INSTITUT FÜR INFORMATIK
Machine Learning
Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Actor-Critic Reinforcement Learning With Experience Replay

Julian Robert Ullrich

Bachelorarbeit

Beginn der Arbeit:	25. Juli 2018
Abgabe der Arbeit:	25. Oktober 2018
Gutachter:	Univ.-Prof. Dr. S. Harmeling
	Univ.-Prof. Dr. M. Leuschel
Betreuer:	Julius Ramakers

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 25. Oktober 2018

Julian Robert Ullrich

Abstract

Deep Reinforcement Learning and policy gradient methods majorly contributed to the most recent advances in the field of Artificial Intelligence. These Methods enabled machines to surpass human performance for Atari console games (Mnih et al., 2013), boardgames like Chess, Shogi (Silver et al., 2017a) or Go (Silver et al., 2017b) and most recently even complex team-based computer games (OpenAI, 2018).

As environments get more complex the cost of simulating the environment increases and often outweights the isolated computational cost of training the agent, making sample efficient methods nessecary.

This thesis will take a look at off-policy methods and learning from previously sampled data, with the main focus being the implementation and evaluation of the "Actor-Critic with Experience Replay"(ACER) algorithm proposed by Wang et al., 2016 on the Atari 2600 console games.

Contents

1 Introduction

Sutton and Barto (2018) describes the reinforcement learning task as "learning what to do". Acting optimal within an unknown environment can be very difficult. The field within machine learning adressing this problem is called reinforcement learning.

The reinforcement problem consist of an *agent* taking *actions* within some sort of *environment*. By interacting with the *environment* the *agent* receives *rewards*.

The goal of reinforcement learning is to create fast and reliable learning algorithms for the *agent* to gain the maximum *reward*

Environments can range from simple tasks, like balancing a pole to very complex and demanding tasks where *environment states* are given as pixels, continuous control problems or real life robotic tasks. This thesis will work with the Atari 2600 environments offered by OpenAI (Brockman et al., 2016)

By combining deep learning techniques (Hinton and Salakhutdinov, 2006) with reinforcemen learning, the problems posed by most of the Atari games can easily be solved.

However complex environment like the Atari 2600 games can often be costly to simulate.

ACER (Wang et al., 2016) provides a sample efficient learning agent. This work aims at implementing and evaluating the Algorithm

This thesis will provide a short overview for important concepts of reinforcement learning

To lay out the foundation for ACER, policy gradient, specifically Actor-Critic methods and the Advantage Actor Critic(A3C) - Algorithm (Mnih et al., 2016) are looked upon, followed by an introduction to some approaches of Off-Policy learning.

Finally the ACER- Algorithm is presented, implemented and evaluated.

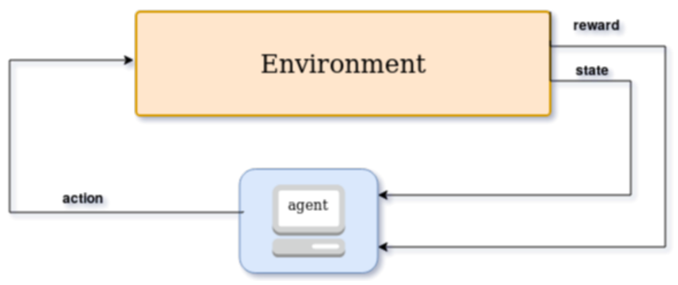


Figure 1: Agent interacting with the environment

2 Reinforcement Learning Framework

The core opponent of the reinforcement learning framework are the *agent* and the *environment*.

An agent interacts with the *environment* over time, by taking in the environment state, evaluating the state and deciding on an *action*. A core question is the one of exploration and exploitation. In order to learn the best behaviour the *agent* needs to make sure to explore the *environment* to avoid getting stuck on local maxima, however at some point the gained knowledge should be used to achieve the best possible reward.

Whenever the agent interact with the environment, a reward and a new state are given to him in return.

2.1 Elements of Reinforcement Learning

Sutton and Barto (2018) names 4 core elements of the reinforcement learning framework.

Policy

The behaviour of the agent within at any given time is determined by the *policy*. A policy can roughly be described as a mapping of states to an action or a distribution over actions.

Reward Signal

The problem posed by an environment is defined through the reward function. The goal of the learning agent is to receive the maximum accumulated future reward at any given time. One of the most important features of reinfocement learning is the fact, that rewards are often very delayed. Good opening moves in for example Chess will play a major role in winning the game, which however usually occurs at a much later stage.

Value Function

The value of a state (or a state - action pair) describes how much more reward can be earned from this state onwards. Values represent the sum of the future rewards, and indicate the long term desirability of states.

Environment Model

In order to solve a problem, a model of the environment can be learned and used for planning. A model can be used to predict future states and rewards before they happen.

Model-based and model-free reinforcement learning methods, which explicitly learn by trial and error both play an important role in reinforcement learning.

2.2 Markov Decision Process

The sequential decision making process of the *agent* can be more formally described as a Markov decision process (MDP).

The sequential decision making process is given by a sequence of states, actions and rewards:

$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \dots, S_t, A_t, R_t, S_{t+1}$

Within this thesis a finite environment is assumed. We call a state *Markov* or say it has *Markov property* if it only depends on it's predecessor rather than the whole history.

$Pr(s_{t+1} = s', r_1 = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r1, a0, s0) = Pr(s_{t+1} = s', r_1 = r \mid s_t, a_t)$

A finite discounted Markov decision process $MDP(S, A, P_a, R_a, \gamma)$ contains a finite set of states S, a finite set of Actions A, the transition probablity to end up in state s' if action a is taken in state $s : P^a_{ss'} = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$, the reward function $R^a_{ss'}$, and the discount factor $\gamma \in [0, 1)$, used to define the importance of immediate reward in contrast to future reward.

2.3 Deep Reinforcement Learning

In contrast to simply mapping an input to an output value, deep learning algorithms contain so called *hidden layers*. For each layer the the weighted sum of units from the previous layer is computed as input. Usually a transformation or activation function is used on the input. Rectified linear units (ReLU), tanh or the sigmoid function can be named as popular activaton functions. After feeding an input into the network and calculation an error value, the weights are ajusted through backpropagation.

Convolutional neural networks (CNN) were inspired by visual neuroscience and are great tools to process image data. CNNs usually contain convolutional layers, pooling layers and fully connected layers. Other relevant methods are recurrent neural networks (RNN) or long short term memory networks (LSTM).

Combining deep learning methods with reinforcement learning methods was a major breakthrough, enabeling reinforcement learning methods to be successfully applied to complex problems like those posed by the Atari 2600 console. (Li, 2017)

3 Actor-Critic Methods

Many different approaches to different kind of reinforcement learning problems exist. Dynamic programming methods can compute optimal policies, however a perfect model of the environment as MDP is required. Monte-Carlo methods on the other hand can estimate value functions and discover optimal policies by averaging over sampled trajectories.

3.1 TD-Learning

Sutton and Barto, 2018 describes *temporal difference* (TD) learning as one of the central ideas in reinforcement learning. TD learning combines dynamic programming with Monte-Carlo methods to learn eather *state-values: V(s)* or *state-action values: Q(s, a)* which are given as:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \tag{1}$$

and

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \tag{2}$$

respectively, where E_{π} denotes the expected *return*, which is the accumulated discounted reward starting from a state or state-action pair, following the policy π

This is a good place to define the *advantage* which is given by

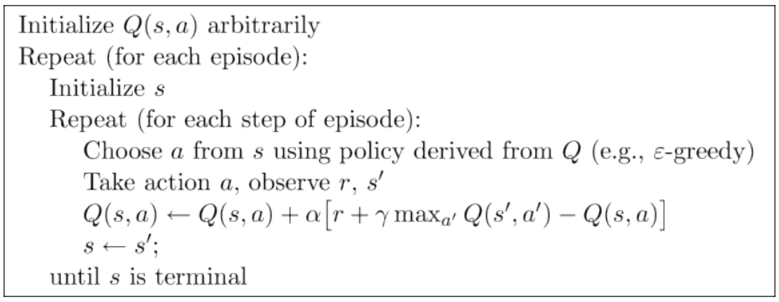
$$A(s,a) = Q(s,a) - V(s,a) \tag{3}$$

and denotes how much better an action is, compared to the average action.

One of the most important TD-learning methods is Q-Learning. It is used to approximate the state-action value function. The update step is given as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+a}, a) - Q(s_t, a_t)] \tag{4}$$

The agent can then direcly choose the action which is expected to give the best total reward. The ϵ-greedy policy is usually applied, which uses eather the best estimated action or a random policy to decide the next action, depending on the ϵ value, which is gradually lowered over the learning period, to reach optimal behaviour while ensuring sufficient exploration. Following algorithm is given by Sutton and Barto (2018)



```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$;
    until $s$ is terminal
```

Q-learning algorithm taken from (Sutton and Barto, 2018)

3.2 Critic-Only Methods

The shown Q-learning algorith or SARSA are popular critic-only methods.

Critic-only methods learn state-action values. They do not contain an explicit function for the policy, but rather derive it from the learned state-action values by acting greedy on the Q-Values.

By only using a critic a low variance estimate of the expected returns is achieved, however the methods suffer from being biased and can be problematic in terms of convergence.

3.3 Actor-Only Methods

Onlike critic-only methods, actor only methods do not learn any state or state-action values. Instead they perform optimization directly on the policy. Usually a stochastic and parameterized policy π_θ is used.

Policy gradient methods like REINFORCE change the policy in order to maximize the average reward at a given timestep by performing a gradient ascent step. (Williams, 1992) Given a performance (average reward per timestep) J and a policy which is parameterized by θ we can denote the gradient as

$$\Delta\theta = \alpha \frac{\partial J}{\partial \theta}$$

where α denotes the step size.

Algorithm 1: REINFORCE provided by Sutton and Barto (2018)

```
Initialize parameters $\theta \, for \pi$
Repeat forever:
    Generate episode: S$_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, following\ \pi$
    For each step t =0,1,... in T:
        G $\leftarrow$return from step $t$
        $\theta \leftarrow \theta + \alpha\gamma^t G \nabla_\theta ln(A_t \mid S_t, \theta)$
```

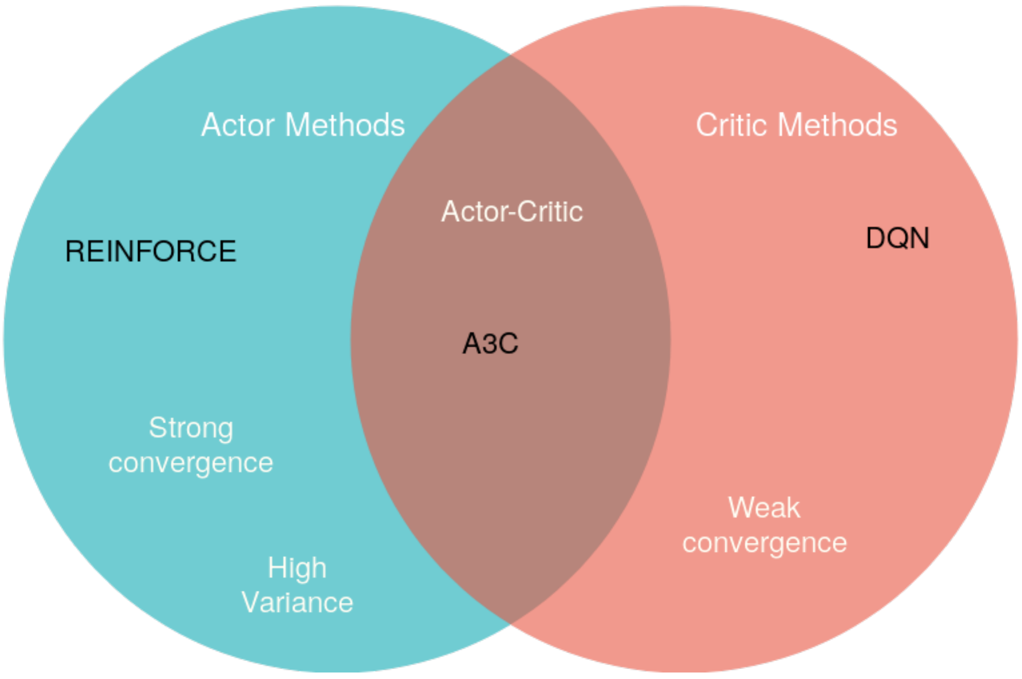


Figure 2: Actor, Critic, and Actor-Critic approach

In contrast to value based approaches, policy gradient methods provide strong convergence to at least a local maximum. On top of that, actor methods are applicable on continuus action spaces. (Sutton et al., 2000)

Actor-only methods however suffer from a large variance of the gradient. Compared to critic-only methods their learning process is significantly slowed down. (Grondman et al., 2012)

3.4 Actor-Critic Methods

Actor-critic methods tackle the problem of high variance in policy gradient methods with the use of a critic.

They combine the strenght of both approaches to achieve a learning agent, which has strong convergence, yet low variance.

Since actor-critic methods are still policy gradient methods at their core, they provide the possibility to work on continuous actions spaces just like the actor-only approach.

3.5 Asynchronous Advantage Actor Critic (A3C)

In general we call an algorithm on policy, if the data used in the policy update was sampled under the same policy. The sequence of observed data encountered by an RL agent is strongly correlated and non-stationary (Mnih et al., 2016). This can have a negative influence on the learning process.

Previous methods usually approached this problem by using randomly selected samples from a replay memory (Mnih et al., 2013)

Training an agent comes with a high demand for computational power. To achieve feasible training times, former algorithms heavily relied on a strong GPU.

The asynchronous advantage actor critic(A3C) algorithm solves both problems, by training simultaneously on multiple environment. Each learner samples trajectories and computes gradients. Those gradients are then applied to the shared parameters. After each global update step, the local parameters are synchronized. This method enables efficent CPU computation, rather than using a GPU. The core idea is, that each agents environement is in a very different state, thus reducing the correlation of the samples.

Algorithm 1: A3C algorithm by Mnih et al., (2013)

```
// Assume shared Parameters and counter $\theta^{global}$, $\theta_v^{global}$, $T = 0$
// with local conterparts $\theta$, $\theta_v$
Initialize parameters for policy $\theta$ and critic $\theta_v$ and counter t=1
repeat
    Reset gradients $d\theta, d\theta_v$
    Synchronize $\theta, \theta_v$ with $\theta^{global}, \theta_v^{global}$
    t$_{start} = t$
    get state s$_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t \mid s_t; \theta)$
        Receive reward $r_t$ and next state s$_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until $s_t$ is terminal or $(t - t_{start}) == t_{max}$;
    R = $\begin{cases} 0 & s_t \text{ terminal} \\ V_\theta(s_t) & s_t \text{ not terminal} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt. $\theta : d\theta^{global} \leftarrow d\theta^{global} + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_{\theta_v}(s_i))$
        Accumulate gradients wrt. $\theta_v : d\theta_v^{global} \leftarrow d\theta_v^{global} + \partial(R - V_{\theta_v}(s_i))^2/\partial\theta_v$
    Perform asynchronous update of $\theta^{global}$ and $\theta_v^{global}$ using d$\theta$, d$\theta_v$
until $T > T_{max}$;
```

4 Off-Policy Learning

Off-policy methods use data sampled from a so called *behavior policy* we will denote as $\mu(a \mid s)$ to optimize the agents *current/target policy* π.

One benefit of off-policy learning is the possibility to chose a more exploratory behaviour policy. Another benefit is the possibility to increase sample efficiency by reusing old data. (Degris et al., 2012)

Off-policy methods can have the drawback of being divergent. The class of Asynchronous Methods A3C belongs to is always slightly off-policy is only slightly off-policy, which doesn't impact the convergence.

If the behaviour policy can be very different from the target policy, the algorithm can no longer be viewed as *safe*. (Munos et al., 2016)

This problem is adressed by many different Methods, in order to ensure convergence, even for arbitraty "off-policyness" of sampled data.

4.1 Importance Sampling (IS)

One of the most basic ideas is to correct for the "off-policyness" by using Importance sampling. It is a classic technique for estimating the value of a random variable x with distribution d if the samples were drawn from another distribution d' By using the product of the likelihood ratios

$$p_t = \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)} \tag{5}$$

Even though this method can guarantee convergence (Munos et al., 2016), it comes with the risk of high, possible infinite variance, due to the variance of the product of importance weights.

4.2 Tree-backup, TB(λ)

The tree-backup method allows off-policy corrections, without the use of imporance sampling by using the expectation and values under the target policy π of every untaken action. Precup et al. (2000)

The algorithm provides low variance off-policy learning with strong convergence. However if a sample is drawn from a policy which is close to the target policy, the algorithm unnessecarly cuts the traces. Without using the full returns, the learning process is slowed down.

4.3 Retrace(λ)

Munos et al. (2016) introduced the Retrace(λ) algorithm. By combining ideas of importance sampling and tree-backup, low varience with strong convergence was achieved while keeping the benefits of full returns.

Like TB(λ) the traces are safely cut in case off strong "off-policyness", without impacting the update too much, if the data was sampled under a behaviour policy μ close to the target policy π.

Retrace values for a q funcion are obtained recursively by

$$Q^{ret}(x_t, a_t) = r_t + \gamma\tilde{p}^{t+1[Q^{ret}(x_{t+1},a_{t+a}-Q(x_{t+a},a_{t+1})]+(x_{t+1})} \tag{6}$$

with $\tilde{p} = min\{c, p_t\}$ being the truncated importance weight p_t (5).

In case of a terminal state, the retrace value is equal to the final reward.

As $\lambda = 1$ performs the best for the Atari console games (Munos et al., 2016), other values were not considered within this Thesis.

5 Actor-Critic with Experience Replay (ACER)

"Actor critic with experience replay" (ACER) introduced by Wang et al. (2016) was one of the first approaches to create a sample efficient, yet stable actor critic method, that applies to both contiuous and discrete action spaces.

ACER combines recent breakthroughs in the field of RL, by utilizing both the ressource efficient parallel training of RL agents proposed by Mnih et al. (2016) and the Retrace algorithm (Munos et al., 2016).

These approaches were combined with truncated importance sampling with bias correction and an efficient trust region policy optimization. For continuous action spaces, stochastic dueling network architectures were used.

Acer can be viewed as an off-policy extension of A3C (Mnih et al., 2016).

The importance weighted policy gradient is given by:

$$\hat{g}^{imp} = \left(\prod_{t=0}^{k} p_t\right) \sum_{t=0}^{k} \left(\sum_{i=0}^{k} \gamma^i r_{t+i}\right) \nabla_\theta \, log \; \pi(a_t \mid s_t) \tag{7}$$

The unbounded importance weights can cause massive variance. Degris et al. (2012) approached this problem by approximating the policy gradient as

$$g^{marg} = E_{s_t\text{\char`~}\beta, a_t\text{\char`~}\mu} \left[p_t \nabla_\theta log \pi_\theta(a_t \mid s_t) Q^\pi(s_t, a_t) \right] \tag{8}$$

where β denotes the limiting distribution and behaviour policy μ

To compute this gradient, knowledge about Q^π is nessecary. The retrace values (6) present a good estimation for Q^π.

To further reduce the variance, the importance weights are truncated. A core problem of actor-critic methods is the tradeoff between bias and variance. By truncating the importance weights, bias is introduced. To counter this, ACER uses a bias correction term.

The final ACER policy gradient is given as:

$$\begin{gathered} g_t^{ACER} = \tilde{p}_t \nabla_\theta log \pi_\theta(a_t \mid s_t) \left[Q^{ret}(s_t, a_t) - V_{\theta v}(s_t) \right] \\ + E_{a\text{\char`~}\pi} \left(\left[\frac{p_t(a) - c}{p_t(a)} \right]_+ \nabla_\theta log \pi_\theta(a \mid s_t) \left[Q_{\theta v}(s_t, a) - V_{\theta v}(s_t) \right] \right) \end{gathered} \tag{9}$$

The critic $Q_{\theta v}$ is trained by minimizing the mean squared error with the retrace values as the target.

Wang et al. (2016) proposed an efficient trust region policy optimization (TRPO), which limits the update step in a way, that the new policy doesnt deviate too much from an average policy.

Even though the improvement through the TRPO method on the continuous action space environments was significant, the results presented in the paper only showed a marginal improvement for discrete action space environments. Because all experiments in this thesis are on discrete action space in order to run more experiments the TRPO was not used.

5.1 Experimental Setup

5.1.1 Environment

All experiments were made using multiple selected games from the Atari 2600 console game environments providede by Brockman et al. (2016).

In our experiments the following OpenAI-Gym environments were used:

Breakout is a game that rapidly speeds up, making it easy for machines to show "superhuman" performance.

Seaquest poses a serious problem for learning agent, as they tend to get stuck on local maxima. It is especially interesting, as it provides multiple possibilities to achieve rewards, and the loss of the game if the player runs out of oxygen is a game mechanic a human can grasp in seconds, while it is notoriously hard for reinforcement learning agents to 'understand'.

Scoring well on this environment can be considered a great achievement.

Space Invaders is a fast paced game. Human and reinforcement learning approaches usually have similar scores. (Mnih et al., 2013)

The hyperparameters were adjusted to ensure a good performance for the Breakout environment. Test on other environments were all performed with the same set of hyperparameters.

OpenAI gym environments provide the agent with a 210x160 pixel colored image. An *environment step* consist of frame skipping, which means, that the action fed into the environment is repeated for k frames. Where k denotes a random number within $\{2, 3, 4\}$.

5.1.2 Preprocessing

Using full scale colored images would come with huge computational cost. To be able to work with the data, each frame is preprocessed. After grayscaling and transforming the frame, the scoreboard area is chopped off. With this method a 84x84 grayscaled pixel image is obtained.

Mnih et al. (2015) proposed to stack multiple frames. More specifically the state contains the last 4 frames received and therefore has a shape of 84x84x4.

To receive the initial state, we simply stack the first frame 4 times.

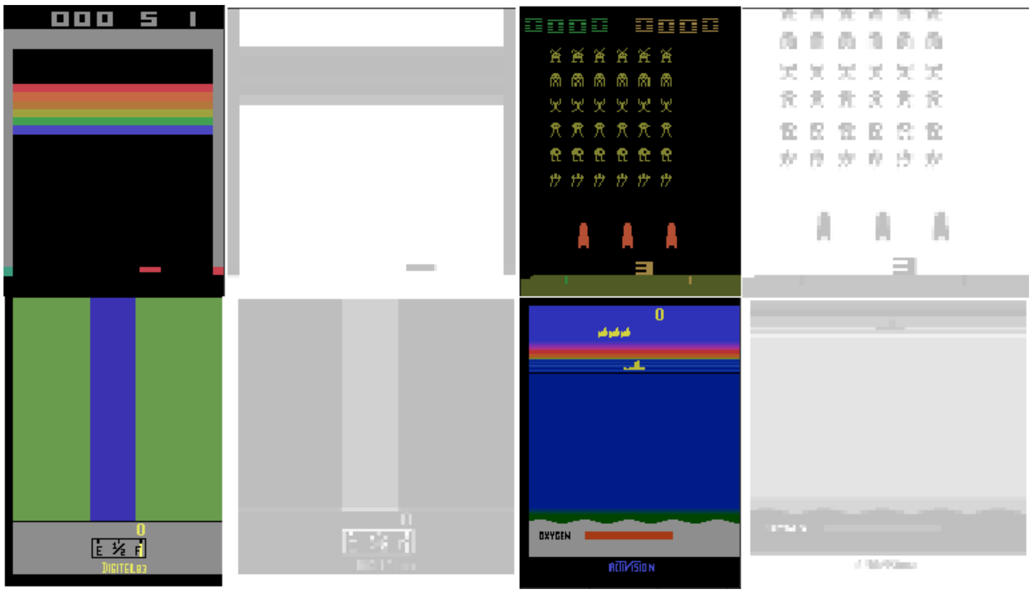


Figure 3: The Atari 2600 console games Breakout, SpaceInvaders, Riverraid and Seaquest before and after being processed.

5.1.3 network architecture

Two architectures were tested. (Mnih et al., 2013) (Mnih et al., 2015).

Both provided good results. Since the paper uses the architecture proposed in the nature paper (Mnih et al., 2015), all further experiments were done using this architecture.

Similar to the (Mnih et al., 2016) network, most of the parameters are shared and used for both estimating Q^{π} and to output a policy.

The shared net consists of 3 convolutional layers. First 32 8x8 filters are applied with a stride of 4. The 2nd layer consists of 64 4x4 filters using a stride of 2. The last convolutional layer uses 64 3x3 filters with stride 1. Finally a fully connected layer of size 512 is applied.

Each of the mentioned layers is followed by a rectifier nonlinearity (ReLU) function, ReLU is used to set all negative outputs to 0, ensuring only positive values are passed to the next layer. Even though other activation/transformation functions exist, the ReLU has empirically done really well.

A test run using tanh instead of ReLU caused a strong loss of performance.

The final shared layers is then mapped to the action space twice to obtain the action scores, and the Q-values. We use a softmax policy to receive the distribution over actions. The action is randomly sampled from the softmax probabilities.

$$P(A = a'|s) = \frac{e^{z_{a'}}}{\sum_{i=0}^{N} e^{z_{a_i}}} \tag{10}$$

where z_a denotes the score assigned to the action through the last network layer.

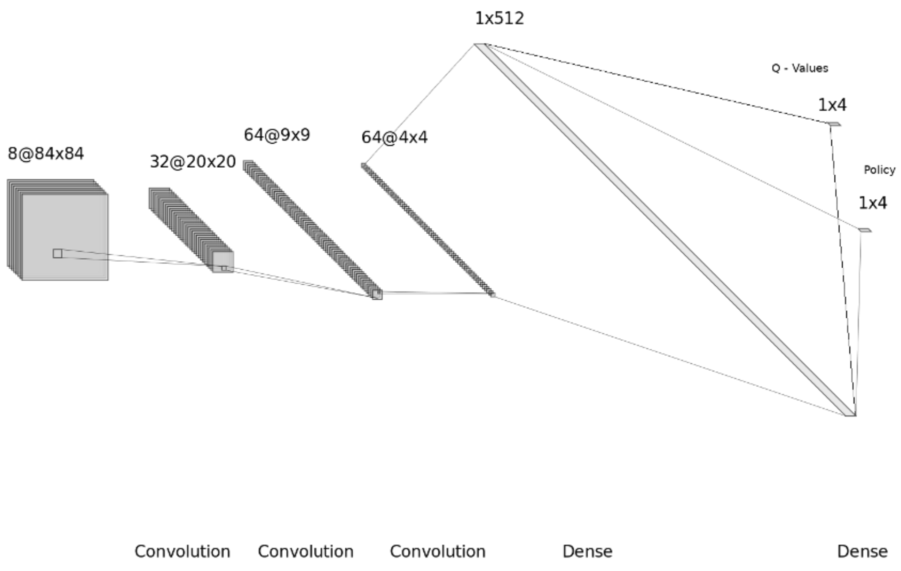


Figure 4: Shared network for policy and Q-Value approximization for an environment with 4 possible actions

The agent is trained by using 16 learner threads running on the CPU. Each Thread has it's own replay buffer, which ensures a more balanced replay and reduces the risk of a single trajectory being used unreasonably often compared to a single shared replay buffer.

Updates to the network were performed every 20 steps or if the environment reached a terminal state.

Replay buffers were designed to hold up to 2500 trajectories, therefore 50.000 frames were saved for each thread and ~ 800.000 frames were stored in total, which is the same size used by DQN (Mnih et al., 2015)

If no replay is used, ACER essentially becomes a version of A3C which uses retrace and Q-values, rather than learning the value function. (Mnih et al., 2016)

For the offline setting, replay ratios of 1,2 and 4 were considered, where a ratio of 4 would mean, that the net is trained on trajectories sampled from the replay buffer 4 times for every online update.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). “OpenAI Gym”. In: *CoRR* abs/1606.01540.

Thomas Degris, Martha White, and Richard S. Sutton (2012). “Off-Policy Actor-Critic”. In: *CoRR* abs/1205.4839.

Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska (2012). “A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients”. In: *Trans. Sys. Man Cyber Part C* 42.6, pp. 1291–1307.

G. E. Hinton and R. R. Salakhutdinov (2006). “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786, pp. 504–507. eprint: http://science.sciencemag.org/content/313/5786/504.full.pdf.

Yuxi Li (2017). “Deep Reinforcement Learning: An Overview”. In: *CoRR* abs/1701.07274.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller (2013). “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602. arXiv: 1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540, pp. 529–533.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare (2016). “Safe and Efficient Off-Policy Reinforcement Learning”. In: *CoRR* abs/1606.02647.

OpenAI (2018). *OpenAI Five*.

Doina Precup, Richard S. Sutton, and Satinder P. Singh (2000). “Eligibility Traces for Off-Policy Policy Evaluation”. In: ICML ’00, pp. 759–766.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis (2017a). “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *CoRR* abs/1712.01815. arXiv: 1712.01815.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis (2017b). “Mastering the game of Go without human knowledge”. In: *Nature* 550. Article.

Richard S. Sutton and Andrew G. Barto (2018). *Introduction to Reinforcement Learning*. 2nd. MIT Press.

Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour (2000). “Policy gradient methods for reinforcement learning with function approximation”. In: *In Advances in Neural Information Processing Systems 12*. MIT Press, pp. 1057–1063.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas (2016). "Sample Efficient Actor-Critic with Experience Replay". In: *CoRR* abs/1611.01224. arXiv: `1611.01224`.

Ronald J. Williams (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3, pp. 229–256.

List of Figures

List of Tables