivity of the agents with respect to the network size and action selection policy. It is intuitive that a smaller hidden layer dimension could increase the training stability, but figure 1 shows that the semi-gradient SARSA algorithm also performs better with a smaller network and with a softmax policy. This is important to keep in mind, because in the next section we will see that softmax actions and too small NNs are clearly hindering the DQN from unfolding its potential.

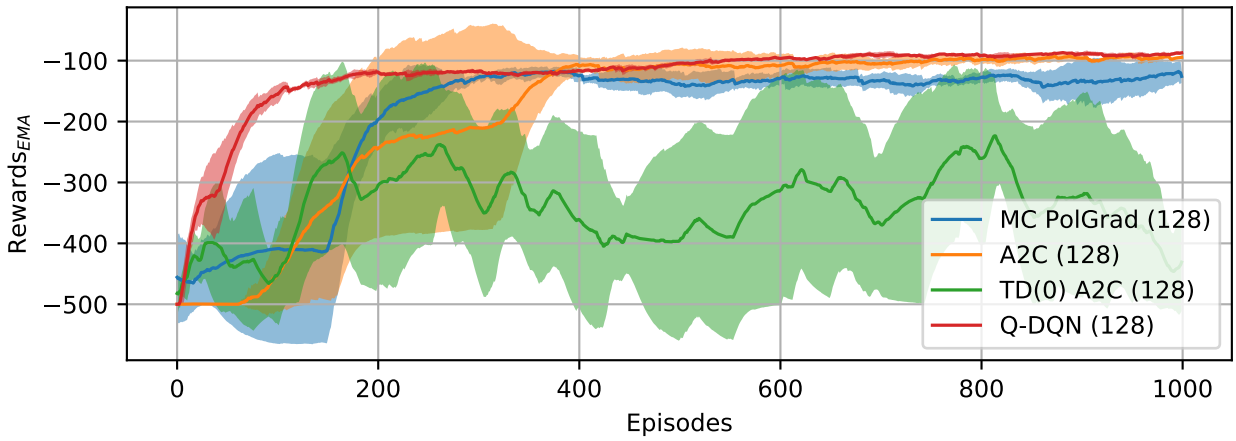
4.2 Q-learning with DQN

Compared to the classic semi-gradient algorithms in the previous section DQNs with experience replay and a target network allow for a larger and deeper network to approximate the action-value function. Furthermore, experience replay compensates the forgetting Q-learning and SARSA algorithms suffered from with an epsilon-greedy policy. This is important, because the performance of the DQN was not superior to a conventional SARSA algorithm when using a softmax policy instead of a decaying epsilon-greedy policy (Figure 3). Most likely, this phenomenon is related to the temperature parameter $\tau = 1$, which can balance the exploration and exploitation of the softmax policy like ϵ for ϵ-greedy (Tijsma et al., 2016). Exploration seems to be underrepresented, so a higher temperature could prevent an agent with a softmax policy from getting stuck in a sub-optimal state.
Figure 3 further shows the significant deterioration in performance and variance of the Q-learning DQN algorithm when neglecting the target network (C=1). These observations, together with marginally tuned parameters, led to the development of a DQN with a hidden layer dimension of 256 neurons, C=200 and a decreased $\gamma_\epsilon = 0.999$ that was able to reach a reward threshold for the EMA of -75 after less than 2000 episodes of training (Figure 5).

4.3 Agents with policy-network

A first look on figure 2b already reveals that both implemented A2C algorithms actually reached a quite similar performance as the DQN of the previous section. This is further supported by figure 5 in the appendix, which shows that all three agents are able to surpass an increased reward threshold of -75 for the EMA compared to the -100 suggested by the environment.
Nevertheless, figure 2 also addresses significant differences between the policy gradient methods. While the classic MC policy gradient (or REINFORCE) and the MC-A2C algorithm performed alright with a dropout factor of 0.4, the TD(0)-A2C fell short of expectations (Figure 2a). It only delivered without dropout, but then with a significantly faster initial convergence compared to the MC-A2C with respect to the number of episodes as visualized in figure 2b. This is plausible, because TD(0)-A2C makes use of bootstrapping and corresponding weight updates after each step whereas Monte-Carlo methods have to wait until the end of an episode to update their network's weights. Initially, TD(0) methods

(a) Policy agents (Dropout: 0.4)



(b) Policy agents (Dropout: 0.0)

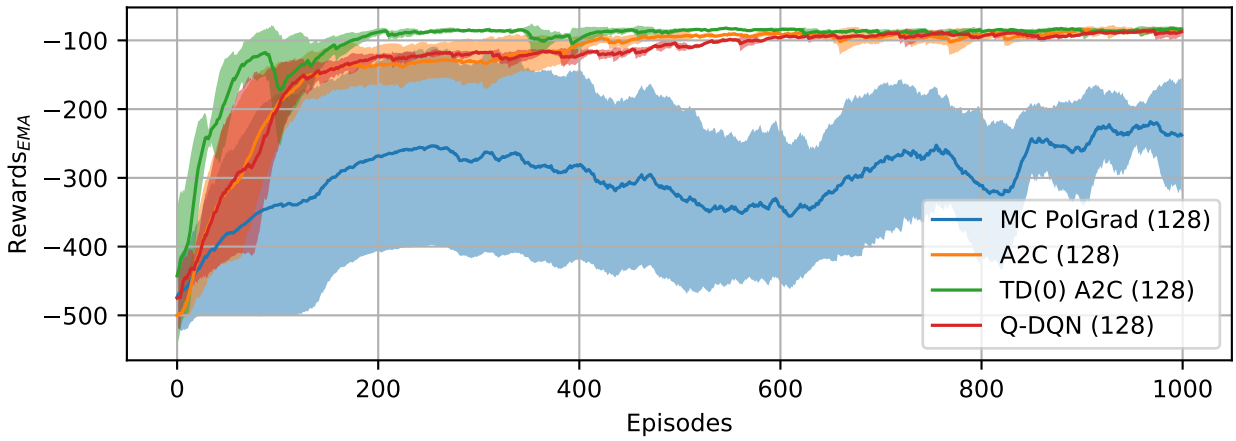


Figure 2: A comparison of agents with a policy network with dropout (a) and without dropout (b) and a DQN agent. Visualized are the mean and standard deviation for all agents based on four training iterations. The number in the brackets refers to the size of the network's hidden layers.

will learn faster due to the higher update rate. This advantage is a drawback at the same time. Bootstrapping leads to a biased target (target depends on weights of the Critic $\hat{V}_\phi^\pi(S)$). A biased target entails a higher variance, so introducing even more variance with a dropout factor to improve robustness of the algorithm had a counteracting effect (Figure 2a).
At last, figure 4 in the appendix visualizes the results of varying the dimension of the NN's hidden layers. Though larger networks could be considered a better approximation of the infinite state space of the environment, they have to be treated with caution and are definitely no guarantee for improvements. On the contrary, only 32 neurons in each hidden layer proved to be very effective, while a hidden layer dimension of 512 neurons already entailed strong variance for the learning progress.

5 Conclusion

As expected, conventional semi-gradient algorithms left room for improvement in the continuous state, discrete action environment of the Acrobot. Implementing a DQN was a game changer and led to a strong and consistent performance surpassing the -100 threshold of the environment easily. The very same DQN was even capable of solving the quite hard MoutainCar-v0 environment (included in attached code). Advanced policy gradient methods like the MC-A2C or TD(0)-A2C algorithms also

proved their point in being great approaches to solve environments like the Acrobot and performed more or less equal to the DQN.
Since a well tuned softmax policy (tuned temperature τ) is often considered a superior solution to ϵ-greedy, it might further improve the semi-gradient methods' learning and best performance (Tijsma et al., 2016). More involved methods like a Value-Difference Based Exploration (Tokic, 2010) could take this even further. Furthermore, there is a middle ground between the one-step TD(0)- and the complete episode MC-updates for the A2C algorithm. The so-called TD(λ)-A2C uses eligibility traces, which can be tuned to effectively balance actual rewards and estimates used to calculate the return.
It became clear that dropout as a regularization technique is generally less helpful within the RL framework, but a weight controlling L1 or L2 regularization could be another powerful technique to improve generalization and performance of the agents.
From a more general perspective, function approximators should become better and better the more parameters they incorporate to estimate the infinite number of states in a continuous environment. So larger and deeper NNs are a good choice as long as the training framework can handle the complications and required resources that come along with it.

References

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Roth, T. (2015). Let's build a dqn: Simple implementation [Accessed: 2021-01-15]. https://tomroth.com.au/dqn-simple/

Tijsma, A., Drugan, M., & Wiering, M. (2016). Comparing exploration strategies for q-learning in random stochastic mazes. https://doi.org/10.1109/SSCI.2016.7849366

Tokic, M. (2010). Adaptive -greedy exploration in reinforcement learning based on value differences, 203–210. https://doi.org/10.1007/978-3-642-16111-7_23

A Figures supporting discussion part

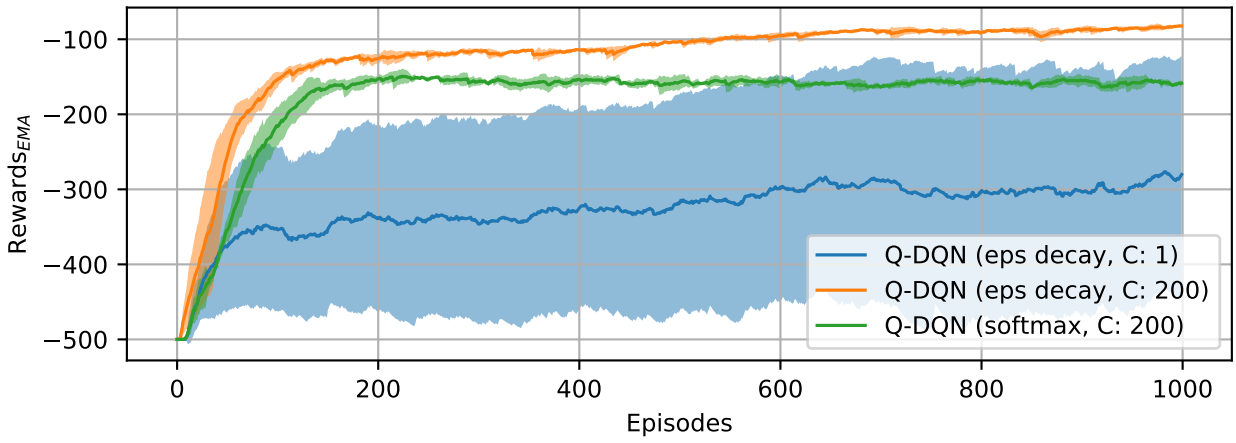


Figure 3: Three different DQN agents. One with softmax action selection and one without a target network (C=1). Visualized are the mean and standard deviation for the agents based on four training iterations.

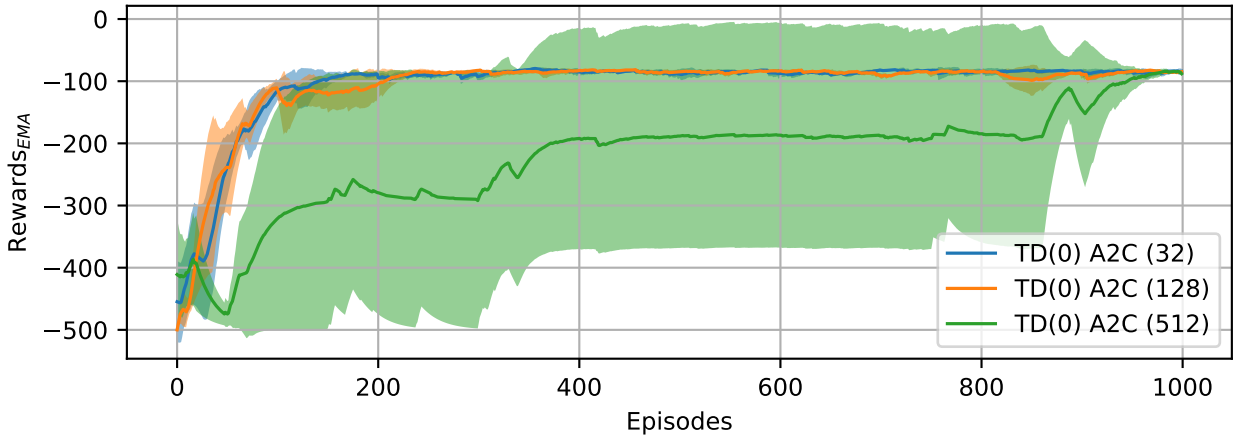


Figure 4: Comparing different sizes of the network's hidden layers for the TD(0)-A2C algorithm without dropout. Visualized are the mean and standard deviation for all agents based on four training iterations.

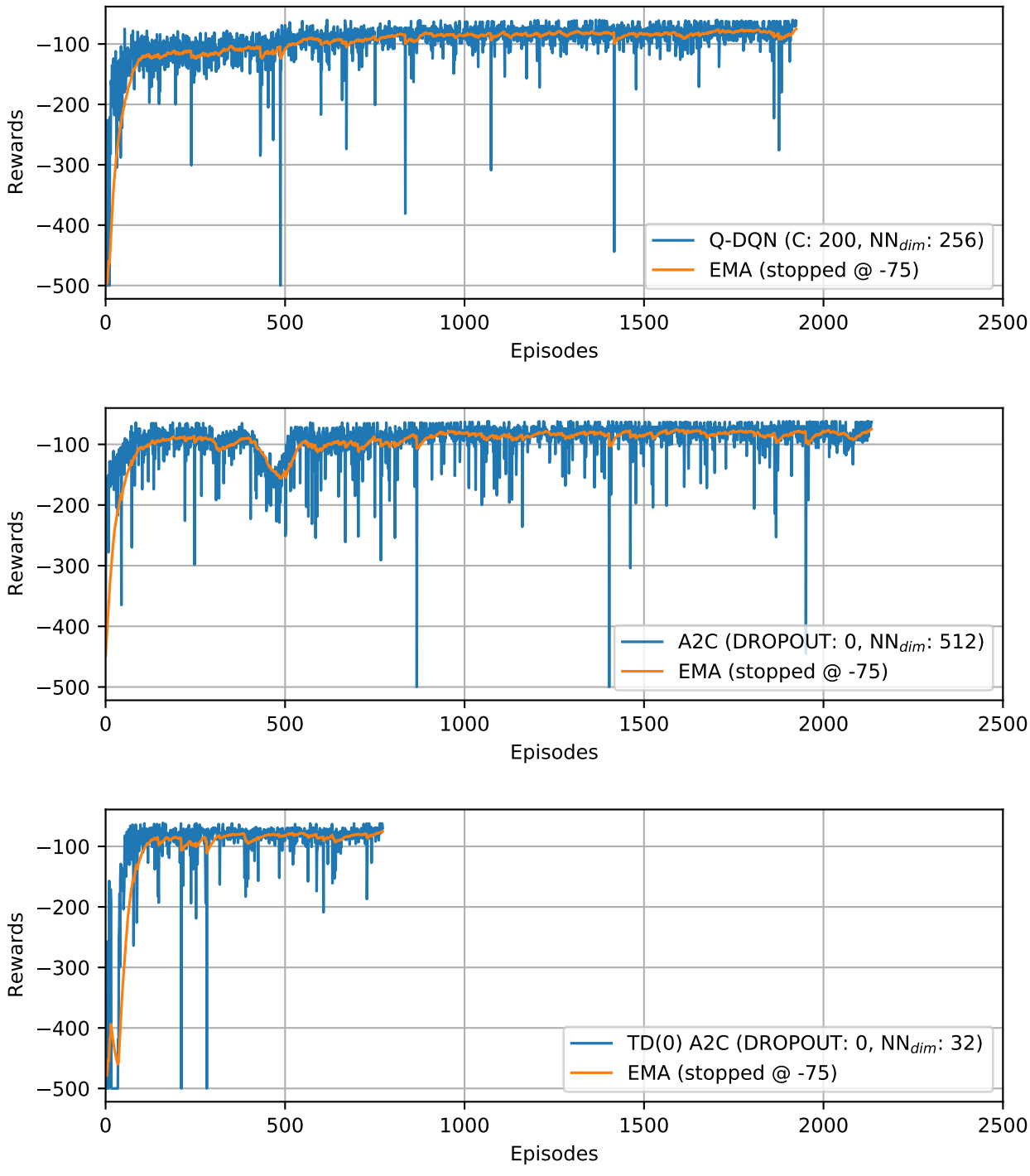


Figure 5: Single agents of the three best algorithms trained until they surpassed a reward of -75 with the exponential moving average (EMA).