Reinforcement Learning Project

Rainbow: Combining Improvements in Deep Reinforcement Learning

Done by :
Mohamed Wadhah Dahouathi
Sami Elmokh
Mahdi Attia

Supervised by :
Erwan Le Pennec

Academic Year : 2023-2024

Contents

Introduction

In this report, we analyze a pivotal piece in artificial intelligence, specifically in reinforcement learning (RL). Unlike traditional machine learning paradigms that depend heavily on pre-existing knowledge or instructional guidance, RL thrives on the principle of experiential learning—where agents derive insights through direct interaction with their environment. This distinctive approach crosses over disciplinary boundaries, such as robotics and cognitive science, and highlights the potential of RL to transform how machines learn and adapt.

The focus of our discussion is on a pioneering study that proposes an integrated framework known as the 'Rainbow model'. The evolution of Deep Q-Networks (DQN) is being topped by this model, which combines multiple key enhancements to create a learner of unprecedented capability. Our narrative will dissect the methodology behind the Rainbow model, elucidating on how the confluence of various DQN extensions can orchestrate a leap in performance, particularly demonstrated through its remarkable achievements on the Atari benchmark.

To set the stage for this discussion, we'll first explore the fundamental principles of reinforcement learning and then gradually uncover the details of the Rainbow model. This journey will include a careful exploration of each constituent improvement, their individual and collective impacts on the model's efficiency, and the overarching implications of such advancements in the broader landscape of AI research.

By charting the progression from elementary RL concepts to the architectural refinement of the Rainbow, our aim is to encapsulate the essence of the referenced article [11], providing a detailed synopsis that not only foregrounds the technical accomplishments but also situates them within the broader narrative of artificial intelligence investigation. Our ultimate goal is to implement this algorithm, thereby empirically validating its results.

Chapter 1

Background

1.1 Agents and Envinroments

Before we explore the complexities of the Deep Q-learning algorithm and its enhancements, let's clarify some core concepts of reinforcement learning.

- **The Agent:** This is our learner or decision-maker, akin to a character in a video game, navigating through challenges and opportunities.
- **The Actions:** These are the choices our agent can make, similar to the moves or decisions in a game.
- **The Environment:** This represents the world in which the agent operates, reacting to the agent's actions and providing feedback in the form of changes or rewards.
- **The State:** It's the current condition or scenario the agent finds itself in, like a snapshot of the game, showing the agent's position and surroundings.

The essence of reinforcement learning is for the agent to learn from trial and error, aiming to discover the best actions that maximize rewards without explicit instructions. This learning process involves developing a strategy or policy, which guides the agent's decision-making based on the current state to optimize the reward outcome.

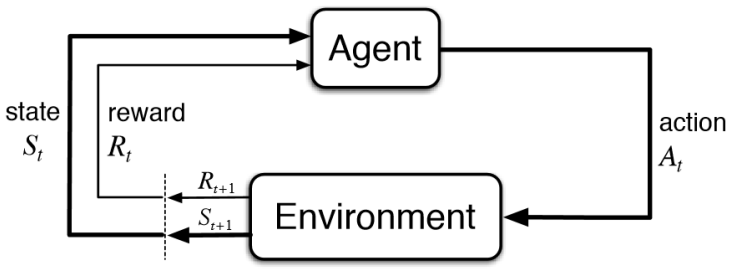


Figure 1.1: Interaction between the agent and the environment [2]

To structure these interactions and learning processes, we utilize the concept of Markov Decision Processes (MDP). MDPs offer a mathematical framework to model decision-making where outcomes are partially random and partially under the agent's control. They comprise:

- **S (States):** A finite set of scenarios describing the environment.
- **A (Actions):** A finite set of possible actions the agent can take.
- **T (Transitions):** Probability functions determining the likelihood of moving from one state to another, given an action.
- **R (Rewards):** Feedback functions assigning rewards or penalties based on the agent's actions in certain states.

MDPs help in breaking down the agent's journey into manageable components, paving the way for a deeper understanding of Deep Q-Learning and its enhancements.

1.2 Definition of DQN and its Extensions

Deep Learning plays a pivotal role in managing complex or high-dimensional state and action spaces by mapping these to a simpler, lower-dimensional representation. This is particularly effective in reinforcement learning, where neural networks, trained through gradient descent to minimize loss, learn the Q-values associated with actions in given states. A notable example of this technique's success involved the use of convolutional neural networks to refine action value approximations, as demonstrated in significant research works.

Applying deep learning to approximate value functions, however, does not automatically ensure smooth convergence. The process can be hindered by weight oscillations and divergence, caused by the tight correlation between states and actions. To address these challenges and enhance training stability and efficiency, two innovative techniques are employed:

- **Experience Replay:** This strategy involves storing and later revisiting past experiences, or tuples of states, actions, and rewards, to reduce correlation between sequential learning steps. By sampling from this diverse pool of experiences, the network can learn from a broader spectrum of scenarios.
- **Target Network:** Implementing a secondary, more stable network to estimate future Q-values helps in stabilizing the learning updates and mitigating the issue of rapid value estimation shifts.

Experience Replay

For SGD (Stochastic Gradient Descent) optimization to work effectively, it's crucial that the data samples are independent and identically distributed. However, in the context of an agent learning from interactions with its environment, the sequence of experiences can be highly correlated. This correlation might skew the learning process, leading a straightforward Q-learning approach astray.

To mitigate this, a technique known as **experience replay** is employed. It involves maintaining a buffer of past experiences, or transitions $(S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1})$, in a fixed-size deque. Rather than relying solely on recent experiences for training, data is sampled from this buffer. This method not only disrupts the problematic correlations but also enriches learning by revisiting past experiences multiple times and recalling infrequent events.

Target Network

Updating the parameters of a neural network to optimize the Q-value $Q(s, a)$ can accidentally affect the subsequent Q-value $Q(s', a')$, introducing instability into the training process. To address this, a practical solution is the use of a **target network**. This involves creating a duplicate of the neural network whose Q-values $Q(s', a')$ are used in the Bellman equation for stabilizing updates . The predicted Q values of this second network are used to backpropagate and train the main network. The parameters of the network are optimized by minimizing the following loss function:

$$(R_{t+1} + \gamma_{t+1} \underset{a'}{max}\, q_{\bar{\theta}}(S_{t+1}, a') - q_{\theta}(S_t, A_t))), \tag{1.1}$$

where the gradient of the loss is computed with respect to the parameters θ, and $\bar{\theta}$ represents the parameters of the target network, which are updated less frequently to ensure stability.

1.2.1 DQN: Deep Q-Networks algorithm

Deep Q-Networks (DQN) revolutionized reinforcement learning by integrating deep neural networks to approximate Q-values, a critical component in the Q-learning framework. DQN is known for its ability to handle complex environments by using a model-free approach, which involves storing and sampling experiences (state, action, reward, next state) for training via a replay buffer. The combination of this mechanism and an epsilon-greedy strategy allows for precise balance between exploring new actions and exploiting known rewards.

A distinguishing feature of DQN is the employment of a target network, an innovative solution to stabilize training by periodically syncing with the main network's weights. This architectural advancement addresses the oscillation challenges inherent in rapid Q-value estimations.

Despite its success across various benchmarks, including Atari games, DQN encounters challenges such as Q-value overestimation, high data demand, and scalability issues. The algorithm operates in two phases: collecting experiences to populate the replay buffer and leveraging these memories through mini-batch SGD training. DQN's efficiency in learning and adapting to diverse environments is supported by this dual-phase process.

Initialize network Q
Initialize target network $\hat{Q}$
Initialize experience replay memory D
Initialize the *Agent* to interact with the Environment
while *not converged* **do**

  `/* Sample phase`
  $\epsilon \leftarrow$ setting new epsilon with ϵ-decay
  Choose an action a from state s using policy ϵ-greedy(Q)
  Agent takes action a, observe reward r, and next state s'
  Store transition $(s, a, r, s', done)$ in the experience replay memory D

  if *enough experiences in* D **then**
    `/* Learn phase`
    Sample a random *minibatch* of N transitions from D
    for *every transition* $(s_i, a_i, r_i, s'_i, done_i)$ ***in*** *minibatch* **do**
      if $done_i$ **then**
        $y_i = r_i$
      else
        $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
      end
    end
    Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2$
    Update Q using the SGD algorithm by minimizing the loss $\mathcal{L}$
    Every C steps, copy weights from Q to $\hat{Q}$
  end
end

Figure 1.2: DQN algorithm

Chapter 2

Rainbow Agents Components

2.1 Double Q-Learning in the Rainbow Framework

Double Q-learning is among the 6 extensions to Deep Q-learning that were mentioned in the article.

It is is a key component of the Rainbow agent that addresses the overestimation bias common in standard Q-learning algorithms. This enhancement employs two neural networks, Q^A and Q^B, with distinct weights to separately determine the best action and to evaluate that action's value. The update mechanism is as follows:

1. For a state s, an action a is selected based on the output of both Q^A and Q^B.
2. One of the networks, chosen at random, is updated using the observed reward r and the subsequent state s'.
3. When updating Q^A, define a^* as the best action in s' according to Q^A, and adjust $Q^A(s, a)$ based on r and the value provided by $Q^B(s', a^*)$.
4. Conversely, when updating Q^B, determine b^* as per Q^B, and adjust $Q^B(s, a)$ using r and the value from $Q^A(s', b^*)$.
5. This alternating process between Q^A and Q^B for action selection and evaluation continues throughout the learning.

The decoupling of action selection from the update process in Double Q-Learning minimizes overestimation risks. The learning update utilizes the loss function:

$$\text{Loss} = \left(R_{t+1} + \gamma_{t+1} \max_{a'} Q^{\bar{\theta}}(S_{t+1}, a') - Q^{\theta}(S_t, A_t)\right)^2, \tag{2.1}$$

where θ and $\bar{\theta}$ represent the parameters of the active and target networks, respectively. This bidirectional estimation process allows for more precise Q-value approximations and a steadier learning progression.

2.2 Prioritized Experience Replay

In the realm of reinforcement learning, Replay Memory is crucial for agents to remember and utilize past experiences to make future decisions. This is akin to learning from history, which accelerates the learning process. The Rainbow agent employs a sophisticated version of this called Prioritized Experience Replay, a concept introduced by Schaul [15] in 2015. This technique doesn't treat all past experiences equally; it prioritizes them based on their utility for learning.

Consider an agent navigating a maze: Prioritized Experience Replay would focus on recalling the turns that were most effective, rather than all turns equally. This selective memory is akin to concentrating on the most challenging questions when studying for an exam.

The value of each experience is determined by its Temporal Difference (TD) error, akin to a measure of surprise, indicating the informativeness of an experience. High TD-error means the experience was unexpected and potentially valuable for learning.

To ensure a balanced learning process, the Rainbow agent uses a stochastic method that blends the focus on highly informative experiences with random sampling. The probability of an experience being replayed is calculated using:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.2}$$

where $P(i)$ is the probability of selecting experience i, p_i is the priority level of the experience, and α regulates the degree of prioritization. With $\alpha = 0$, all experiences have an equal chance of being selected, whereas higher α values increase the likelihood of selecting experiences with higher TD-errors.

This probabilistic approach ensures that the agent does not overlook less surprising but still informative experiences, maintaining a comprehensive learning experience. The implementation of the algorithm is as follows:

Algorithm 1 Double DQN with Proportional Prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} \leftarrow \emptyset$, $\Delta \leftarrow 0$, $p_1 \leftarrow 1$
Observe S_0 and choose $A_0 \sim \pi_0(S_0)$
for $t = 1$ to T **do**
    Observe S_t, R_t, γ_t
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ **then**
        for $j = 1$ to k **do**
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{target}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla Q(S_{j-1}, A_{j-1})$
        end for
    end if
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta \leftarrow 0$
    From time to time copy weights into target network $\theta_{target} \leftarrow \theta$
end for
Choose action $A_t \sim \pi_0(S_t)$

2.3 Dueling Networks

Dueling Networks is a reinforcement learning strategy that helps agents discern the quality of actions and the value of the situations they encounter. This method, introduced by Wang [16], bifurcates the learning process into two parts: one for appraising the current situation (the state), and another for evaluating the benefits of potential actions in that situation.

Consider the example of a game. Dueling Networks would help determine not just how advantageous a particular move is, but also how favorable the current position on the map is. This dual assessment is achieved by separating the estimation into:

- The state value $V(s)$: a measure of the value of being at state s,
- The action advantage $A(s, a)$: the additional benefit of taking action a at state s.

These two aspects are then synthesized to determine the overall quality of an action in a state, known as the Q-value:

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

Here, θ denotes the parameters of the neural network's convolutional layers processing the state input, while α and β correspond to the parameters of the fully-connected layers for the action advantage and state value, respectively. The term $\frac{1}{|A|} \sum_{a'} A(s, a')$ ensures a zero-centered advantage function, which prevents unwarranted favoring of certain actions.

This segregation enables the agent to evaluate the value of each state without the necessity to consider the outcomes of each possible action. Such an approach can expedite learning by allowing the agent to focus on the broader context of the game, rather than the immediate repercussions of its next action.

In practical scenarios, like being at a crossroads, Dueling Networks would independently assess the potential of each path, devoid of the direct rewards of the steps taken. This assessment modality facilitates a deeper understanding of the environment, which is particularly advantageous in complex decision-making tasks where immediate rewards do not always reflect the true state value.

The Dueling Networks architecture employs a shared lower layer for input processing, followed by two distinct pathways:

- One for estimating the state value $V(s; \theta, \beta)$,
- Another for the action advantage $A(s, a; \theta, \alpha)$.

This dual-pathway architecture is particularly beneficial in contexts where the action choice is not critical unless a specific condition arises, such as the need to avoid obstacles.

Separate learning of the value and advantages through these streams enhances the training efficiency and leads to more dependable action values, thereby considerably improving the agent's performance across various tasks.

This section is inspired by the concepts introduced by Ziyu Wang [16] in their 2016 paper on Dueling Network Architectures for Deep Reinforcement Learning.

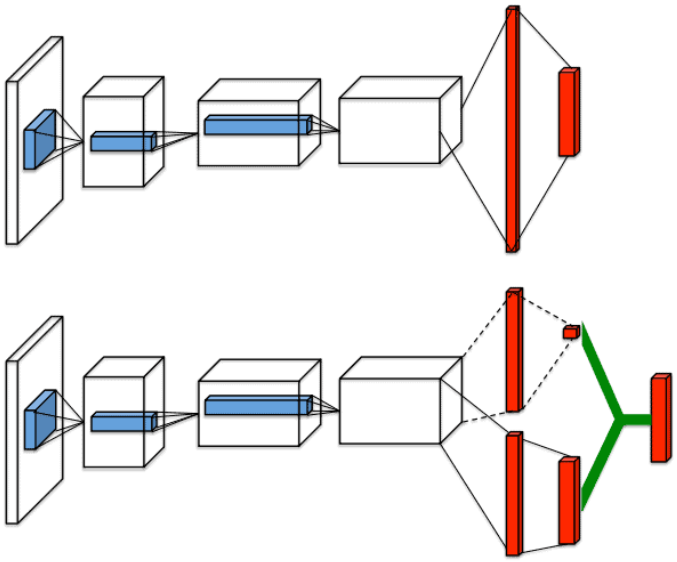


Figure 2.1: Q-network top vs the dueling trained dueling architecture.

2.4 Multi-step Learning

One of the fundamental algorithms in Reinforcement Learning is known as 1-step Q-Learning. It updates the Q-value for a given state-action pair (s_t, a_t) based on the reward received after taking action a_t in state s_t and moving to a new state s_{t+1}:

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

where α is the learning rate, and γ is the discount factor. This method always selects the action that has the highest estimated Q-value in the current state, known as the greedy policy.

However, a more nuanced approach called multi-step Q-Learning extends the horizon of the learning process. Instead of considering only the immediate reward, it looks at a sequence of actions and the cumulative rewards over n steps, which helps to address the short-sightedness of 1-step Q-Learning. This is particularly beneficial in complex environments where the consequences of an action are not immediately apparent. The truncated n-step return, starting from state s_t, is calculated as follows:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}$$

The objective then is to minimize the difference between the predicted Q-values and the n-step returns:

$$\left(R_t^{(n)} + \gamma^n \max_{a'} q_{\bar{\theta}}(s_{t+n}, a') - q_\theta(s_t, a_t)\right)^2$$

By selecting an appropriate n, the algorithm balances the bias-variance trade-off, leading to a more effective and faster learning process.

2.5 Distributional Reinforcement Learning

Distributional RL [8] is a variant of traditional Q-learning that focuses on the entire distribution of potential returns instead of just the expected return. This extension to the standard DQN framework aims to address the inherent limitations by forecasting a full spectrum of possible outcomes, providing a more comprehensive understanding of future rewards.

In standard Q-learning, the goal is often to maximize the expected return, which we express mathematically as:

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

Here, $r(s,a)$ represents the immediate reward expected after taking action a in state s, and γ is the discount factor which quantifies the importance of future rewards.

In contrast, Distributional RL introduces a random variable $Z(s,a)$ that embodies the full return from starting in state s, taking action a, and thereafter adhering to the current policy. The Q-value in this context is the expected value of $Z(s,a)$, denoted as $Q(s,a)=\mathbb{E}[Z(s,a)]$.

Rather than simply minimizing the expected error, Distributional RL seeks to minimize the discrepancy between the predicted and actual return distributions:

$$\sup_{s,a}\operatorname{dist}\left(R(s,a)+\gamma Z(s',a^*),Z(s,a)\right)$$

where $s'\sim p(\cdot|s,a)$ represents the distribution over subsequent states after taking action a in state s.

By incorporating this approach, Distributional RL can account for the variability in returns, leading to potentially more robust and informed decision-making strategies.

2.6 Noisy Nets

Introduced in 2017 [9], Noisy Nets enhance reinforcement learning models by integrating noise into the neural network parameters, thus facilitating exploration. This technique diverges from traditional methods like epsilon-greedy by embedding exploration within the network's architecture itself.

The mechanism behind Noisy Nets involves perturbing the network's weights and biases with noise:

$$\theta=\mu+\sigma\odot\epsilon \tag{2.3}$$

where θ denotes the noisy parameters, μ and σ represent the learnable parameters of the network, and ϵ is a noise vector with zero mean. This approach allows for dynamic adjustment of exploration intensity, directly influenced by the network's learning process.

Noisy Nets can be particularly advantageous in scenarios where the balance between exploration and exploitation is crucial. By incorporating noise into the network's parameters, it autonomously navigates the exploration-exploitation trade-off, adapting its strategy as it learns from the environment.

2.6.1 Resutls Noisy Nets with DQN

The DQN was implemented with the NoisyNets (Blue curve below) and was compared to the classical DQL (Orange curve bellow). The results were impressive, as we can see the reward values increases significantly.

The DQN was implemented with the NoisyNets (Blue curve below) and was compared to the classical DQL (Orange curve bellow). The results were impressive, as we can see the reward values increases significantly.

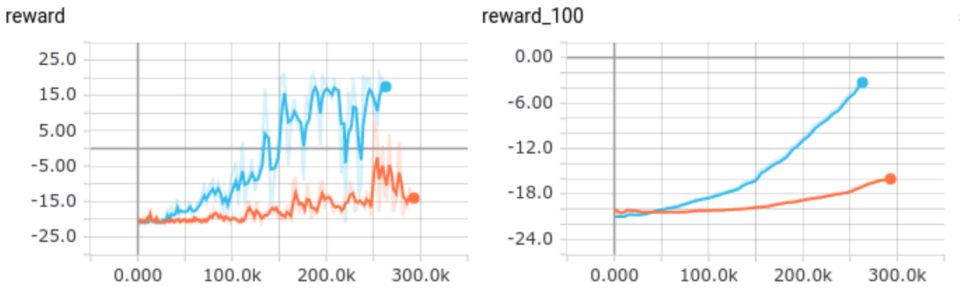


Figure 2.2: Partially-trained DQN on Pong with NoisyNets Blue vs Classical DQN Orange.

Chapter 3

Analysis of Experiments

Having seen the deep Q-learning algorithm, its implementation, how the use of deep learning techniques allowed to improve its performance, and how its different extensions helped tackle different issues related to it, we'll now give the experimental methods and analyze the results of Rainbow whose agent integrates the components of the different extensions in an attempt to obtain optimal performance.

3.1 Experiments

Overview of Experimental Design for the Rainbow Model

This subsection provides an overview of the experimental framework and methodology employed to assess the Rainbow model's performance. The evaluation was conducted on a suite of 57 Atari 2600 games sourced from the arcade learning environment, adhering to the training and assessment guidelines established in [12]. The performance metrics for the agents were standardized across each game, with a 0% score indicative of a random agent's performance and a 100% score benchmarked to that of a human expert. An analysis metric was the enumeration of instances where the agent's performance surpassed that of human experts, providing insight into the model's advancements.

For the training phase, the Rainbow model utilized a deep neural network architecture, processing a training corpus of 200 million frames, and subsequent evaluations were performed on a separate test set comprising 10 million frames.

Further assessments of the optimal agent were executed under two distinct testing conditions:

- *No-ops starts regime:* At the onset of each episode, a random count of no-operation (no-ops) actions are incorporated to inject variability.
- *Human starts regime:* The initiation points for episodes are determined by randomly choosing segments from the beginning of human expert gameplay trajectories.

The variance observed between these testing scenarios serves as an indicator of the agent's potential for overfitting to specific trajectories.

Details regarding the hyperparameters employed are presented in Figure 3.1, offering a comprehensive view of the experimental setup and the quantitative benchmarks used to evaluate the Rainbow model's efficacy.

Parameter	Value
Min history to start learning	80K frames
Adam learning rate	0.0000625
Exploration ϵ	0.0
Noisy Nets σ_0	0.5
Target Network Period	32K frames
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	$[-10, 10]$

Figure 3.1: Rainbow hyper-parameters from [11]

3.2 Cost and Computational Power [1]

We'll first start by discussing the cost and the computational power needed for the Rainbow algorithm.
Rainbow's agent might be praised and might be considered one of the state of the art algorithms in Reinforcement Learning, however, it should be mentioned that it requires a high computational power that is only accessible to a large research laboratory.
In fact, on a Tesla P100 GPU, training an Atari game takes about 5 days. It is also typical to employ at least five independent runs to report results with confidence boundaries. All in all running experiments takes about 34,200 GPU hours or more clearly 1425 days.
It is also important to point out that a Tesla P100 cost about US$6000.which limits the implementation and exploitation of the algorithm in the research field to large laboratories.

3.3 Performance analysis

The Rainbow algorithm was empirically analyzed, comparing it to other state-of-the-art deep reinforcement learning algorithms on several Atari game benchmarks. The analysis includes ablation studies, which investigate the individual contributions of each of the algorithmic components that make up Rainbow, and also provides insights into the interactions between the components.

First, Rainbow is compared to the DQN algorithm with experience replay, and show that Rainbow outperforms DQN on most of the games in the Atari benchmark, and achieves a new state-of-the-art performance on six games. Ablation studies are then performed, which involve systematically removing individual components from Rainbow to investigate their contribution to performance. The ablation studies show that all of the components of Rainbow contribute to its performance, with prioritized experience replay and dueling networks being the most important.

Interactions between the components of Rainbow were also investigated by performing a series of experiments that involve turning different combinations of components on and off. It is found that all of the components interact in complex ways, and that the benefits of the individual components are often dependent on the presence of other components. Rainbow's performance is also robust to changes in hyper-parameters, and that it is able to generalize to new games that were not used during training.

The experiment's analysis provides a detailed investigation of the Rainbow algorithm and its performance on the Atari benchmark, and provides valuable insights into the contribution of its individual components and their interactions.

The results show that Rainbow significantly outperforms all of the baseline algorithms on average, achieving a new state-of-the-art performance on the Atari benchmark. In particular, Rainbow is shown to be more robust and reliable than DQN and its extensions, while also achieving higher scores and more consistent performance across games. The paper also includes

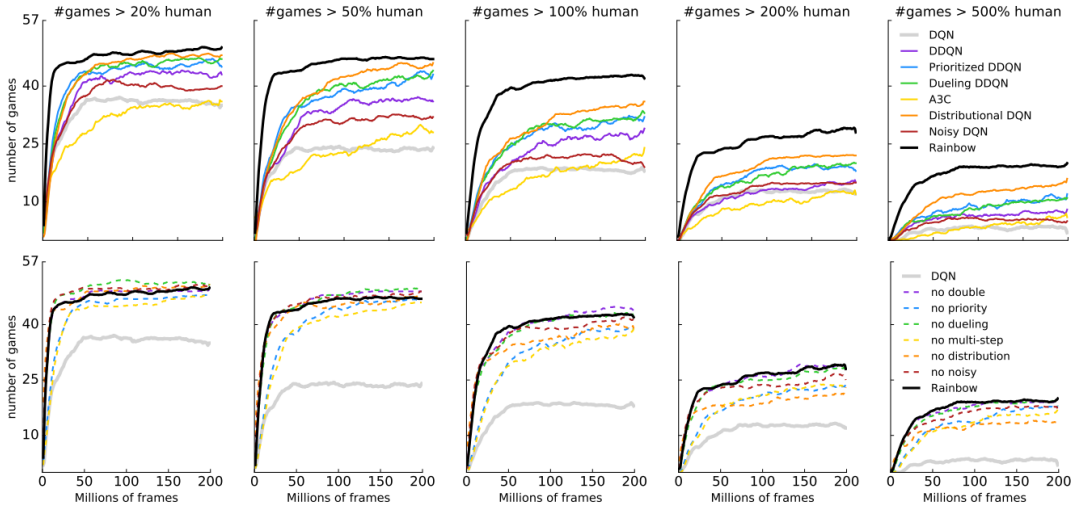


Figure 3.2: First row: Rainbow vs baselines.Second raw: Rainbow vs its ablation [11]

The paper also includes a detailed analysis of the individual components of Rainbow, including a comparison of the different value distributional methods, prioritized experience replay, and multi-step learning. The results show that each of these components contributes to the overall performance of Rainbow and that their combination leads to significant improvements over the individual methods.

Overall, the experiments demonstrate the effectiveness of Rainbow as a state-of-the-art algorithm for deep reinforcement learning on the Atari benchmark. The results suggest that the combination of multiple extensions to the Q-learning algorithm, including value distributional methods, prioritized experience replay, and multi-step learning, can significantly improve the performance and robustness of deep reinforcement learning agents.

Chapter 4

Related Work

It cannot be disputed that the introduction of DQN in general in Reinforcement learning and of Rainbow, in particular, helped significantly in revolutionizing the field. It demonstrated the power of deep learning techniques for complex tasks. These approaches led to significant improvements in the performance, especially of the algorithm on the Atari 2600 benchmark, surpassing all the other performances. The remarkable results found did definitely draw the attention of the research specialists in the fields and were the origin of the important increase in research papers related to that domain.

Furthermore, the integrated agents, as we have seen before, did not include all possible components that could potentially enhance their performance. There is always an area for further research to investigate whether adding or changing certain components could lead to better results. This reinforces the idea of research paper growth in that field.

Hierarchical RL is a good example of an unexplored yet promising method that has shown good results when it was applied to the Atari games. It was in addition to that paired with DQN to give h-DQN which has shown to be successful and achieve better performance by learning a more effective policy through the use of sub-tasks and hierarchical Q-function representation. It has also helped in solving the scaling issues.

Bootstrapped DQN is also one of the components that weren't integrated into Rainbow's agent even though they did exist. It is a technique that explores in a computationally and statistically efficient manner through the use of randomized value functions. Rather than relying on a single Q-function, bootstrapped DQN trains a set of Q-functions, each of which is initialized with different random weights. By sampling from the different Q-functions during training, bootstrapped DQN can efficiently explore the state-action space and reduce the variance in the Q-function estimates.

Intrinsic motivation is another unintegrated technique that allows the agent to explore when rewards are sparse. In many environments, rewards are only given sparsely, making it difficult for the agent to learn an effective policy. Intrinsic motivation provides the agent with additional rewards that are not directly related to the task but encourage exploration and learning. These additional rewards can help the agent discover new state-action pairs that lead to higher rewards and improve its overall performance.

Count-based exploration is also a technique that was not integrated, despite the fact that it has been shown to give near-optimal performance when applied to small discrete Markov decision processes. It works by keeping track of the number of times each state-action pair has been visited and use this count to guide exploration. When the agent encounters a state-action pair that has been visited infrequently, it gives it a higher priority for exploration, which can lead to the discovery of new, high-reward state-action pairs.

This proves that even if DQN and Rainbow delivered excellent results, there will always be room for improvement either in the performance score or in the computation time by integrating the adequate components.

Chapter 5

Implementation of Rainbow

Integrating the seven components of the Rainbow algorithm into a single agent represents a significant advancement in reinforcement learning, especially for environments like Atari 2600 where a wide range of decision-making skills is required. The components are chosen to address various challenges encountered in reinforcement learning, from exploration-exploitation dilemmas to the stability of learning. Here's a brief overview of each component and considerations for their integration:

1. **DQN (Deep Q-Network)**
2. **Double DQN**
3. **Prioritized Experience Replay**
4. **Dueling Network**
5. **Noisy Network**
6. **Categorical DQN**
7. **N-step Learning**

Integration Challenges and Considerations

- *Noisy Network* $\leftrightarrow$ *Dueling Network*: Integrating these two can be straightforward, as Noisy Networks primarily modify the exploration process, while Dueling Networks change the architecture. However, care must be taken to apply noise correctly to both the value and advantage streams.
- *Dueling Network* $\leftrightarrow$ *Categorical DQN*: The main challenge here is to maintain the distributional perspective of rewards in both the value and advantage streams. Implementing the Dueling Network architecture within the framework of Categorical DQN requires careful separation of the distributional estimates of value and advantage.
- *Categorical DQN* $\leftrightarrow$ *Double DQN*: Combining these involves using Double DQN's action selection strategy within the distributional framework of Categorical DQN. The key is to select actions using the current network but to evaluate their distributional value using the target network.

5.1 Experience Replay

5.1.1 ReplayBuffer

The `ReplayBuffer` class is a foundational implementation of an experience replay buffer, pivotal for the training of reinforcement learning (RL) algorithms. It archives the experiences of an agent, encompassing observations, actions, rewards, subsequent observations, and termination signals, which are instrumental for the agent's learning trajectory.

Key Features

- **Numpy Implementation**: Employs Numpy arrays to effectively manage and store experiences.
- **N-step Learning Support**: Integrates logic for N-step learning, accounting for rewards from several future steps when refreshing the agent's knowledge base.
- **Experience Sampling**: Enables random sampling of experiences from the buffer, promoting diverse training samples for the RL agent.

Operations

- **Initialization**: Prepares the buffer with specified dimensions for observations, buffer size, batch size for sampling, N-step count, and the discount factor (gamma).
- **Storing Experiences**: Manages the storage of individual experiences and the handling of N-step transitions, facilitating the accumulation of rewards and adjustment of observations and done flags.
- **Batch Sampling**: Permits the random sampling of a batch of experiences, essential for training the RL model with a variety of experiences.

5.1.2 PrioritizedReplayBuffer

Building upon `ReplayBuffer`, the `PrioritizedReplayBuffer` introduces priority-based sampling of experiences, significantly enhancing the efficiency of learning by concentrating more on pivotal experiences.

Enhancements

- **Priority-Based Sampling**: Implements a strategy to assign and refresh priorities to experiences, with the sampling process skewed towards experiences of higher importance.
- **Efficient Priority Management**: Leverages a sum segment tree and a min segment tree for priority management and efficient sampling of experiences based on their significance.

- **Importance Sampling Weights**: Computes weights for the sampled experiences to amend the bias introduced by prioritized sampling, utilizing a beta parameter for bias adjustment.

Priority Handling

- **Initialization**: Beyond the foundational initialization, it configures the alpha parameter to modulate the priority level of experiences.
- **Priority Updates**: Facilitates the updating of experience priorities based on their temporal-difference (TD) errors, augmenting the emphasis on more informative experiences.
- **Weighted Experience Sampling**: Supports sampling based on experience priority, in conjunction with calculating and applying importance sampling weights to rectify sampling bias.

These implementations of replay buffers are crucial for the proficient training of RL agents, with the prioritized replay buffer presenting a refined approach to concentrate training on more critical experiences.

5.2 Implementation of the Rainbow Agent

The Rainbow Agent represents a significant evolution in reinforcement learning, integrating multiple advancements over the traditional DQN algorithm to form a comprehensive and robust framework. This section details the implementation of the `DQNAgent` class, which incorporates key techniques such as Dueling Networks, Noisy Networks, Categorical DQN, N-step Learning, and Prioritized Experience Replay (PER).

5.2.1 Core Methods and Attributes

- `select_action`: Employs an ϵ-greedy strategy or Noisy Networks for action selection based on the current state.
- `step`: Executes the chosen action, managing frame stacking and reward clipping, and returns the new state, reward, and done flag.
- `compute_dqn_loss`: Computes the loss using either standard DQN or Categorical DQN approaches.
- `update_model`: Applies the computed loss to update the model, including gradient clipping.
- `target_hard_update`: Synchronizes the target network weights with the online network at specified intervals.
- `train` and `test`: Facilitate the training loop and performance evaluation, respectively.

- `_plot`: Visualizes training progress with respect to scores and losses.

5.2.2 Advanced Techniques Integration

The integration of advanced techniques addresses various reinforcement learning challenges:

- Combining **Categorical DQN with Double DQN** mitigates overestimation bias in value assessment.
- **Prioritized Experience Replay** improves sample efficiency by focusing on informative experiences.
- **N-Step Learning** enhances learning speed and depth by considering multiple future steps.
- **Noisy Networks** provide a dynamic exploration-exploitation balance without relying on external strategies.
- The **Dueling Network Architecture** offers a refined approach for state value and action advantage estimation.

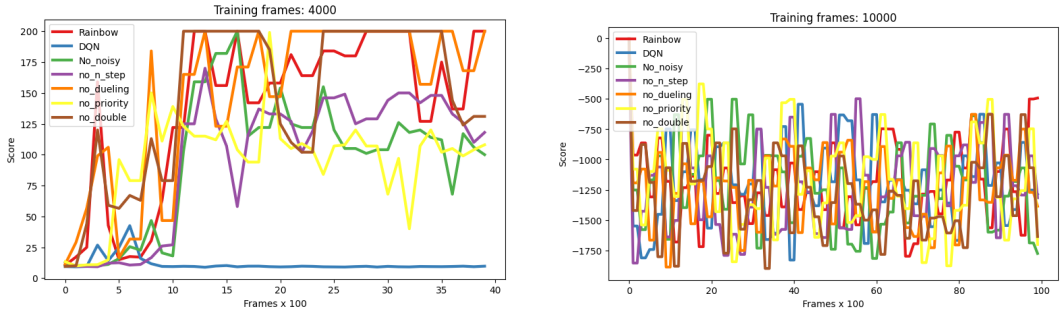
5.2.3 Configuration and Flexibility

The `DQNAgent` class's constructor is designed for extensive customization, allowing adjustments to the replay buffer size, batch size, learning rate, and the inclusion or exclusion of advanced features based on the specific requirements of the task at hand. This design provides a flexible framework suitable for a wide array of environments, challenges, and research pursuits, marking the Rainbow Agent as a potent tool in the field of reinforcement learning.

5.3 Results of the Implementation

In order to assess the impact of the Rainbow algorithm and compare its performance with other agents, we implemented the algorithm and tested it in both discrete and continuous environments, specifically "CartPole" and "Pendulum", respectively. These environments are readily available through the OpenAI Gym library. The **CartPole** environment is a benchmark task in reinforcement learning, where the objective is to balance a pole on a cart. The system's state is represented by the cart position x, cart velocity v, pole angle θ, and pole angular velocity $\dot{\theta}$. The dynamics are controlled by applying a force of +1 or -1 to the cart. The goal is to prevent the pole from falling over.

The **Pendulum** environment tasks an agent with swinging up and stabilizing a pendulum in the upright position. The state of the pendulum is described by its angle θ and angular velocity $\dot{\theta}$. The equation of motion is given by $\ddot{\theta} = \frac{g}{l}\sin(\theta) - \alpha\dot{\theta} + \tau$, where g is the gravitational acceleration, l is the length of the pendulum, α is the damping coefficient, and τ is the applied torque. The challenge is in applying the correct τ to achieve and maintain the upright position.



(a) Cartpole Environement

(b) Pendulum Environment

Figure 5.1: A comparative performance analysis of the Rainbow agent against baseline models in the CartPole environments.

The application of the Rainbow agent demonstrates superior performance and efficiency compared to agents that do not utilize the full spectrum of the Rainbow components. This implementation allowed us to evaluate the effectiveness of the Rainbow agent, enabling us to conduct our own benchmark inspired by the original paper.The full implementation of the code was inspired by [14] and it is available in our github [7]

Conclusion

This document has provided a comprehensive overview of the Rainbow algorithm, which integrates a diverse range of techniques including DQN, DDQN, Multi-Step Learning, Prioritized Experience Replay, Dueling Q-Network, Distributional RL, and Noisy Networks. Each technique has been meticulously described, illustrating how the Rainbow algorithm excels beyond conventional benchmarks. Notably, prioritized replay and multi-step learning are identified as key contributors to its superior performance. The synthesis of these methodologies culminates in the creation of an optimal agent that sets new standards in the domain of deep reinforcement learning, showcasing the Rainbow agent's extraordinary achievements beyond previous approaches.

In the landmark paper **Revisiting Rainbow: Promoting More Insightful and Inclusive Deep Reinforcement Learning Research 2020**, authors Johan S. Obando-Ceron and Pablo Samuel Castro [13] shed new light on the foundational elements of the Rainbow algorithm. Their research explores the dynamic interplay among the algorithmic components within the Rainbow agent and its network structure, proposing enhancements to the Rainbow algorithm. A pivotal insight from their analysis is the observation that agent performance may suffer when distributional RL is incorporated in isolation with DQN. To mitigate this, they suggest a strategic combination with other algorithmic elements in classical control scenarios or pairing it with a CNN for the MinAtar games.

Bibliography

[1] https://psc-g.github.io/posts/research/rl/revisiting_rainbow/#the-cost-of-rainbow.

[2] https://medium.com/sciforce/reinforcement-learning-and-\asynchronous-actor-critic-agent-a3c-algorithm-explained-f0f3146a14ab.

[3] https://medium.com/swlh/states-observation-and-action\-spaces-in-reinforcement-learning-569a30a8d2a1.

[4] https://towardsdatascience.com/double-deep-q-networks-905dd8325412.

[5] https://mtomassoli.github.io/2017/12/08/distributional_rl/.

[6] https://shmuma.medium.com/summary-noisy-\networks-for-exploration-c8ba6e2759c7.

[7] Rl-project. https://github.com/mehdi-byte/RL_project.

[8] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[9] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration, 2019.

[10] J. Fernando Hernandez-Garcia and Richard S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target, 2019.

[11] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

[12] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

[13] Johan S. Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research, 2021.

[14] Curt Park. rainbow-is-all-you-need. https://github.com/Curt-Park/rainbow-is-all-you-need.

[15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[16] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2016.