Democracy in Deep Reinforcement Learning

Semester Thesis

Konstantin Donhauser
donhausk@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Oliver Richter, Ginno Brunner
Prof. Dr. Roger Wattenhofer

January 28, 2019
I want to thank Oliver Richter and Ginno Brunner for the great weekly meetings and for sharing helpful thoughts. Further, I want to thank professor Wattenhofer for providing the resources needed during this report.
This report analyzes ensemble methods in deep reinforcement learning using a highly optimized architecture, Rainbow [1], as a basis for the ensemble. Two algorithms, Basic Ensemble and Ensemble with Retrace, are introduced. Based on tests on the Atari 2600 domain the advantages and disadvantages of these implementations are shown. The report comes to the conclusion that in case of environments with higher complexity the proposed algorithms do not show comparable performance against Rainbow. This is not the case for simpler environments where the proposed algorithms show competitive performance.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Previous Work</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Background</td>
<td>2</td>
</tr>
<tr>
<td>2.2 DQN-Learning</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Extensions</td>
<td>3</td>
</tr>
<tr>
<td>2.3.1 Double and Duelling DQN</td>
<td>3</td>
</tr>
<tr>
<td>2.3.2 Prioritized Experience Replay</td>
<td>4</td>
</tr>
<tr>
<td>2.3.3 Multi Step Bellman Update</td>
<td>4</td>
</tr>
<tr>
<td>2.3.4 Distributional Reinforcement Learning</td>
<td>4</td>
</tr>
<tr>
<td>2.3.5 Noisy Networks for Exploration</td>
<td>5</td>
</tr>
<tr>
<td>2.3.6 Rainbow</td>
<td>5</td>
</tr>
<tr>
<td>2.3.7 Parallelized Reinforcement Learning</td>
<td>5</td>
</tr>
<tr>
<td>2.3.8 Off Policy Correction</td>
<td>6</td>
</tr>
<tr>
<td>2.3.9 Ensemble Methods</td>
<td>7</td>
</tr>
<tr>
<td>3 Analysis of Literature</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Open Research Questions</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1 Ensemble Methods and Rainbow</td>
<td>9</td>
</tr>
<tr>
<td>3.1.2 Retrace and Rainbow</td>
<td>10</td>
</tr>
<tr>
<td>3.1.3 Shared Target Networks for Ensembles</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Approach to Research Questions</td>
<td>11</td>
</tr>
</tbody>
</table>
4 Experiments

4.1 Algorithms ................................................. 12
  4.1.1 Basic Ensemble ................................. 12
  4.1.2 Rainbow with Retrace ......................... 13

4.2 Experimental Setup .............................. 14

4.3 Results and Analysis ............................. 15

5 Conclusion ............................................. 18

Bibliography ........................................... 19

A Prototypes ............................................. A-1

  A.1 Naive Implementation of Ensemble with Rainbow  ... A-1
  A.2 Ensemble with Retrace ............................. A-2
  A.3 Analysis of Retrace Alternatives ................... A-3
    A.4.1 Favor own Trajectories ....................... A-5
    A.4.2 Use common Initialization ..................... A-6
    A.4.3 Shared Target Network ......................... A-6
Chapter 1

Introduction

The phenomena of birds traveling is definitely something impressive, especially when considering the enormous orientation challenges these birds face. A recent study [2] uses cumulative cultural evolution as an explanation, a process allowing groups to develop increasingly complex knowledge and skills over time, beyond the capacities of a single individual.”[2]. This process is a typical example of collective intelligence. The basic concept behind this process, which can be summarized as groups performing better than each individual by itself, has also found approval in machine learning where it is generally referred to as ensemble learning.

In the field of deep reinforcement learning, the DQN architecture introduced by Minh et al is described as “the first artificial agent that is capable of learning to excel at a diverse array of capable tasks”[3]. The algorithm is able to beat human level performance on a variety of Atari 2600 games solely by learning from pixel inputs generated by interactions with its environment. The creation of ensembles is one approach to enhance DQN and has resulted in notable publications [4, 5]. These paper have shown that the powerful concept of collective intelligence also finds its approval in the field of deep reinforcement learning.

Besides the creation of ensembles, many extension to the DQN architecture have been made. Rainbow [1] is one successful attempt to combine important extensions into a single architecture, resulting in a significant improvement of state of the art results. Yet, there does not exist any noticeable attempt to combine such a highly optimized architecture with ensemble learning which is exactly what this report does. The motivation can be explained metaphorically by the bird example from above. There exist two possibilities to make the travel of the birds more secure. Either we select the most intelligent birds, capable of learning faster than the others, or we increase the size of the group aiming to increase the collective intelligence. The goal of this report is to make use of both approaches simultaneously in order to create a superior reinforcement learning agent consisting of the aggregation of many agents.
2.1 Background

The general setup in reinforcement learning can be described as follows: Every time step $t$, the agent selects an action $a_t$ based on the current state $x_t$, which is then executed in an environment. The environment returns a reward $r_t$ and a next state $x_{t+1}$, which are again passed to the agent. The procedure is repeated until the environments reach a terminal state or a maximum amount of steps is exceeded. Based on these observations, the agent tries to maximize the reward received during an episode. This is an extremely general setup which can be applied to a substantial amount of problems. Every agent represents a policy function $\pi(x_t, a_t)$ which assigns to every action a probability of choosing it. In case of a greedy policy, where the agent selects the action which it believes gives the highest expected reward, the policy function is deterministic with $\pi(x_t, a_t) \in \{0, 1\}$.

Every policy has a belonging Q-function $Q(x_t, a)$ which contains the expected reward when following the policy under the condition, that $a$ is the chosen action $a_t$. The expected reward is often regularized by a discount factor $\gamma$ to decrease the importance of rewards in the future. Hence, $Q(x_t, a) = \mathbb{E}[\sum_{n=t}^{t_{end}} \gamma^{n-t}r_n]$. The expectation needs to be taken since the policy can be stochastic and the $r_n$ are therefore random variables of the rewards received at time step $n$ when following the stochastic policy except for $a_t = a$. In addition, $t_{end}$ has to be viewed as a random variable. Q-learning is an important branch of reinforcement learning where such a Q-function is approximated. Every Q-function induces a greedy policy by selecting the action with the maximum Q-value. Q-learning can be used to solve control problems through continuously learning to approximate its greedy policy. By doing so, the algorithm indirectly changes the induced greedy policy which leads to a feedback loop. A fundamental theorem in Q-learning is the Bellman optimality equation which states that a greedy policy is optimal, in the sense that it achieves a maximum reward if the Q-value satisfy equation 2.1. The Bellman optimality equation induces an iterative procedure by updating the Q-value in every iteration to satisfy equation 2.1, where $x_{t+1}$ is the state reached when performing the action $a$ with state $x_t$. It is shown that for the tabular
setting where a Q-value for any reachable state is stored, this procedure converges the optimal policy for any initial Q-function [6] if all Q-values are updated simultaneously.

\[ Q^*(x_t, a) = r + \gamma \max_{\hat{a}} Q^*(x_{t+1}, \hat{a}) \]  

Equation 2.1: Bellman optimality equation

2.2 DQN-Learning

As Minh et al. [3] highlight, the DQN architecture is the first stable reinforcement learning agent based on neural networks. The network simultaneously outputs a Q-value for any action from a finite action space given a state \( x \) as input. In DQN learning, the agent explores its environment with a \( \epsilon \)-greedy exploration strategy where the action is determined by the greedy policy and with probability \( \epsilon \) a random action is taken. The observed trajectories are stored in the form of single transitions \((x_t, x_{t+1}, a_t, r_t)\). The structure which is used to store the transitions is called replay buffer and acts as a FIFO queue with a maximum limited size. After the replay buffer reaches a critical size, a small batch of transitions is sampled from the replay buffer every fourth step and the TD-loss shown in equation 2.2 is computed. \( \theta \) represents the parameters of the agent, often referred to as online network or online agent. To stabilize training, parameters from a previous version of the agent \( \theta_- \) are used in equation 2.2. The network with these parameters is referred to as target network and its parameters are updated every \( \tau \) steps from the online agent. The online network is then updated in direction of the derivative of the TD-loss. This procedure is generally called a stochastic gradient descent.

\[ \delta_{td} = r + \gamma \max_a Q(x_{t+1}, a, \theta_-) - Q(x_t, a_t, \theta) \]  

Equation 2.2

2.3 Extensions

2.3.1 Double and Duelling DQN

Both Duelling DQN [7] and Double DQN [8] are attempts to make learning more stable, which results in an enhanced performance on the Atari 2600 domain. Double DQN addresses overestimation by computing the TD-loss according to 2.3. The idea is to use a learned, but uncorrelated action \( a_{t+1} \) for the target Q-values. For reasons of computational limitations, the online network is taken.
\[ \delta_{\text{Double}} = r_t + \gamma Q(x_{t+1}, \text{arg max}_a(Q(x_{t+1}, a, \theta), \theta_\text{-}) - Q(x_t, a_t, \theta) \]  

(2.3)

In Duelling DQN the network output is separated into state value \( V(x_t) \) and advantage \( A(x_t, a_t) \). Based on these two outputs, the Q-value can be computed according to equation 2.4 where \( N_{\text{actions}} \) is the amount of possible actions. As mentioned in [7], this separation allows a better generalization while keeping the underlying structure of the algorithm the same.

\[ Q(x_t, a_t) = V(x_t) + A(x_t, a_t) - \frac{\sum_a A(x_t, a)}{N_{\text{actions}}} \]  

(2.4)

2.3.2 Prioritized Experience Replay

The original DQN architecture samples uniformly from its replay buffer. Prioritized experience replay [9] extends this behavior by weighting the samples by the last encountered absolute value of the TD-loss. The effect is that the agent focuses on unexpected transitions during learning, which when testing on the Atari 2600 domain, results in a learning curve that is twice as fast when compared to DQN [9].

2.3.3 Multi Step Bellman Update

In multi step learning [10], the Bellman update is performed over a sequence of \( n \) steps. The TD-loss rewrites as equation 2.5. As shown in [1], a small \( n \) can lead to an enhanced performance when running on the Atari 2600 environment.

\[ \delta_{n\text{-Step}} = \sum_{k=t}^{t+n-1} r_k \gamma^{k-t} + \gamma^n \max_a Q(x_{t+n}, a, \theta_\text{-}) - Q(x_t, a_t, \theta) \]  

(2.5)

2.3.4 Distributional Reinforcement Learning

The idea of distributional reinforcement learning, introduced as categorical DQN [11], is to model the Q-values as discrete distributions. The Q-values are then received by taking the expectation from the distributions. The motive is to capture the uncertainty an agent faces when acting in its environment, which can be on account of randomness in the environment but also on the limited experience an agent has in its environment.
2. Previous Work

2.3.5 Noisy Networks for Exploration

To enhance exploration compared to $\epsilon$-greedy methods, Fortunato et al. [12] extended the DQN architecture by Noisy Networks. The idea is to apply noise to the parameters of the last two fully connected layers. The linear output of the layers is changed from $y = Wx + b$ to equation 2.6 where $W$, $b$ are the weights and biases of the layer, $x$ the input and $y$ the output. $W_{\text{noise}}$ and $b_{\text{noise}}$ are two additional trainable parameters which are initialized by a constant value at the beginning. $\odot$ is the element wise multiplication. The $\epsilon$’s have the same dimension as $W$, respectively $b$, and are uniformly resampled every $n_{\text{noise}}$ steps. During evaluation all $\epsilon$ values are set to 0. As a consequence, the chosen actions might differ due to different $\epsilon$ values, which is therefore an alternative to $\epsilon$-greedy exploration.

Paper [12] shows improvements in performance on the Atari 2600 environment against $\epsilon$-greedy exploration. A reason for the success can be intuitively found in the following argumentation: An agent that has a lot of experience in a certain region is expected to be sure about the chosen action even with Noisy Nets. Also, because $W_{\text{noise}}$ are trainable parameters, an agent can learn to not explore [13] and therefore to focus the exploration on less confident areas.

$$y = (W + \epsilon_{nw} \odot W_{\text{noise}})x + b + \epsilon_{nb} \odot b_{\text{noise}}$$ (2.6)

2.3.6 Rainbow

The Rainbow architecture proposed by Hessel et al. [1] essentially combines all the previously mentioned extensions of DQN learning into a single agent. This agent significantly raised state of the art results on the Atari 2600 environment and is used as a baseline for recent publications like [13]. A fundamental strength of Rainbow is its sample efficiency compared to each of the single extension. The disadvantage is that due to the high computational complexity, the time efficiency is comparably bad. A rigorous analysis of the importance of any extension and a comparison between performances on the Atari 2600 games can be found in [1].

2.3.7 Parallelized Reinforcement Learning

There exist many attempts to introduce parallelization into reinforcement learning and one of the recently most notable ones is A3C [14]. In this work we restrict ourselves to an approach based on DQN learning, Distributed Prioritized Experience Replay [15]. The algorithm contains several working threads, each acting in a separate environment and asynchronously sending the collected trajectories in form of batches to a learning thread. Every $n_{\text{update}}$ steps, each worker updates its parameters from the learning thread. The learning thread trains its parameters by performing $n$-step Bellman updates as described above. In order to handle the significantly higher amount of transitions, Prioritized Replay is used for sampling.
One surprising result is that simply by extending the amount of working threads, the performance on the Atari 2600 environment improves drastically. The downside is that the sample efficiency worsens. Figure 2.1 visualizes the dependency of the performance on the amount of workers, while keeping the amount of trained batches on the same level.

2.3.8 Off Policy Correction

When performing multi step Bellman updates as described earlier, the actions taken in trajectories sampled from the replay buffer can differ from the actions preferred by the online policy. This can either be due to a high exploration where actions apart from the current policy are chosen on purpose to expand the experience, or because the Q-values change over time. For the one step chase this is not a problem but in case of several steps, the trajectories should not differ too
much from the trajectory that the online policy would choose. To address this issue, several attempts to adapt the target Q-value exist. One recent attempt is Retrace [16]. The adapted n step TD-loss for DQN is shown in equation 2.7.

$$\delta_{\text{Retrace}} = \sum_{k=t}^{t+n-1} \gamma^{k-t} \left( \prod_{i=t+1}^{k} c_i \right) [r_k + \gamma \sum_a \pi(a, x_{k+1}) Q(x_{k+1}, a) - Q(x_k, a_k)] \quad (2.7)$$

Where $c_i = \min(1, \frac{\pi(a_{i+k}, x_{i+k})}{\mu(a_{i+k}, x_{i+k})})$. $\pi(a_{i+k}, x_{i+k})$ represents the probability of selecting the action $a_{i+k}$ under the policy which generated the trajectory and $\pi(a_{i+k}, x_{i+k})$ the probability of the online policy. For DQN with $\epsilon$-greedy exploration, these probabilities are usually $1 - (n - 1)\epsilon$ for the maximum action and $\epsilon$ for the other actions.

### 2.3.9 Ensemble Methods

The basic assumption behind ensembles is that, overall, the average over many approximating functions is a better approximation than each function itself. This concept has been successfully applied to DQN-learning [4, 5, 17, 18]. One interesting fact is that majority voting is a better way to combine the ensemble than averaging [19].

In Bootstrapped DQN [4], $K$ Q-functions are each computed by a single head, with all of them optionally sharing the same convolutional layers. Each head has its own parameters and hence represents a different Q-function. In the beginning of every episode, a head is randomly selected and an episode is run using the $\epsilon$-greedy policy of the Q-function of the chosen head. For training, each head has its own target network, again optionally sharing the convolution layers, and its own replay buffer. Any transition is added to each replay buffer with an independently sampled probability for each head. The reasoning behind this is to keep a high variance among the different heads. Bootstrapped DQN has shown superior performance in terms of peak performance over DQN. A fundamental strength over DQN beside the ensemble effect is its improved exploration due to the variance among the heads.

Another approach is given by Chen et al [5]. They proposed an ensemble algorithm where an ensemble is created by training $K$ independent agents, with all agents sharing the same replay buffer. The trajectories are following majority voting among the agents except for random actions with probability $\epsilon$. Tests in the Atari 2600 environment show superiority in performance over Double DQN learning [5]. One disadvantage is that ensemble learning requires considerably more computational resources and needs a careful parallelization to get comparable time efficiency.

Beside creating a more robust actor through majority voting, a fundamental strength of using ensembles, which only applies to reinforcement learning, is that
exploration can be increased through sharing trajectories. This is a two sided effect with the drawback that the trajectories from the replay buffer are in a higher degree off policy. At this point it needs to be specified what off policy is and how to understand the notation "high degree off policy".

Per definition, any trajectory which is not generated by the online policy is called off policy. However, training the parameters with an appropriated learning rate does not usually change the parameters too much. Two temporal versions of the same network do normally only differ significantly if the time difference is high. Therefore, trajectories following a previous version of the network are expected to be similar to trajectories, that the online network would choose. The reason is because the outputs of both networks are expected to be highly correlated. This, on the other hand, is not the case if the trajectories are following a network with a different initialization. When an issue related to off policy is mentioned during the report, this issue is referred to.

Another critical aspect which might occur in ensembles is pointed out by Harutyunyan et al: "... ensembles of diverse Q-learners are bound to have larger disagreement amongst themselves and with the behavior policy, and have a much larger potential of becoming unstable" [20]
3.1 Open Research Questions

3.1.1 Ensemble Methods and Rainbow

Both ensemble methods, like \cite{4, 5}, and Rainbow have shown improvements on DQN. While the approach used for Rainbow is primarily taken to make extensions in the architecture, the presented ensemble methods rely on single step Bellman updates and do barely make use of architectural extensions. It is not yet clear whether it is possible to make use of both extensions by combining them into a single architecture.

One strong argument for a successful combination can be based on the different approaches that were taken by both algorithms. Rainbow has made no specific improvements in exploration, except for Noisy Nets. The improvements are mainly addressing the ability to learn from the trajectories. The presented ensemble methods, on the other hand, mainly increase their performance by means of a better exploration and by majority voting.

However, there are several critical open questions that need to be answered to ensure success. The most important one might be the fact that when using a shared replay buffer the trajectories are in a far stronger degree off policy. The share of trajectories generated by a previous version of each agent is reduced to $\frac{1}{N}$. Multi step reinforcement learning, on the other hand, is based on the assumption, that the trajectories are not too far away from the online policy. Furthermore, the side effects which may occur when combining the extensions used in Rainbow, such as categorical DQN with a shared replay buffer, are neither analyzed nor is it possible to give a theoretical answer. Especially when using a high share of off policy trajectories and prioritized experience replay, the current policy might be disturbed due to a preference on off policy trajectories during sampling from the replay buffer.
3. Analysis of Literature

3.1.2 Retrace and Rainbow

The use of \(n\)-step Bellman updates in Rainbow is one strong factor that is responsible for its superior performance [1]. \(n\)-step Bellman updates do not include any off-policy correction, which is one of the reasons why multi-step learning can be performed over only three steps. We did not find any research on the usage of Retrace in combination with Rainbow.

A motivation for this approach is that multi-step Bellman updates, as used in Rainbow, do not involve off-policy correction, which can be quiet easily done by including Retrace. The effect of Retrace is that the target Q-values become more reliable in the view of the online agent. This effect has shown improvements on DQN learning.

Reactor [13] introduced an adaptation of retrace for categorical architectures but is itself an actor-critic algorithm, which makes it difficult to draw conclusions about the effects of Retrace on Rainbow. In addition, the side effects that might occur when combining prioritized experience replay with retrace are not clear. In its simplest form, Retrace stops as soon as the originally chosen action and the current maximum action of the online network are not the same anymore. It is expected that the loss increases with the amount of steps taken for the Bellman update. In combination with prioritized experience replay, the learning might concentrate too much on online policy trajectories.

3.1.3 Shared Target Networks for Ensembles

Paper [21] showed that for DQN, learning can be improved by taking an average of target Q-values, with target networks from different previous time steps. This approach shows similar effects as Double DQN learning in the sense that both are addressing overestimation. Similar to this approach, it is possible to take a linear combination of the target networks among the distinct agents trained in an ensemble. More formally, the concept is the following: Each target network \(Q^t_k\) is a copy of the online network \(Q_k\), which is updated every \(\tau\) steps. Instead of solely using \(Q^t_k\) as target network, \((1 - \alpha)Q^t_k + \frac{\alpha}{K} \sum_{j \in [1,K]} Q^t_j\) can be used for the Bellman update.

The aim is to force agents to learn from the experience that other agents gained. Each agent tries to find an approximation of an optimal Q-value. By means of the above described concept two aims shall be achieved. The first one is that the variance is expected to decrease, resulting in more stable target Q-values. This is also experienced in [21], where the average over Q-functions achieves better performance than the one of a single agent. The second intention is to profit from other networks, which perform better in certain regions. However, it has to be noted that this is far stronger than only sharing trajectories in the replay buffer. The proposed concept can also be a drawback in the sense that because the target Q-values are dependent, the variety among the agents decreases and
3. Analysis of Literature

the effect of ensemble vanishes. Another possible drawback is that the target Q-values get less expectable for each agent, leading to higher losses and therefore making the convergence more difficult or even causing divergence to be more likely.

3.2 Approach to Research Questions

In order to propose a research question, this report investigates whether it is possible to enhance the performance of Rainbow when combining it with the concepts from ensemble methods and Retrace. Based on the thoughts of the last section, two leading research questions are proposed:

1) Can an ensemble architecture enhance Rainbows performance?

2) Can the results from question 1 be improved by including Retrace?

In addition, the question about sharing the target networks, as described in 3.1.3, was investigated but only took on a minor role. The methodology consists of the creation of explicit algorithms and testing them on a subset of the Atari 2600 domain. This methodology is used in a variety of recent publications, including many of the ones introduced. Due to time and resource limitations, a necessary hyper parameter search cannot be carried out. The approach taken in this report splits into two parts. In a first part, several experiments are run on the game boxing. These experiments have to be seen as prototypes to identify promising approaches and to reduce the amount of tests needed for the second part. An overview over this part can be found in the appendix. The second part covers the scientific demands and diligently compares selected algorithms on a subset of games of the Atari 2600 domain. A clear description of the experiments and the results can be found in the next chapter.
Chapter 4

Experiments

4.1 Algorithms

4.1.1 Basic Ensemble

The challenge occurred while designing this algorithm was to profit from a strong parallelization while keeping the algorithm synchronized and similar to the Rainbow architecture. The chosen structure is visualized in figure 4.1. As a basis for this implementation the code from https://github.com/Kaixhin/Rainbow is used. The replay buffer contains a list of transitions and for any learner a separated list with the corresponding priority weights. Each worker has its own environment, where it interacts by following its own policy. For exploration Noisy Nets are used, which is resampled every four steps. Every 100 steps, each worker updates the parameters by copying them from the learner and inserting its trajectories into the replay buffer, with maximum weight for any priority weight list. At the same time, the learner samples 100 batches from the replay buffer and trains them exactly as described in [1] but without noise for neither the target nor the online network. This change had to be made to get a reasonable learning curve. Figure 4.2 shows the learning curves on boxing for Ensemble with Retrace which is introduced in the next section with and without noise during training. Similar curves were also observed for Basic Ensemble, which is why we decided to omit noise for both algorithms. It is important to mention that noise was only disabled during the training of the batches and not during exploration in the environment. The disadvantage is that in this setup, the weights $W_{\text{noise}}$ as in subsection 2.3.5 are kept constant and therefore the agent cannot reduce the noise. Since the algorithms were able to reach peak performance in Boxing, we decided to keep this setup. Unfortunately, it was not observed what happens when Rainbow is run without Noisy Nets during training. Any other hyper parameter is taken from the official Rainbow paper.

On a lower level, each learner and each worker is run by a separated process. The learner and the worker are synchronized over a master process which com-
4. Experiments

Figure 4.1: Structure of Basic Ensemble Algorithm

4.1.2 Rainbow with Retrace

Retrace [16] was originally designed for DQN without the categorical architecture. Like in [13], the algorithm can be written as equation 4.1 with $\alpha_{n,a}$ defined in equation 4.2 and $c_{n,i} = \lambda \min(1, \frac{\pi(a_{n+1})}{\mu(a_{n+1})})$. We choose $\lambda = 1$ and set $\pi(a, x_t)$ to equal one for the action $a = a_{\text{max}}$, with $a_{\text{max}}$ the action of the maximum Q-value, and zero for every other action. $\mu(a, x_t)$ is set to be one for $a = a_t$ and zero for any other action. Equation 4.1 then rewrites as equation 4.3 with $\hat{n}$ the minimum $n \geq 1$ such that $a_{\text{max}} \neq a_t$. It is important to stress that for simplicity we set $\pi$ and $\mu$ to be deterministic policies. This simplification was done to keep the algorithm as close as possible to a multi step Bellman update and because we use Noisy Nets for which it is difficult to model the stochastic policy when taking the noise into account.

$$\Delta Q(x_t, a_t) = \sum_{n \geq 1} \sum_{a} \alpha_{n,a} \left[ \sum_{s=t}^{t+n-1} \gamma^{s-t}r_s + \gamma^n Q(x_{t+n}, a_{t+n}) - Q(x_t, a_t) \right]$$  (4.1)

$$\alpha_{n,a} = (c_{t+1}...c_{t+n-1})(\pi(a, x_{n+t}) - I_{a_{n+t}=a_c_{n+t}})$$  (4.2)
Figure 4.2: Ensemble with Retrace, with and without noise during the training, compared to Rainbow on Boxing. All hyper parameters for evaluation are exactly the same as described in figure 4.3.

\[
\Delta Q(x_t, a_t) = \sum_{s=t}^{t+\hat{n}-1} \gamma^{s-t}r_s + \gamma^\hat{n}Q(x_{t+\hat{n}}, a_{t+\hat{n}}) - Q(x_t, a_t) \tag{4.3}
\]

Equation 4.3 can be easily adapted to the categorical setup since the target networks are exactly as in the \(n\)-step Bellman update but with \(\hat{n}\) variable. This change is essentially the only modification which was made against Basic Ensemble. One important detail has to be mentioned. The priority weights are computed from the losses. In this case, however, the target Q-values do not need to have the \(\hat{n}\) in common. It is expected that updates with a higher \(n\) lead to a higher loss which then leads to a higher priority weight. Hence the priority weight sampling might get lost in a feedback loop. As an alternative, additional losses with a fixed \(n\) of three were computed and solely used for the prioritized weight update. Tests on Boxing have not shown any significant difference between the two versions, which is why we decided on the first version since it is slightly more efficient. The alternative of disabling prioritized sampling was not an option since it was shown in [1] that beside \(n\)-step this is one of the most important extensions in terms of the performance of Rainbow.

4.2 Experimental Setup

Six environments for extended tests have been selected. Beside Boxing the environments Breakout, Bank Heist, Enduro, Q-Bert and Seaquest were chosen. The aim was to cover a variety of different challenges especially in terms of com-
plexity. As python implementation of the environments the Arcade Learning Environment [22] is used. Except for the hyper parameters described in the last two subsection, all hyper parameters are taken from [1]. As proposed in [1] the noise standard deviation for the Noisy Nets was set to 0.1 instead of 0.5. The tests were run for 10 million training steps. Due to externalities some tests had to be stopped after 9.5 million steps. Because the learning behavior leads to meaningful conclusions, we decided to not repeat these experiments again. For the ensemble architectures the total amount of steps is taken. The evaluation is taken for the ensemble versions every 500’000 steps and for Rainbow every 250’000 steps. Because frame skipping is used, every step corresponds to 4 frames. Noisy Nets was disabled during evaluation and replaced by a $\epsilon$-greedy exploration with a $\epsilon$ of 0.0001. The reported results are the average over a single training run with 10 evaluation runs. The fact that we had to restrict it to a single training run is not ideal but it was the only option due to resource limitations.

The baseline tests were run on a single NVIDIA GTX 1080 ti GPU. For the tests with ensembles, three GPUs of the same model were used to increase time efficiency.

4.3 Results and Analysis

The learning curves and the peak results are shown in figure 4.3 and table 4.1. Neither Ensemble with Retrace nor Basic Ensemble show a clear advantage over Rainbow. In the games Boxing, Bank Heist and Enduro an improvement is visible. The opposite is true for Breakout and Seaquest. For Seaquest, both ensemble versions do not show a promising learning curve during the first 10 million steps. The appearance is even strengthened when analyzing the single network performances of the ensemble which show a similar trajectory and therefore the learning curve cannot be based on majority voting. The problem concerning the instability of the algorithms has been verified as both algorithms face divergence in at least one of the six tested environments. Interestingly, when comparing the average Q-values from Rainbow with the ones from the ensemble versions during the different tests, no significant difference can be found even for agents before diverging.

The strong performance on Enduro, where both algorithms break state of the art results and the bad performance on Seaquest are good examples to explain the advantages and disadvantages of the chosen method. In Seaquest the algorithm needs to learn two difficult tasks to get a high rewards, which are both not immediately obvious: filling up the air tank and rescuing persons. Both actions do not give instant reward. Although the exploration is expected to be higher through sharing the replay buffer, the agents are not capable to learn such complex tasks. We believe the reason for this issue is the high degree of off policy trajectories.
4. Experiments

Figure 4.3: The algorithm Basic Ensemble and Ensemble with Retrace are compared to the reproduction of Rainbow referred to as Baseline. The evaluation is performed for the Baseline every 250 thousand steps and for the ensemble versions every 500 thousands steps. The mean is taken from a single training run with 10 evaluation runs. The ensemble is created by majority voting of the agents.
Table 4.1: The peak performance of the investigated algorithms over a time frame of 10 M training steps are shown. The evaluation details are taken to be exactly the same as described in figure 4.3. In addition, peak performances for DDQN and Rainbow trained over 50 million steps are taken from [1]. These test were executed under different testing conditions.

Because learning is restricted on short transitions, the agent needs to be able to solve complex tasks only by extracting information from these transitions. To achieve a high reward, it is necessary that the extracted information comes to the conclusion to favor not directly obvious trajectories. We belief that highly uncorrelated trajectories make this process much more difficult. The other example, Enduro, shows exactly an opposite behavior. The task is easier and both, the agents and especially majority voting, show superior performance over Rainbow. Besides the deactivation of Noisy Nets during training, which makes over fitting to a high reward trajectory easier, the high amount of different trajectories in the replay buffer seem to enhance the training and hence performance. We believe these different behaviors arise because trajectories in Enduro naturally cannot vary too much due to the structure of the game.

When comparing Rainbow with Retrace with Basic Ensemble, no remarkable difference can be observed. It seems that Basic Ensemble is learning slightly faster than Rainbow with Retrace which is expectable due to the off policy correction. Since only one training run has been made for each game and the learning curves are very similar, it is not possible to offer a deeper analysis.

The tests were not run in a setup which allows us to compare time efficiency since the computational speed highly depends on the external workload on the server. It was, however, possible to measure the peak throughput in steps per second when the server was not used by any external process and with a full replay buffer. Rainbow reached 110 steps per second, Ensemble with Retrace 133 steps per second and Basic Ensemble 166 steps per second. The increased time efficiency is mainly due to the exclusion of Noisy Nets during the training of the batches. The parallelization also influences the time efficiency but due to the fact that any agent needs to learn the same amount of batches per step as Rainbow, a parallelized architecture cannot be far more time efficient than Rainbow.
The motivation behind this work was to combine ensemble methods with a highly specialized architecture resulting in an enhanced performance. This aim was not reachable for more complex environments of the Atari 2600 domain. We believe the main issue is that the used architecture, Rainbow, is highly sensitive to the trajectories in the replay buffer. This argument is based on experiments showing an increased risk of divergence when being trained with a shared replay buffer from agents with different initializations. The experiments have shown that the introduced algorithms Basic Ensemble and Ensemble with Retrace cannot benefit from the experience among the different agents but even impair each other. This is, however, not the case for simpler environments where the proposed architectures showed superior performance over Rainbow.
Bibliography


A pervasive constraint in deep reinforcement learning is the considerable time consumption needed to train an agent. For example, 10 days are needed to train Rainbow for 200 million frames on a single game of the Atari Domain. This forces the author to carefully select the experiments. As a compromise between rigorouslyness and variety among the experiments, we decided to invest a considerable time in creating prototypes in order to identify auspicious versions which are then used for further analysis. As a basis for any experiment the Rainbow implementation provided by https://github.com/Kaixhin/Rainbow is used. Unless stated otherwise, any test in this section is run on the game Boxing. For the implementation of these games the Arcade Learning Environment [22] is used. This chapter of the appendix briefly summarizes the work done during the report and explains important decisions being made like the selection of the algorithms.

A.1 Naive Implementation of Ensemble with Rainbow

After several tests in a toy environment, which was primarily used for debugging and to introduce the author to the field, we started with a naive implementation of an ensemble based on Rainbow. The implementation was highly optimized, as it is used as a base code for the succeeding work. The fact that this work started directly with all of Rainbows extensions was clearly a compromise. On the one hand, the chosen approach does not give a coherent clarification about unexperienced combinations such as ensemble with categorical DQN. Only the whole combination is tested and it is therefore not possible to rigorously analyze the interdependencies between the extensions in an ensemble setup, which makes the algorithms less understandable. On the other hand, such an analysis would take up too much time, leaving no remaining time to make further experiments. For this work, the latter argument was dominant.

The first serious investigated algorithm, Basic Ensemble, was explained in detail in chapter 4. A comparison of the the learning curves, shown in figure A.1, lead to the following conclusions: Both Rainbow and Basic Ensemble show a similar
Figure A.1: The performance of the Basic Ensemble when trained on Boxing is compared to Rainbow. The evaluation details are described in figure 4.3. Additionally, the learning curve of each agent of Basic Ensemble is shown.

performance with both reaching the maximum reward of 100.

The effect of the majority voting is clearly visible and even with one agent diverging, the ensemble remains stable. The divergence of an agent in a time frame of less than 10 million steps is a concern which changes the positive outlook. When analyzing the average Q-values of the diverged actor, no anomaly can be found in comparison with the average Q-values of Rainbow and the other actors. This leads to the conclusion that the issue is not due to an uncontrolled overestimation but due to a too high loss encountered while training. We believe this is a cause of the high amount of off policy trajectories in the replay buffer.

A.2 Ensemble with Retrace

To make learning more stable and reduce the chance of divergence, we decided to include an off policy correction instead of reducing to the one step Bellman update. Multi step learning has shown to be essential for Rainbows strong performance [1] and we did not want to lose this advantage. The algorithm which was used for the tests is described in chapter 4 and called Ensemble with Retrace. Additionally, a deviation is created and in the following referred to as Ensemble RD. This version follows the lines from Ensemble with Retrace, except that it uses the parameters $\theta_{\text{double}}$ from the target network of another agent to estimate the double network action. More explicitly, $Q(x_{t+n}, \arg\max_a Q(x_{t+n}, a, \theta), \theta_-)$ becomes $Q(x_{t+n}, \arg\max_a Q(x_{t+n}, a, \theta_{\text{double}}), \theta_-)$. Figure A.2 shows the learning curves compared to Rainbow. Both Rainbow and Ensemble with Retrace show a similar learning curve. Conspicuous is the high variance in performance that
the agents face. Through majority voting the deflections can be compensated. Nevertheless, it is questionable how strong this drawback affects the ensemble on more complex tasks. We assume that the reason for the high variance each agent faces is again due to the high amount of off-policy trajectories. Regarding Ensemble RD, the amount of steps for reaching the peak performance is worsened. Additionally, one agent diverged after 9 million steps. We assume that by selecting the maximum action from the double network, the agent faces too much noise during training. In this case, the double network needs to be regarded as additional noise since the online network has no knowledge about the actions selected by the double network. This is clearly not the case when the double network is the online network itself.

So far, the authors came to the following conclusion: For the ensembles the performance is comparable to Rainbow. It even seems that the learning curves are more stable which can not be verified due to the different evaluation frequency. The fact that all algorithms reach peak performance and therefore might not have shown the full potential is clearly promising.

### A.3 Analysis of Retrace Alternatives

A disadvantage of Retrace is that the target Q-values might change significant when the preferred action changes due to a small change in the Q-values of the online network. This, however, is not the case when using $n$-step Bellman updates. We have made several attempts to improve this issue with limited success.
when evaluating on Boxing which is why only the key ideas are presented. It was experienced that the Rainbow architecture is highly sensitive to operations on the target Q-values. Especially when working with the categorical DQN architecture where the network outputs are distributions, off policy correction shows to be a difficult task.

The fundamental idea for the work on improvements on off policy algorithms comes from the Reactor architecture [13] where an actor critique architecture, which includes categorical DQN, is introduced. In order to train batches from a replay buffer a modified version of Retrace for categorical DQN is taken in this architecture. The target distributions are defined to be \( \sum q^*_i(x_t, a_t)\delta_{z_i} \), where the \( z_i \) come from the discretization as in [13].

\[
q^*_i(x_t, a_t) = \sum_{n \geq 1} \sum_j q_j(x_{t+n}, a_{t+n})h_{zi}(z^n_j) \tag{A.1}
\]

A problem here is that DQN does only imply a greedy policy and not a smooth policy as the actor critique architectures do. The idea at this point was to artificially construct a policy out of the DQN values. For two versions, prototypes were created following equation A.2 and A.3. The first version explicitly relies on the categorical architecture. In this version the Q-values are regarded as random variables distributed according to the output of the network. The policy is set to be the probability that \( q(a, x_t) \) is the maximum Q-value. The second version, which is called Softmax Policy Selection [10], is based on the Q-values. In case of a categorical architecture the Q-values are the expectation of the distributions \( Q = \mathbb{E}[q] \). We define \( \mu(Q) \) to be the average of the Q-values and \( \sigma \) to be the root of the variance.

\[
\pi(a, x_t) = P[max_\hat{a}(Q(\hat{a}, x_t)) = Q(a, x_t)] \tag{A.2}
\]

\[
\pi(a, x_t) = \exp\left(\frac{Q(a, x_t) - \mu(Q(x_t))}{\sqrt{N\sigma(Q(x_t))}}\right) \tag{A.3}
\]

Several unsuccessful implementations of these concepts led to the following conclusion: For both versions, the policy is extremely flat and equally distributed over the actions. To overcome this issue we changed the \( \alpha \) values from equation A.1, to \( \alpha_{n,a} = (c_{t+1}...c_{t+n-1})(\mathbb{I}_{a=a_{max}} - \mathbb{I}_{a_{n+i}=a_{n+i}}) \) with \( a_{max} \) the maximum action from the online network. This change led to a promising learning curve starting with a significantly faster learning when compared to Rainbow. The disadvantage is that no version achieved a peak performance over 92.

Another approach extended the output of the architecture by a policy. The policy is updated every batch based on the algorithm \( \beta_{los} \) introduced in [13]. The
policy is only used for off policy correction when computing the target Q-values and not for exploration because we wanted to rely as much as possible on the Rainbow architecture. The off policy algorithm is the above described adaptation of Retrace for categorical DQN. Multiple tests with a prototype implementation showed bad stability and no sights of convergence. At this point it has to be mentioned that this behavior does not exclude the potential of these algorithms. Several other reasons could have led to these results including a lack of hyper parameter search. The criterion was to find an alternative to retrace without a costly hyper parameter search and with similar or faster learning curves. As a last test series in this category the categorical architecture was removed from Rainbow and replaced with scalar Q-values. The above mentioned implementation have been tested on this setup. By mistake, the hyper parameters were not properly tuned, causing the algorithms to not learn.

A.4 Improvements on the Ensemble

Several attempts to Improve the Basic Ensemble beside Ensemble with Retrace have been made. All of them use Basic Ensemble as base code.

A.4.1 Favor own Trajectories

The described algorithm from this subsection is called FT Ensemble. The structure is essentially the same as Basic Ensemble except for a change in sampling the transitions from the replay buffer. The idea is to favor the trajectories generated by the belonging agent. The belonging agent is the agent which repetitively copies the parameters from the learner with the corresponding prioritized weight list. Originally, the prioritized weight $p_{k,i}$, in this section for simplicity called weight, is insert into the replay buffer by setting it to be equal to the maximum weight $p_{k,max}$. After a transition occurs in a batch, the weight is updated to be equal to the $TD$-loss. The probability that a sample occurs in a batch is $P(k,i) = \frac{p_{k,i}}{\sum_{i=1}^{N} p_{k,i}}$, with $N$ the number of samples in the replay buffer. The index $k$ exists because any learner has its own prioritized weight list. During learning from the batches, FT Ensemble favors transitions from the belonging agent by initially setting the weight of transitions from other agents to $\delta_l \ast p_{k,max}$ and also scales these weights by the same factor after occurring in a batch. In this way, the probability that a transition generated by an different agent occurring in the batches is reduced.

To reduce a bias toward transitions with high probability, Schaul et al [9] introduce a weight $w_{k,i}$ for each transitions in a batch which is then multiplied to the loss during backpropagation. $w_{k,i}$ is computed according to equation A.5. $\beta$ is some hyper parameter which is set to 0.5 and increased over time to one.
Prototypes

\[ w_{k,i} = \frac{1}{(NP(k,i))^\beta} \] (A.4)

Because we explicitly want the bias toward sample which were generated by the corresponding agent, equation A.5 was changed to
\[ \hat{N} = \frac{N(1+\delta_l(K-1))}{K} \]
where \( K \) is the amount of different agents. Further \( \hat{P}(k,i) = \frac{P(k,i)}{\delta_l} \) if the transition was not generated by the corresponding agent and \( \hat{P}(k,i) = P(k,i) \) else.

\[ w_{k,i} = \frac{1}{(NP(k,i))^\beta} \] (A.5)

FT Ensemble was tested on Q-Bert because this game has shown to be the most sensitive to divergence among the tested environments. Tests with slightly adapted hyper parameters and a \( \delta_l \) of 0.2 and 0.05 have still lead to divergence.

### A.4.2 Use common Initialization

Another attempt to improve Basic Ensemble was, instead of using a different initialization for every agent, to use the same set of parameters as initialization for any agent. In addition, a small amount of independent noise for each agent was added to the parameters during initialization, to ensure a diversity among the agents. More formally, during initialization, a parameter set \( W \) is randomly drawn according to the specified rules. The convolution layers of each agent are initialized by \( W_{CNN} \subset W \) and the fully connected layers are initialized by \( W_{FC} + W_{\text{noise},i} \), with \( W_{FC} \subset W \) and \( W_{\text{noise},i} \) independently sampled according to the same rules but additionally scaled by a factor of \( \epsilon_{\text{init}} \) which is set to 0.05. The idea behind this procedure is to make the agents correlated because the policies are expected to be more similar compared to previous versions, resulting in an increase stability during training. Unfortunately, when testing on Q-Bert the agent still faced divergence and therefore the correction has shown to be insufficient.

### A.4.3 Shared Target Network

Inspired by the research question from subsection 3.1.3, we implemented the concept of this subsection and tested the implementation on Boxing where Basic Ensemble performs well. The training curve did not show a promising learning curve in the first 10 million training steps. The hyper parameter \( \alpha \), specified in subsection 3.1.3, is tested for 0.25 and 0.1.