

Research on the usage of target networks in Deep Reinforcement Learning (Machine Learning 2022 Course)

Andrey Spiridonov^{1,2} Vladislav Trifonov¹ Yerassyl Balkybek¹ Nikolay Sheyko¹ Dmitrii Gromyko¹

Abstract

This paper is dedicated to studying the popular model-free reinforcement learning algorithm - Q-learning. Models under investigation such as DQN, Double DQN, and Dueling DQN has an auxiliary network within its architecture to increase stability of training - target network. We conducted several experiments, which shows that replacing target network with action network leads to instability of DQN and Double DQN training. Nevertheless, it can be removed in Dueling DQN model without loss in stability and achieved score.

 **Github repo:** [GitHub](#)

 **WandB project summary:** [WandB](#)

 **Video presentation:** [Google drive](#)

1. Introduction

Every living organism that we know receives information about the surrounding environment by the means of sensory input. While vision and hearing do not provide us exact coordinates, sizes, velocities of the surrounding objects, we, nevertheless, possess an amazing ability to successfully operate in the constantly changing world. This ability is not inherited at all, as the phenomenon of learning and mastering is observed in all domains of our life, from walking and predicting trajectories of moving objects to driving a vehicle. At the same time, algorithms of artificial intelligence usually deal with some human-prepared data, thus resembling the human behaviour only partially. Although operation in the domain of distilled data is much more efficient (compare the difficulty of encoding digits in binary system with graphical representation of symbols), ability to learn and adapt though some sensory input potentially leads

¹Skolkovo Institute of Science and Technology, Moscow, Russia ²Higher School of Economics, CS department, Moscow, Russia. Correspondence to: Dmitrii Gromyko <dmitrii.gromyko@skoltech.ru>.

to creation of more complicated and powerful algorithms. A significant attempt to construct such system was proposed in (Mnih et al., 2015). The authors combined a reinforcement learning (RL) agent with a deep convolutional neural network that efficiently learned behaviour policy by analysing high-dimensional sensory input. This agent is named a deep Q-network (DQN) because the algorithm is centered around an optimal action-value function Q . This work received considerable attention and gave birth to further modifications of the agent such as Double DQN (Hasselt et al., 2016) and (Wang et al., 2016) Dueling DQN. The goal of this project is to check the hypothesis on the usage of the target networks: whether the larger sizes of action neural networks in Q-learning algorithms can achieve stable performance without target network. We choose 3 deep RL algorithms, which use target networks, namely, DQN, Double DQN and Dueling DQN. In order to run our experiments we choose several ATARI 2600 games: Breakout and Daemon Attack. The stability of the network is measured by tracking the average episode rewards.

The main contributions of this report are as follows:

1. For DQN and Double DQN models the necessity of target network was proven.
2. Direct correlation between the neural network size and stability for Q-learning models without target network was observed.
3. It was shown, that Dueling DQN model without target network can achieve the same performance as original Dueling DQN.

2. Related work

One of the oldest discussion in reinforcement learning in which considerations of value and advantage functions to solve sequential decision problems can be found in (Baird III, 1993) and was discussed by Sutton and Barto in their book of "A Reinforcement Learning: An Introduction" (Sutton & Barto, 1998). Baird in his original paper (Baird III, 1993) decomposed the shared Bellman residual update equation into a state value and advantage functions. Further descriptions, their properties for advantage functions and their convergence were studied by (Baird & Klopff, 1993; Harmon et al., 1994; Harmon & Baird, 1996). In

addition, the policy gradients in Q-learning (Sutton et al., 1999) as well as their significant variance reductions were evaluated recently by (Schulman et al., 2015).

One of the major hindrance of optimal policy learning and which is also frequently occurring in Q-learning models, overestimation and overoptimism were first observed and theoretically explained by Thrun and Schwartz in their theoretical work (Thrun & Schwartz, 1993). They assert that even the low inaccuracies in action values may lead to overestimation of Q functions, and this in turn will lead to overoptimism of actions. In addition, Thrun and Schwartz could give justified explicit conditions when Q-learning models and reinforcement learning algorithms will definitely fail, and as an example, Watkin's Q-learning (Watkins, 1989) was provided with its detailed analysis. The further theoretical considerations of overestimation of action-value function and overoptimism of action values relevant to DQN-related architectures were completed by (Sutton, 1990; Agrawal, 1995; Geramifard et al., 2013; Auer et al., 2002; Brafman & Tennenholtz, 2002) and etc. The Hasselt et al in their work (Hasselt et al., 2016) relying on Thrun and Schwartz's theoretical developmets (Thrun & Schwartz, 1993), finds out that the in general all the deep learning based DQN architectures suffer from overestimation and overoptimism but their proposed Double DQN architecture endures against this type of difficulties in Q-learning. They also claim that this Double DQM can be further be developed to the large-scale function approximations.

At the advent of deep neural networks, several DQN (Deep Q-Network) related reinforcement learning algorithms were designed, and applied to several Atari games by (Mnih et al., 2015; Nair et al., 2015; Watter et al., 2015; Guo et al., 2014; Bellemare et al., 2016; Schaul et al., 2015). Most of them mainly produces artificial agents that can develop successful policies via training themselves in high dimensional deep neural networks. For example, Mnih et al showed that the deep Q-network agent, only taking as inputs the pixels and the game scores, can outperform almost all the previously existing algorithms and able to reach to the the professional players records in a set of 49 games.

2.1. DQN

DQN is a RL agent that receives information about the state of the environment in a form of the sequence of images x_t and rewards r_t that follow after the actions taken by the algorithm. The set of legal actions $a_t \in A = \{1, \dots, K\}$ that the algorithm can take on each step are defined by the environment in which the algorithm operates. After each action the environment changes as well as the game score that acts as a reward system. Changes in the game score could be caused not only because of the recent action of the agent, but also as a result of some series of actions taken by the agent. That

is why the agent receives adds a new input x_t to the saved series of observations: $s_t = x_1, a_1, \dots, x_t, a_t$. The number of saved observations is limited, thus the agent action can be described as a finite-length Markov decision process. When deciding what action to take the agent tries to maximize the cumulative future reward. Mathematically, this principle is expressed by the means of the optimal action-value function

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t'=t}^T \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, \pi \right] \quad (1)$$

where rewards r_t at a time step t are discounted by a factor γ . The sum of such rewards $R_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ (T denotes the final step of the game) should be maximised according to the optimal behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a). To address possible instabilities of the RL we randomize over the data to eliminate correlations between the observations. We also iteratively update Q towards target values that are periodically updated, as was proposed in (Mnih et al., 2015), thus we reduced correlations with the target.

One can write the so-called Bellman equation for the action-value function. For the known optimal action-value $Q^*(s', a')$ the agent should take such action a' that maximizes the sum of expected reward and the discounted action-value function:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right] \quad (2)$$

The Bellman equation is further transformed into

$$Q_{i+1}(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q_i(s', a') \mid s, a \right] \quad (3)$$

so that the DQN algorithm could update the action-value function iteratively. As the action-value function is calculated independently for each observation series, one should provide a parametric approximation for it.

The main feature of the current algorithm is that the action-value function is approximated with a nonlinear function $Q(s, a; \theta_i) \approx Q^*(s, a)$ which is a convolutional neural network. This approximation depends on parameters θ_i that are weights of the neural network at an iteration t . The experiences of the algorithm $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step t are saved in the data set $D_t = \{e_1, \dots, e_t\}$. We update the action-value functions by taking random batches of experience $(s, a, r, s') \sim U(D)$ from the data-set D . The Q-learning update is then performed as follows:

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a,r} \left[(\mathbb{E}_{s'}[y \mid s, a] - Q(s, a; \theta_i))^2 \right] = \\ &= \mathbb{E}_{s,a,r,s'} \left[(y - Q(s, a; \theta_i))^2 \right] + \mathbb{E}_{s,a,r} [\mathbb{V}_{s'}[y]]. \end{aligned} \quad (4)$$

Here $y = r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$, $L_i(\theta_i)$ is the loss function, θ_i denote the Q-network parameters at the step i ,

while θ_i^- denote the parameters used to compute the target. These parameters are constant between the updates, every C steps an update $\theta_i^- = \theta_i$ happens. This process is called network synchronization.

To solve the optimization problem one should compute the gradient of the loss function 4 (note that the second term is the variance of the targets and is independent on θ_i):

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right], \quad (5)$$

although it is more efficient to apply the stochastic gradient descent approach. In order to add some randomness in the decision making process the ε parameter is introduced. This parameter declines from 1 in the beginning of the training process to some small number (0.1 or 0.01) after some fixed number of iterations. The agent then behaves randomly with probability ε and follows the behaviour policy with probability $1 - \varepsilon$. The whole algorithm is summarized in Alg. 1.

2.2. Double DQN

The authors in (Hasselt et al., 2016) state that any type of small and uniformly distributed errors can accumulate and affect the overestimation of the action value, even if the action value on average is correct. This in turn leads to the asymptotically sub-optimal policies, π , or in other words overoptimistic values in Q . The main contributor of this problem is because the same values are being used in the selection and evaluation of the action. The main idea to deal with this problem is to decouple the max operation into action selection and action evaluation. At the same time two sets of θ weights will be randomly assigned - one is for the action selection, and the other is for the evaluation. This algorithm by (Hasselt et al., 2016) is called Double Q-learning or Double DQN.

In the case of Double-DQN, we replace the target $y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ in (5) with the following term,

$$y_i^{DDQN} = r + \gamma Q\left(s', \max_{a'} Q(s', a'; \theta_i); \theta_i^-\right) \quad (6)$$

and the rest of the algorithms remains the same. The update on target value y helps to mitigate overestimation of Q values, and improves overall performance in several games. Thus, here two value functions will be evaluated, one is to determine the greedy policy and the other is to evaluate its value.

2.3. Duelling DQN

However, the Duelling Network Architecture consists of single Q-network architecture, which utilizes two sequences

or streams of fully connected layers. These two layers give two separate estimations, one is called state value function $V^\pi(s)$, and another one is state-action-dependent advantage function $A^\pi(s, a)$, as described below,

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)} [Q^\pi(s, a)] \quad (7)$$

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (8)$$

at the end these two streams are combined resulting in a single Q -function,

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \quad (9)$$

The latter is slightly improved by subtracting the mean of action function leading to the final configuration,

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$

The main advantages of this expression is, firstly, to provide the possibility to identify the V and A functions, and secondly, to increase the stability of the optimization. The authors (Wang et al., 2016) uses the latter form Q -function in their algorithms and experiments.

In general, the duelling DQN architecture via dividing the whole architecture into two streams, will be able decide if it is important to estimate the value of the action choice, and unnecessary estimations are skipped. For example, in some games, it might be possible that the state is not changed significantly, then the calculation of state value function is needless.

3. Algorithms and Models

3.1. Algorithms

To conduct experiments were chosen three models: vanilla DQN, Double DQN, and Dueling DQN. Theoretically, Duelling DQN and Double DQN are improvements of DQN algorithms but in two different directions. Double DQN replaces maximum operation with another Q-function in order to separate selection and evaluation of an action. This in general improves the quality in a way that overestimation and overoptimistic scenarios can be avoided.

In case of Duelling DQN, the single DQN is seen as two parallel streams of layers, one is responsible for the state value function, and another one is responsible for the action advantage function as the name it. The main idea behind this is that if the action state is not changed over the last steps, then the value of action choice will not be evaluated. For further information one can read corresponding section of this paper.

Algorithm 1 deep Q-learning with experience replay (adopted from (Mnih et al., 2015))

```

Initialize replay memory  $D$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights  $\theta$ 
Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$ 
for episode = 1,  $M$  DO
    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$ 
    for  $t = 1, T$  DO
        With probability  $\varepsilon$  select a random action  $a_t$ 
        otherwise select  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$ 
        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$ 
        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$ 
        Every  $C$  steps reset  $\hat{Q} = Q$ 
    end for
end for
    
```

3.2. Models

The vanilla DQN model faces issues with stability. To overcome them, neural network for target is created. The target network has fixed weights for some steps in order to stabilize the computation of max Q-value. Then after some iterations, the target network gets its weights updated by a copy of weights from the action network and has constant weights for specified number of learning steps. It's implied, that in case of usage DQN models without target network, the total rewards will be very lower and training will be unstable.

The hypothesis is that as the main action network increases, the need for a stabilization target network disappears. Technically, the test of the hypothesis is to replace the target network with a full copy of the Action network with different depths.

For testing described hypothesis we suggest to use following CNN and Dueling CNN architectures:

Vanilla DQN with CNN (also used in Double DQN):

1. SmallCNN: 3 convolutional layers: 1.7M parameters;
2. MediumCNN: 3 convolutional layers: 6.9M parameters;

ters;

3. LargeCNN: 5 convolutional layers: 14.4 parameters.

Dueling DQN:

1. SmallDuelingCNN: 3 convolutional layers: 1.7M parameters;
2. MediumDuelingCNN: 3 convolutional layers: 6.7M parameters;
3. LargeDuelingCNN: 4 convolutional layers: 7.4M parameters.

Hyperparameters of the algorithms are provided in complementary table of Fig.1.

To check the hypothesis a bunch of experiments with modification of Q-learning algorithms was conducted. Used basemodels are implemented in the [PyTorch Lighthing](#). As environments for agents we took two classical Atari games from [OpenAI gym](#): "Breakout" and "Demon Attack".

In total was conducted 26 experiments. Specific numbers for different environments are as follows:

1. Breakout: PyTorch models - 4; modified DQN - 3, modified Double DQN - 3; modified Dueling DQN - 4.
2. Demon Attack: PyTorch models - 3; modified DQN - 3, modified Double DQN - 3; modified Dueling DQN - 3.

Performance of not modified PyTorch Q-learning algorithms (basemodels) on the chosen environments one can see in pic. 1-a,b and pic. 2-a, b. Every model in each case perform comparable in terms of average reward and loss function for a used number of steps. Code to reproduce the results one can find on GitHub repository: [GitHub](#).

There one can see visualizations of the results for conducted experiments: [WandB](#).

4. Experiments and Results

Computational intensity leads to limitations of possible duration of the experiments. After approximately 75 epochs with 1000 batches per epoch, depending of the size of neural network, RAM on the Google Colab is overwhelmed and current experiment is shut down. Nevertheless, used number of steps are enough to achieve valid results.

Excluding target net from Q-learning algorithm is achieved by replacing it with the main neural network, which approximates Q-value. Architecture of used CNN and Dueling



BATCH SIZE	64	NUMBER OF TRAINING CASES OVER WHICH EACH STOCHASTIC GRADIENT IS COMPUTED
SYNCHRONIZATION RATE	1000	TARGET NETWORK UPDATE FREQUENCY (ONLY FOR REFERENCE DQN)
DISCOUNT FACTOR	0.99	DISCOUNT FACTOR γ USED IN Q-LEARNING UPDATES
LEARNING RATE	10^{-4}	LEARNING RATE OF THE ALGORITHM
REPLAY SIZE	10^5	SGD UPDATES ARE SAMPLED FROM THIS NUMBER OF MOST RECENT FRAMES
MIN EPISODE REWARD	0	MINIMUM REWARD ALLOWED IN EACH GAME SESSION
BATCHES PER EPOCH	1000	
ϵ_{start}	1	INITIAL EXPLORATION FACTOR
ϵ_{end}	0.02	FINAL EXPLORATION FACTOR
ϵ DECAY FRAME	$1.5 * 10^5$	NUMBER OF FRAMES TO REACH ϵ_{end}

Figure 1. Agents performances on Atari Breakout game. Average reward (a) and loss (b) PyTorch basemodels. Average reward (c) and loss (d) of DQN model. Average reward (e) and loss (f) of modified Double DQN model. Average reward (g) and loss (h) of modified Dueling DQN model. Model hyperparameters are presented in the complementary table. Hyperparameters were taken to be the same as the default ones in the PyTorch Lightning DQN algorithm. Here we only present the most relevant ones. We have not performed cross-validation due to the computational complexity.

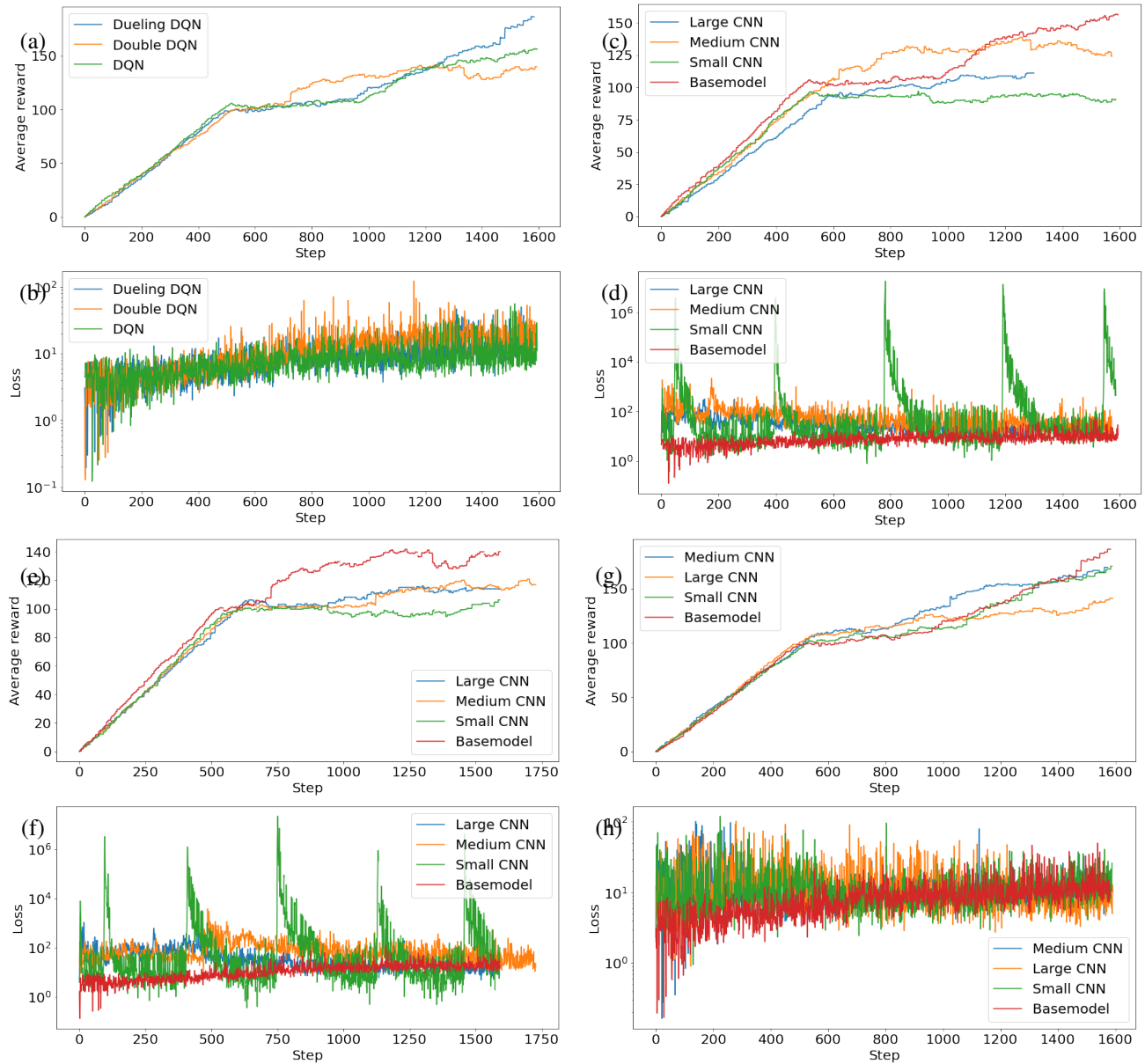


Figure 2. Agents performances on Atari Demon Attack game. Average reward (a) and loss (b) PyTorch basemodels. Average reward (c) and loss (d) of modified DQN model. Average reward (e) and loss (f) of modified Double DQN model. Average reward (g) and loss (h) of modified Dueling DQN model. Model hyperparameters remain the same as for the Breakout environment.

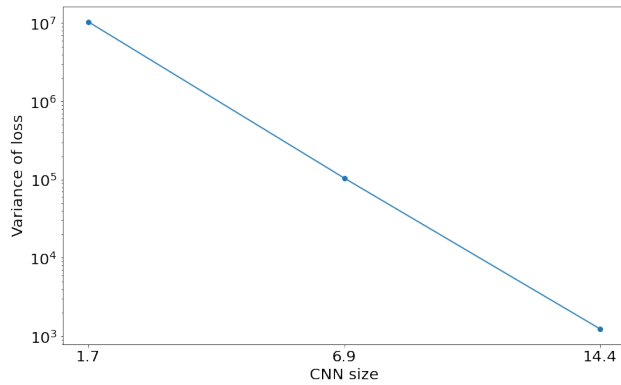


Figure 3. Loss variance dependency on the size of neural network for Modified Double DQN in "Demon Attack" environment. The trend shows that larger networks correspond to more stable algorithm behaviour.

CNN of different sizes one can find in the corresponding section of this paper.

To understand performance of models we track such quantities: train loss (MSE between predicted Q-value and immediate reward with optimal discounted future value), cumulative reward and average reward of the agent.

Conducted experiments with modified DQN algorithm (pic. 1-c,d and pic. 2-c,d) shows, that no model regardless to the size of CNN can't achieve the same score and stability as in the vanilla DQN. Farther, agents in the modified algorithms do not explore the environments and achieve its scores by exploitation starting state.

Modified Double DQN on the game "Demon Attack" showing the same results as for modified DQN (pic. 2-e,f). Meanwhile, the results on modified Double DQN model is slightly different. On pic. 1-e,f one can see that on the game "Break-out" the modified Double DQN algorithm with small CNN achieved similar performance. The growing trend of average reward and stable loss function indicates about it.

All investigated setups show that increasing size of CNN leads to better stability, which was proposed as one of the contribution of this paper. On pic. 3 one can see a huge drop in variance of loss function with growing net size.

Performance of modified Dueling DQN is totally different with previous setups (pic. 1-g,h, pic. 2-g,h). For both environments the agent achieves equivalent value of average reward. Moreover, stability of the loss functions regardless to a size of dueling CNN is also the same.

To justify the results of Dueling DQN we conducted an additional experiment on the machine with higher computation power. On pic. 4 one can see that the results obtained on the lesser amount of training steps are coincide with the results

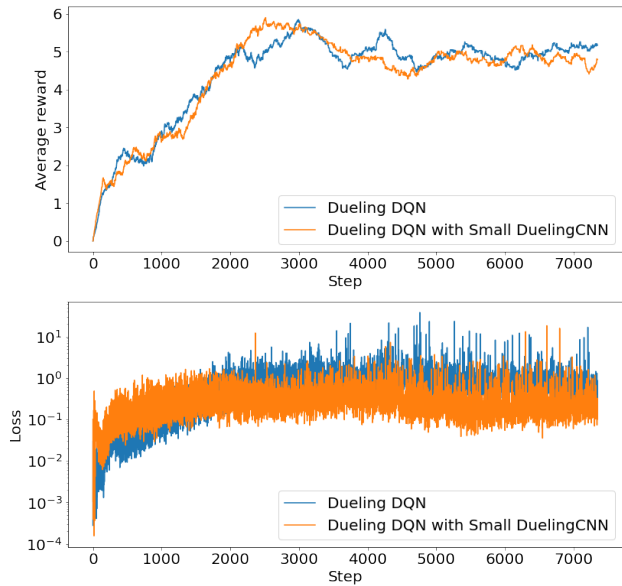


Figure 4. Average reward and loss function vs training for Dueling DQN. Dueling DQN with Small DuelingCNN denotes the modified Dueling DQN model without target network.

of longer training. Performance of the model without target network is stable and the average reward value is equivalent to basemodel. This confirms the lack of need in the target network for Dueling DQN.

5. Conclusion and Prospects

Results of replacing target network with action network in the training process of Q-learning algorithms vary with respect to setup. For each used environment it's proven to be a bad idea for DQN model. The case of Double DQN is not so simple and requires further investigation. For Dueling DQN model was shown, that modified algorithm can perform on the same level as its basemodel. More prolonged training justify the results for Dueling DQN - model without training network perform on the same level as PyTorch basemodel. Also, correlation of depth of neural network and stability of target-network-less model was revealed - large network decrease variance of loss function during training.

Main directions for further research are: to conduct experiments with higher computational powers for more training epochs; to perform fine-tuning of the parameters for chosen environments; to expand zoo of environments. Comprehensive training of Dueling DQN on significant number of environments may lead to reasons for excluding target network from this model.

References

- Agrawal, R. Sample mean based index policies by $o(\log n)$ regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.
- Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.
- Baird, L. C. and Klopff, A. H. Reinforcement learning with high-dimensional continuous actions. *Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147*, 15, 1993.
- Baird III, L. C. Advantage updating. Technical report, WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993.
- Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P., and Munos, R. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
- Brafman, R. I. and Tenenbholz, M. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct): 213–231, 2002.
- Geramifard, A., Walsh, T. J., Tellex, S., Chowdhary, G., Roy, N., and How, J. P. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451, 2013.
- Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. Deep learning for real-time atari game play using offline monte-carlo tree search planning. *Advances in neural information processing systems*, 27, 2014.
- Harmon, M. and Baird, L. Multi-player residual advantage learning with general function approximation (technical report no. wl-tr-1065). *Wright-Patterson Air Force Base Ohio: Wright Laboratory*, 1996.
- Harmon, M. E., Baird, L., and Klopff, A. H. Advantage updating applied to a differential game. *Advances in neural information processing systems*, 7, 1994.
- Hasselt, H. v., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI’16, pp. 2094–2100. AAAI Press, 2016.
- 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., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL <https://doi.org/10.1038/nature14236>.
- Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.
- Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
- Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pp. 216–224. Elsevier, 1990.
- Sutton, R. S. and Barto, A. G. Reinforcement learning: an introduction mit press. *Cambridge, MA*, 22447, 1998.
- Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- Thrun, S. and Schwartz, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the low inaccuracies in action values may lead to overestimation of Q functions, and this 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, volume 6, 1993.
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/wangf16.html>.
- Watkins, C. J. C. H. Learning from delayed rewards. 1989.
- Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28, 2015.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Andrey Spiridonov (21.03% of work)

- Reviewing literature on the topic (3 papers)
- Implemented marginal part of experiment pipeline
- Preparing the GitHub Repo and WandB report.
- Report supervision

Vladislav Trifonov (21.92% of work)

- Minor contribution into code
- Experiments with sizes of the network in Double DQN algorithm
- Presentation
- Preparing the Sections 4 and 5 of this report

Yerassyl Balkybek (19.01% of work)

- Extensive paper reviews
- Experiments with sizes of the network in Dueling DQN algorithm
- Preparing the Section 2 of this report

Nikolay Sheyko (19.01% of work)

- Study of algorithms and models with several convolutional layers
- Experiments with sizes of the network in Double and Dueling DQN algorithms
- Preparing the Section 3 of this report

Dmitrii Gromyko (19.03% of work)

- Reviewing literature on the topic (3 papers)
- Preparing the Sections 1 and 2 of this report
- Experiments with sizes of the network in DQN algorithm
- Presentation

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

☒ Yes.
☐ No.
☐ Not applicable.

General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

Students' comment: We reused implementation of training algorithm for DQN and it's derivatives from popular open-source framework Pytorch-Lightning Bolts, which, in turn bases it's implementations on corresponding paper code. All code for experiments, their analysis as well as thorough logging was developed by us.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

☒ Yes.
☐ No.
☐ Not applicable.

Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

☒ Yes.
☐ No.
☐ Not applicable.

Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

☐ Yes.
☐ No.
☒ Not applicable.

Students' comment: We used standard benchmark-environment from gym which are included into above-mentioned framework. Nothing special was done on our side to collect or prepare any data.

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

☒ Yes.
☐ No.
☐ Not applicable.

Students' comment: It is installed among other Python packages.

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

☐ Yes.
☐ No.
☒ Not applicable.

Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

☐ Yes.
☒ No.
☐ Not applicable.

Students' comment: Training and testing environment runs are controlled by framework, however we did set up all necessary parameters for adequate quality assessment and those are, to some degree, mentioned in the report.

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

☒ Yes.
☐ No.
☐ Not applicable.

Students' comment: Different methods were considered, but most of their hyper-parameters are universal and selection of best of those were out of scope of our work. Conclusion about probed method is in the report.

9. The exact number of evaluation runs is included.

☐ Yes.
☒ No.
☐ Not applicable.

Students' comment: We don't provide these numbers in report because we believe they are irrelevant. However these and similar quantities are easy to check on our [experiment summary page](#) mentioned above.

10. A description of how experiments have been conducted is included.

☒ Yes.

- ☐ No.
- ☐ Not applicable.

Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

- ☒ Yes.
- ☐ No.
- ☐ Not applicable.

Students' comment: Variance of training loss and difference in achieved average reward.

12. Clearly defined error bars are included in the report.

- ☐ Yes.
- ☒ No.
- ☐ Not applicable.

Students' comment: There were not enough resources to conduct experiments with identical setup but different random generator configurations to collect statistics sufficient for producing error bars. However for some setups we were able to conduct few experiments. Thus, some confidence intervals for those are displayed at our [experiment summary page](#) mentioned above.

13. A description of the computing infrastructure used is included in the report.

- ☒ Yes.
- ☐ No.
- ☐ Not applicable.

Students' comment: None