Reinforcement Learning With Intrinsic Rewards on Complex Environments

Author
Nathan Grabaskas ngrabaskas@gmail.com
Zhizhen Wang cwang42@outlook.com

Abstract

Painting is a creative art, and it will take human beings years to learn how to draw. In this paper, we propose an approach to train an artificial intelligent agent using reinforcement learning to draw on a two-dimensional grid. In the training process, we built grid environments with obstacles and challenges that resemble abstract art and then place the agent in different environments to reach the goal. We propose using intrinsic rewards based on the state of the model to stimulate the agent's exploration desire and to increase flexibility in complex environments. In the final step, we designed a rendering pipeline to translate the agent's movement in the training process into a painting. Our results show the intrinsic reward method does increase the agent's ability to learn on environments of medium complexity. The rendering pipeline was only evaluated in one survey and needs additional rounds of evaluation for a concrete conclusion.

1 Introduction

The problem which kicked off this paper is teaching an artificial intelligent agent how to draw. However, the scope of work is mostly focused on agent rewards. Drawing and creative art are a critical part of human civilisation and culture. To learn how to draw would take years of learning and practising for humans. Hence we want to explore the idea of training an artificial intelligence agent and visualizing the training to produce interesting and abstract pieces of art.

The first part of this project focuses on agents learning to explore an environment, avoiding obstacles, and reaching a goal. And secondly, to investigate the abstract artwork an agent can generate during the training process. Based on the goal, the agent is measured against both quantitative and qualitative metrics.

For agents learning to explore, we start by defining a 2D environment (canvas) to simulate the painting, and each training canvas is initialised with a positive rewarding grid (goal) and hazard zones (immediately ends the episode) (Chevalier-Boisvert et al., 2018). The initial parameters for generating the environment are based on abstract art paintings. The next step is to allow the agent to explore the environment while avoiding danger zones and minimize frames taken to reach the goal. We seek a method of rewarding the agent simply for learning, attempting to give an intrinsic desire to explore the environment (Raileanu and Rocktäschel, 2020; Maruan Al-Shedivat, 2018). The intrinsic reward method proves successful in medium complexity environments.

Second part of the experiment is to generate visualizations from the agent's training and compare which methods produce more interesting art. This was inspired by Luo's dissertation in artistic applications for reinforcement learning (Luo, 2020). In order to turn the agent training into artwork, a separate pipeline is setup. This pipeline renders images from the agent training, applies open source painting effects, and sends to human raters for evaluation. The second part is only lightly explored in this paper and further experiments are required.

The paper is organized into the following sections: related works, data used, methods, metrics, results and discussion, and finally conclusion and future steps.

2 Related Work

Similar work has been done by Zhewei, Wen and Shuchange (Zhewei Huang, 2019) where they trained an AI agent that can paint on a canvas to generate a painting that is similar to the target image. Apart from training for a reward policy for

the agent, the team also proposed an approach to decompose the target painting into hundreds of strokes in sequence in a grid. In the end of the project, the agent is trained to be general enough to handle multiple types of images (including digits, handwritten, streetview, human portains and etc.). Another work related to artwork generation is done by Ning and his team (Ning Xie and Sugiyama, 2015). The team is focusing on a particular type of painting, stroke drawing. They applied inverse reinforcement learning to learn the reward function from the training data.

Exploration in sparse reward environments remains a key challenge of reinforcement learning (Raileanu and Rocktäschel, 2020). They propose a novel type of intrinsic reward which encourages the agent to take actions that lead to significant changes in its learned state representation. They evaluate their method on multiple challenging procedurally-generated tasks in MiniGrid. This approach is more sample efficient than existing exploration methods and the intrinsic reward does not diminish during the course of training and it rewards the agent substantially more for interacting with objects that it can control. Rewarding Impact-Driven Exploration (RIDE) is computed as the L2-norm of the difference in the learned state representation between consecutive states. However, to ensure that the agent does not go back and forth they discount RIDE by the number of times that state has been visited. The parameters used to learn the intrinsic reward signal are used only to determine the exploration bonus and never part of the agent's policy.

3 Data

The dataset set used in this project serves one primary purpose: creating the agent's training environment.

In Zhewei's paper(Zhewei Huang, 2019), their team model the agent's painting process as sequential decision-making tasks. During training, the agent was rewarded based on comparing the current status on the canvas to the target painting.

In this project, we use a different strategy. Instead of focusing on decomposing the painting into a sequential environment, we pre-build the training environment based on the chosen abstract art from Kaggle (Surma, 2019). An example can be seen in Fig. 1.

The Kaggle dataset consists of 8,145 abstract art paintings with a resolution of 512*512. To maximize the experiment result, we choose a few arts with precise edges and repeatable patterns. Then a method has been build to transform the raw artwork into a 2D grid representation. The team uses a preprocessing script for resizing, aligning, setting a threshold for detecting the edge, blurring, and defining the hazard zones. Then the interim images are used as examples to create MiniGrid environments.

Figure 1: This images shows a piece of abstract art chosen from Surma's Kaggle Open dataset.

Minimalist GridWorld (MiniGrid) environment seeks to minimize software dependencies, be easy to extend, and deliver good performance for faster training (Chevalier-Boisvert et al., 2018). The environment comes with the existing object types wall, floor, lava, door, key, ball, box and goal. This gives the agent diverse, generated environments with tasks such as getting the key to unlock the door, hazards like lava to be avoided, and the purpose of getting to the goal. This provides a framework in which to execute our agent training.

The team uses customized objects to echo the artworks' characteristics (for example, different color schema, shape, patterns, and transparency). Later, the agent will explore procedurally generated environments and iterate on various policies to achieve maximum rewards. An example MiniGrid environment can be found in Fig. 2.

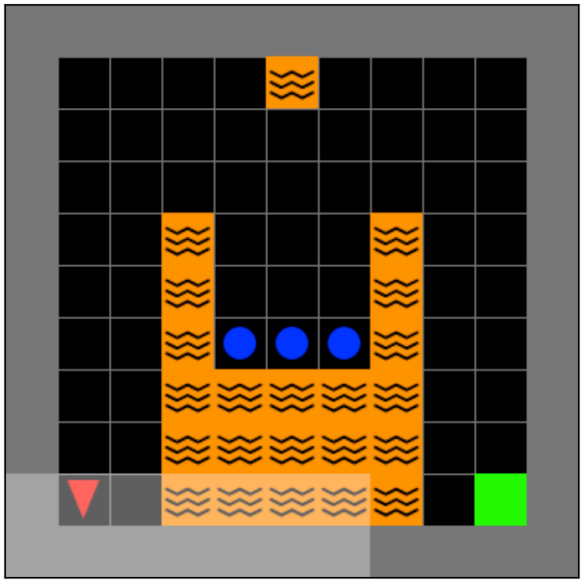

Figure 2: This images shows a piece of abstract art converted to a grid environment for the agent to explore. The orange tiles represent lava and end the episode if the agent touches them and green represents the goal the agent is striving to reach. Only if the goal is reached is an extrinsic reward given.

4 Methods

In this section we discuss some background in how reinforcement learning works, existing learning algorithms, our evaluated intrinsic reward method, and the model architecture used.

4.1 Reinforcement Learning Background

We use the common reinforcement learning setup where an agent interacts with an environment over frames or discrete time steps. At each frame or s_t, the agent receives a current state representation and selects an action a_t from a set of possible actions A. The policy π is a mapping from the given state s_t to an action a_t. Afterwards, the agent receives the next state s_{t+1} and a reward if any. This process continues until a terminal state is reached (reaching a goal or frame limit). The reward at the end is the sum of rewards over each time step with a discount factor λ to decay the reward. The goal of the agent is to maximize the expected end reward from each current state (Volodymyr Mnih and Kavukcuoglu, 2019).

4.2 Model Policies

Advantage Actor-Critic (A2C) - In this algorithm the advantage function captures how much better an action is compared to others at a given state, while the value function captures how good it is to be at this state. This way the evaluation of an action is based not only on how good the action is, but also how much better it can be. The benefit of the advantage actor-critic function is that it reduces the high variance of policy networks and stabilizes the model (Sutton and Barto, 1998; Degris et al., 2012).

Actor-Critic algorithm is a hybrid mechanism that combines the value optimization and policy optimization. More specifically, the Actor-Critic combines the Q-learning and PG (Policy Gradient) algorithms (Volodymyr Mnih and Riedmiller, 2013). At a high level, the resulting algorithm involves a loop that alternates between:

- **Actor:** a Policy Gradient algorithm that decides on an action to take
- **Critic:** off policy reinforcement learning algorithm that critiques the action that the actor selected, providing feedback on how to adjust. It can take advantage of efficiency tricks in Q-learning, such as memory replay.

A2C maintains a policy defined as

$$\pi(s_t; \theta)$$

and an estimate of the value function.

$$V(s_t; \theta_v)$$

A2C operates in the forward view and uses the same mix of time step returns to update both the policy π and the value function V^π . The policy π and the value function V^π are updated after every t_{max} actions. The update performed by the algorithm is

$$update_{loss} = \pi_{loss} - H_\xi * H + V_{loss\xi} * V_{loss} \quad (1)$$

Where ξ is the coefficient (we use 0.01 and 0.5 respectively for entropy H and V) (Ziyu Wang and de Freitas, 2016). We use two fully-connected neural networks for the actor and critic. The actor component outputs the agent action. And the critic outputs the value function estimate. This value function estimate replaces the reward function in policy gradient that calculates the rewards only at the end of the episode.

A2C Intrinsic Rewards - After backward propagation of the model using the $update_{loss}$, there is a second round of backward propagation using:

$$update_{loss} = L_2Norm||\phi_{s_{t+1}} - \phi_{s_t}|| \quad (2)$$

where ϕ represents the weights for all layers in the CNN (Raileanu and Rocktäschel, 2020). This intrinsic reward is to encourage the agent to take actions that lead to significant changes in its learned state representation. Attempting to mimic the sensation of learning or exploring.

4.3 Model Inputs

Input to the model (See Table-1) is the agent's view of the grid environment. For all experiments the view distance is set to 11, therefore the input is an array of size (3,11,11). Each tile is encoded as a 3 dimensional tuple: (OBJECT, COLOR, STATE) (Chevalier-Boisvert et al., 2018).

4.4 Model Architecture

To implement A2C with Deep Learning, 3 components are needed (See Fig. 3). Component 1 is a Convolution Neural Network (CNN) which takes then agent observation and converts this to an embedding of size 576. The other 2 components are the Actor and Critic and have the same architecture. They both take the embedding input from the CNN, have a layer of 64 neurons (Lucas Willems and Chevalier-Boisvert, 2018). The Actor outputs one the possible agent actions, while the Critic outputs an estimate of the value function. These components are optimized using RMSProp (Tieleman and Hinton, 2012).

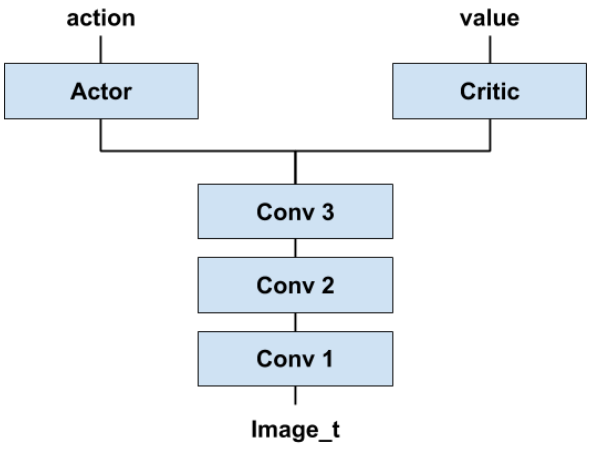

Figure 3: This figure shows a diagram of the model architecture. The CNN contains three layers which take the agents view as input and output an embedding of size 576. Each Actor/Critic component take this as input and outputs an action(size=7) and value function estimate (size=1) respectively.

5 Metrics

We outline both the quantitative and qualitative metrics we use to evaluate our experiments.

5.1 Quantitative Agent Comparison

Each episode has the grid setup and the agent is given a reward for reaching the goal, this reward decays for each frame of the episode. No reward is given if the frame limit is reached or the agent touches the hazard. Quantitative metrics are used to compare agents trained with different algorithms in the same environment. Each agent is evaluated on 100 episodes of the procedurally generated environment. The baseline is the normal A2C algorithm trained and evaluated on each grid environment, both basic and art inspired. We compare the intrinsic reward variation against A2C on all environments and discuss successes and shortcomings.

We chose two metrics to evaluate the algorithms. Mean Reward is the first metric and Max Reward is the second metric, this is across all episodes played, the highest reward the agent received. We don't discuss these further, but there are two other metrics one could consider, mean frames and max frames. The trick with frames as a metric is lower is not always better. Given an environment with hazards, higher may be better, because the agent has learned to avoid the obstacles.

5.2 Qualitative Comparison

After training the agents we sent each trained agent through 50 episodes to evaluate for each environment. For each episode we create a time lapse graphic representation of the environment. The agent's movements are shown throughout the episode with brighter spots representing where the agent spent more time. Next, we used OpenCV xphoto oil painting effect (Bradski, 2000) to take this image and create an abstract representation. These images were combined to to create an A/B comparison (See Fig. 4).

Comparisons were between the same environments and agents with the same amount of frames for training. Again using the normal A2C agent output as the baseline and our intrinsic model as the comparison. The ordering was alternated so that one source was not always on the same side. And 100 comparison images were sampled from the available 350 to be sent off for rating. These

Dimension	Represented Values
STATE	open, closed, and locked
OBJECT	wall, floor, lava, door, key, ball, box, and goal
COLOR	red, green, blue, purple, yellow, and grey

Table 1: Reinforcement model input by dimension and possible values for each dimension.

comparisons were placed in front of human raters using Mechanical Turk (MTurk) and asked which one they find more interesting.

The major challenge with the qualitative comparison is the subjectivity of what each rater finds interesting. Image ordering is varied to help reduce bias, if the rater learns to prefer one method over the other and expects the better image to be on a particular side. Each sample is placed before three raters and their ratings combined so that we are not relying on a single observer, this also helps to reduce bias. The question and possible answers is shown as an example below:

- **Question** - With an A/B comparison of images, "Which Image of do you find more interesting?"
- **Answer** - A, B, Same

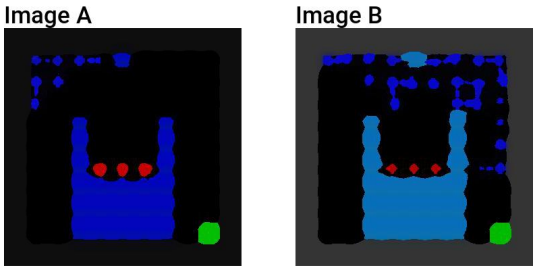

Figure 4: This figure shows two images rendered using the agent environment time lapse along with OpenCV Oil Painting effect. Image A is from Normal A2C and Image B is from A2C Intrinsic. In this example the agent on the right did more exploration and created a visualization with more effect.

6 Results and Discussion

In this section we cover the experiment setup, reporting quantitative and qualitative results. We also discuss what the results show, comparing variations of environments and training algorithms. As well as a piece by piece analysis of the training visualization pipeline.

6.1 Experiment Setup

We set up the base repository to build up the training environment. There are three repositories we combined to lay the foundation for our experiments.

1. Gym-MiniGrid (Chevalier-Boisvert et al., 2018) - contains the framework for the grid environment and obstacles/interactions the agent is able to perform built on top of the popular OpenAI Gym (Greg Brockman and Zaremba, 2018).

2. Torch-AC (Willems and Karra, 2018) - contains the base algorithm implementations for PPO (John Schulman and Klimov, 2017) and A2C (Volodymyr Mnih and Kavukcuoglu, 2019).

3. RL-Starter-Files (Lucas Willems and Chevalier-Boisvert, 2018) - contains the framework for training the agent on each environment, storing model states, and evaluating agent performance.

In the experiment, we first set up the Gym-MiniGrid environment as the base environment, which provides the playground for the agents to explore. Then Actor-Critic (A2C) algorithm was implemented using the Torch-AC repo. We also enhance the default AC algorithm by implementing intrinsic reward when updating the agent's parameters during each batch.

A list of abstract art inspired environments were implemented in Gym-MiniGrid to train the agent. In each environment, the obstacles objects were customized generated based on either precise rules or constraint policy. In some of the complicated grid world, the agent needs to complete a list of tasks (picking up a colored key or going through a linked door) to reach the goal.

The agent was trained with RL Starter repository with the A2C and A2C_intrinsic on 1.5 million and 3 million frames. The agent's convergence rate is highly correlated with the complex-

ity and safety of the environment (Biase and Namgaudis, 2018). And we found out not all agents could find the path to reach the goal at the end of the experiments.

The team generates and selects two types of results for quantitative and qualitative evaluation. For quantitative comparison, an average of return means and an average return maximum of the agent's rewards were picked to show the general performance in each setup. And for qualitative questions, we first rendered an image using the time-lapsed agent's movement on the grid and enhanced the raw image with visual effect before sending them for review.

6.2 Quantitative Results

The normal agent without intrinsic rewards did better on our baseline environments DoorKey, FourRoom, and MultiRoom. These environments are less complex, with FourRoom and MultiRoom requiring navigation only with no hazards. These did not require exploration or curiosity on the part of the agent, eliminating any advantage the intrinsic agent had. While both algorithms were able to adapt to the environment, giving a reward for "learning" did not benefit the intrinsic agent and showed a 15.69% decrease in rewards received by the agent (see Table-2).

PaintingS11N5 and PaintingV1S11N5 are complex, procedurally generated environments which neither algorithm was able to train well on. Since rewards were only given for reaching the goal, the agent is not given a reward for making progress within the environment. These maze-like environments change each episode, and make it difficult for the agent to learn to navigate. In the future we could try first training the agent on a smaller version of the maze and then expand the size of the maze as the agent has learned to do some navigation. This is a common problem and one we tried to overcome with intrinsic rewards. Agents learning to navigate in a sparse reward and complex environment. Further experiments are still needed here.

PaintingS9 and PaintingS11 are less complex and singleton environments, but they still require some exploration to move around the lava and find the goal. In both of these environments the A2C model with intrinsic reward agent was able to explore the environment and learn to reach the goal in fewer frames but also more effectively than the normal A2C.

6.3 Qualitative Results

For our qualitative analysis, each image combination was shown to three raters and we use the average across all three raters. A normal image vote was given a 0.0 for each time it was selected as more interesting and an intrinsic image vote was given a 1.0 for each time it was selected as more interesting. A vote for "they are the same" was given a 0.5. The average was taken across all 300 raters and this gave the 0.503. Threshold based was only count votes where a confidence interval of 2/3 votes was needed to count. The means 97/100 images received two out of three votes with only 3 images receiving a neutral score. Normal image received 48 votes and intrinsic images received 49 votes.

The 100 samples sent off for rating came back almost completely even. This shows there is a lot of room to improve both our rendering pipeline and the agent rewards to prioritize "good art" creation. At each step in the rendering process there is the potential for improvement. Time lapse of the environment is only one option, we could also look at a time lapse of the agent's view, a time lapse of the layers of the CNN or Actor/Critic components. Instead of a time laps we could attach a theoretical brush to the agent and render brush strokes based on the agents movement. If the critic could evaluate how good the final artwork is this could lead to more interesting artwork.

Additionally, instead of the oil painting effect, we could try watercolor effect, brush stroke effect, cartoon effect, sketch effect. The color scheme could be changed to give more variety or less for a monochromatic final image. Each of these steps could be evaluated using human raters to determine the best decision at each phase and combine all steps to create even more interesting final artwork.

Asking a human rater which image they find more interesting is very subjective, and this was clearly shown in our survey results. Perhaps if there was a target painting, then we could ask which model version was able to more closely match the target. Or even more pointed questions, such as which image is more colorful, which one more closely matches a specific style, etc. Since human ratings on art are very subjective, it would also be very difficult to compare results

Environment	A2C - μr_t	A2C - *max* r_t	A2C Intrinsic - μr_t	A2C Intrinsic - *max* r_t
DoorKey-8x8	0.0135	0.1112	0.0079	0.0788
FourRooms	0.2976	0.7379	0.2178	0.6675
MultiRoom-N2-S4	0.6675	0.8397	0.6370	0.8500
PaintingS11N5Env	0.0000	0.0000	2.00E-06	0.0190
PaintingV1S11N5Env	0.0070	0.0800	0.0072	0.0721
PaintingS9Env	0.0012	0.0786	0.2685	0.9474
PaintingS11Env	0.0012	0.0585	0.0024	0.0791

Table 2: All of these agents were trained for 1.5M Frames on each environment. The means and max are shown for the entire training period. This method of evaluation was chosen to reduce variance in agent performance based on only a small number of final episode.

from multiple rounds of evaluations, since each group would be different. We need to use the same group or a group with a similar to makeup to have confidence between evaluation rounds.

7 Conclusion and Future Work

Our work set out to give reinforcement learning agents a reward beyond just the explicit reward of reaching a goal. And to visualize this training as a way to generate interesting abstract artwork. We called this an intrinsic reward and sought to incentivize state changes in the model to promote learning. We evaluated a method of determining the difference between two states using the L2 norm and adding this to the loss in order to perform a second round of backward propagation. We evaluated this change on the Advantage Actor-Critic algorithm and used Gym-MiniGrid for the environment. We looked at several baseline environments: one involved a simple task of using a key to open a door and the others involved navigating through a limited number of rooms to reach the goal. Additionally, we created 4 environments inspired by abstract art, two complex procedurally generated mazes with hazards, and two singleton environments with hazards.

Our experiments show the normal A2C algorithm was able to learn the baseline environments more effectively, achieving 15.69% higher average rewards over the intrinsic variant. The complex procedurally generated environments were too difficult for either of our variants tested to learn. The A2C algorithm with intrinsic rewards was able to learn the singleton environments more effectively, mastering the PaintingS9 where the normal A2C was unable to achieve even small rewards.

From our qualitative analysis, it is clear our process to create artwork from the agent's actions was flawed at multiple stages. Our rewards need to better incentivize art creation actions, our rendering pipeline has many untested assumptions which need to be verified, and our evaluation needs more specific questions with a similar group of raters for each iteration.

The intrinsic rewards did show positive results. We need further training and analysis across more environments and for longer training periods. A stepped approach where an agent is trained on a simple environment and then tuned on a more complex environment, could help the agent learn faster than from starting with random weights on a complex environment. Additional tuning of agent view distance may also help the agent explore.

A Supplemental Material

1. Code Repo:
   https://github.com/catwang42/xcs229ii
2. Environment Repo:
   https://github.com/ngrabaskas/gym-minigrid
3. Evaluation Repo:
   https://github.com/ngrabaskas/rl-starter-files

References

Andres De Biase and Mantas Namgaudis. 2018. Creating safer reward functions for reinforcement learning agents in the gridworld. *University of Gothenburg Chalmers University of Technology. Sweden*.

G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. Minimalistic gridworld environment for openai gym.

T. Degris, P. M. Pilarski, and R. S. Sutton. 2012. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pages 2177–2182.

Ludwig Pettersson Jonas Schneider John Schulman Jie Tang Greg Brockman, Vicki Cheung and Wojciech Zaremba. 2018. Openai gym. *ACL 2016*, arXiv:1606.01540.

Praffula Dhariwal Alec Radford John Schulman, Filip Wolski and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Computer Vision and Pattern Recognition (cs.CV)*, arXiv:1707.06347v2. Version 2.

Dzmitry Bahdanau Lucas Willems, Ye Yuan and Maxime Chevalier-Boisvert. 2018. Rl starter files.

J. Luo. 2020. *Reinforcement Learning for Generative Art*. Ph.D. thesis, UC Santa Barbara.

Ruslan Salakhutdinov Eric Xing Maruan Al-Shedivat, Lisa Lee. 2018. On the complexity of exploration in goal-driven navigation. *Relational Representation Learning Workshop (NIPS 2018)*, arXiv:1811.06889.

Feng Tian Xiaohua Zhang Ning Xie, Tingting Zhao and Masashi Sugiyama. 2015. Stroke-based stylization learning and rendering with inverse reinforcement learning. *IJCAI'15: Proceedings of the 24th International Conference on Artificial Intelligence-July*. Version 2.

Roberta Raileanu and Tim Rocktäschel. 2020. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *Machine Learning (cs.LG)*, arXiv:2002.12292v2. Version 2.

Greg (Grzegorz) Surma. 2019. Abstract art images.

Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement learning: an introduction. *MIT Press*.

T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Volodymyr Mnih, Koray Kavukcuoglu and Martin Riedmiller. 2013. *NIPS Deep Learning Workshop 2013*, arXiv:1312.5602v1.

Mehdi Mirza Alex Graves Timothy Lillicrap Tim Harley David Silver Volodymyr Mnih, Adria Puigdomenech Badia and Koray Kavukcuoglu. 2019. Asynchronous methods for deep reinforcement learning. *Computer Vision and Pattern Recognition (cs.CV)*, arXiv:1903.04411v3. Version 3.

Lucas Willems and Kiran Karra. 2018. Pytorch actor-critic deep reinforcement learning algorithms: A2c and ppo.

Shuchang Zhou Zhewei Huang, Wen Heng. 2019. Learning to paint with model-based deep reinforcement learning. *Computer Vision and Pattern Recognition (cs.CV)*, arXiv:1903.04411v3.

Nicolas Heess Volodymyr Mnih Remi Munos Koray Kavukcuoglu Ziyu Wang, Victor Bapst and Nando de Freitas. 2016. Sample efficient actor-critic with experience replay. *ICLR 2017*, rXiv:1611.01224v2. Version 2.