Imagination-Augmented Agents for Deep Reinforcement Learning

Angela Denninger

Munich, 18.10.2018

Dynamic Vision and Learning Group

Supervisor: Prof. Laura Leal-Taixé
Advisor: Tim Meinhardt

Contents

1 Introduction

Deep reinforcement learning is an active research area in artificial intelligence, where a reinforcement learning agent, learns to solve a given problem by interacting with its environment and maximizing the received reward. This technique is also known as model-free reinforcement learning. One drawback of this approach is, that the agent is not able to imagine what will happen in the future, which leads to bad performance in planning intensive tasks. Model-based approaches are using a model of the environment, modeling the transitions between states in the environment, to maximize the received reward. This allows the agent to predict into the future without actually performing an action and to choose the optimal action. However in real world applications it is usually very hard, if even possible, to get a sufficiently good model to solve a given problem.

The paper "Imagination-Augmented Agents for Deep Reinforcement Learning" from Weber et al. [10] combines model-based and model-free reinforcement learning approaches. The imagination-augmented agent (I2A) uses the information the environment model provides, but is also able to ignore defective parts of the environment model. To choose the best possible action, I2A imagines for each possible action how the future will develop, by using a model of the environment, modeling it as a markov decision process. I2A works directly on an observed image of the environment, with no additional information given, except the reward signal.

The goal of this guided research is to analyze the results of Weber et al. in terms of performance, drawbacks and reproducibility. For this we implemented an own imagination-augmented agent and trained it on the same test environment as Weber et al. did.

We will first give a short overview over current reinforcement learning algorithms, followed by a detailed explanation of the imagination-augmented agent algorithm and the used test environments. Then the reproduced results will be presented and compared with the results of the paper. In the last part the drawbacks of the algorithm and possible future work will be discussed.

2 Reinforcement learning

Reinforcement Learning refers to a kind of Machine Learning method in which an agent learns to solve a given task by maximizing the received reward signal, where the agent represents the reinforcement learning algorithm.

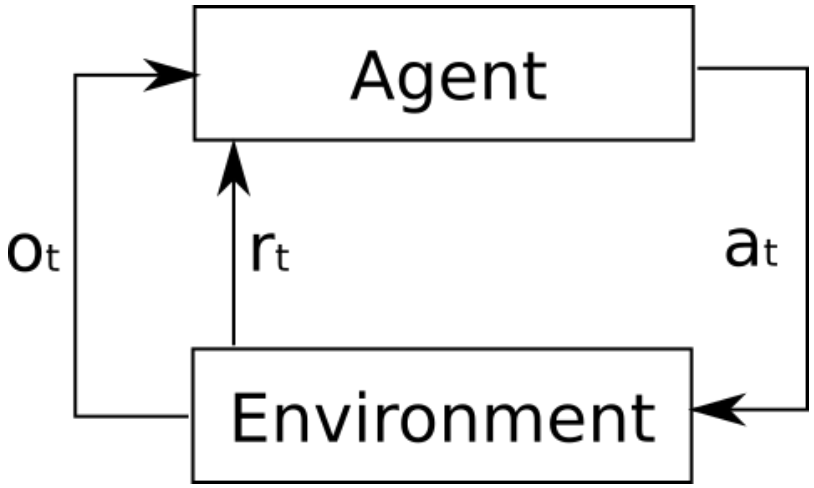


Figure 1: Agent: reinforcement learning algorithm; Environment: object the agent is acting on; a_t: action the agent performes in the environment; o_t: observation (current state) of the environment the agent receives; r_t: reward signal the agent receives from the environment

Figure 1 shows the interaction of the agent with the environment, where the environment refers to the object the agent is acting on. As initial state the agent receives the observation o_t of the environment at time $t = 0$. The observation o_t can be the complete state of the environment, or just a subset of it. The agent then, based on the observation o_t, performs an action a_t in the environment, chosen from a set of possible actions (e.g. moving to the right or moving to the left). After performing the action a_t the agent receives the new observation o_{t+1} of the environment and also the reward signal r_{t+1} which evaluates how good the chosen action was. Based on the new information the agent again chooses an action a_t. The loop continues until the environment sends a termination state.

The goal of the agent is to maximize the received reward, which is done by learning an optimal action choosing function called policy, using the received reward signal. There exist different reinforcement learning algorithms, which all follow the iterative learning algorithm described above, but differ in the update strategy for learning an optimal policy. In the following a short overview over different types of reinforcement learning algorithms will be given.

2.1 Q-learning

Q-learning is a reinforcement learning algorithm introduced by Watkins [9] in 1989. For any finite markov decision process, Q-learning finds a policy that maximizes the expected total reward over all following steps, starting from the current state. To do so Q-Learning learns a Q-table, which contains for each state-action pair (s, a) a Q-value representing the expected future reward for taking the action a in state s. By iteratively updating the Q-value function with the Bellman Equation, the Q-value function will converge to the optimal Q-value function [8].

The Bellman Equation

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r + \gamma Q_\pi(s_{t+1}, a_{t+1})|s_t, a_t] \tag{1}$$

states, that the Q-value for the state-action pair (s_t, a_t) is equal to the expectation of the received reward r, of taking action a in state s, plus the discounted future reward the agent will get by following

the policy π from the next state onwards, where γ refers to the discount factor.

The optimal Q-value, denoted as $Q^*(s_t, a_t)$ can be expressed as:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t] \tag{2}$$

After the agent has converged to the optimal policy $Q^*(s_t, a_t)$ it is able to to choose in each possible state the optimal action.

2.2 Value-based versus policy-based reinforcement learning methods

Value-based methods are learning a value function, which maps state-action pairs directly to values. The best action in a certain state can then be found by taking the action with the biggest value. The policy is therefore the action choosing strategy, if the action is always taken by choosing the action with the biggest value the strategy is called greedy. However if the agent always chooses the greediest action, it will soon get stuck in a local minima and will never be able to find the optimal policy. To find the optimal policy the agent needs to explore as many states as possible. Thats the reason why usually an ϵ-greedy policy is used, which chooses with a probability of ϵ a random action, exploration, and otherwise a greedy action, exploitation. An example of a value-based method is the Q-learning method mentioned in the previous section.

Policy-based methods, in contrast to value-based methods, optimize the policy directly without using a value function. The main problem of policy-based methods is to find a good score function to compute how good a policy is. An example of a policy-based method is reinforcement learning with policy gradient.

2.3 Model-free versus model-based reinforcement learning

Model-free reinforcement learning maps observations of the environment directly to values or actions. In contrast to this, model-based reinforcement learning algorithms are using a model of the environment to simulate the dynamics of the environment. The model knows the transition probability $T(s_{t+1}|s_t, a_t)$ to the next state s_{t+1} given the current state s_t and the current action a_t. By taking this model into account, adverse consequences of trial-and error can be avoided, also the performance of the agent can be increased by increasing the amount of internal simulations. But there are some drawbacks. If the model is imperfect the performance of model-based agents suffers. Also it is not always possible to get an exact transition model or to get a transition model at all, especially in real world applications.

2.4 Deep Q network

In the paper "Human-level control through deep reinforcement learning" Mnih et al. [5] introduce the first reinforcement learning algorithm which uses a neuronal network as a function approximator for the Q-value function. Reinforcement learning algorithms, which are using a neuronal network as function approximator, are also called deep reinforcement learning.

The approximated target values are

$$y = r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-) \tag{3}$$

The network parameters θ_i can than be optimized by minimizing the loss between the optimal target values and the approximated target values. Because the optimal target values are not know a second network with fixed weights θ_i^- is used. The loss function looks as follow:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}[(\mathbb{E}_{s_{t+1}}[y|s, a] - Q(s, a; \theta_i))^2] \tag{4}$$

For more details see the paper [5].

2.5 Advantage-actor-critic

Mnih et al. introduce in the paper "Asynchronous Methods for Deep Reinforcement Learning" [4] the deep reinforcement learning algorithm asynchronous advantage-actor-critic (A3C). Shortly after introducing A3C, Mnih et al. published a synchronous, deterministic variant of A3C called advantage-actor-critic (A2C) which gives equal performance. A2C combines value-based and policy-based reinforcement learning and consists of two parts. A critic which measures how good a taken action is (value-based) and an actor which controls how the agent behaves (policy-based).

The **actor** learns the policy function $\pi(a|s, \theta)$, probability of choosing action a given state s, which is used to decide the best action a given a specific state s. The actor controls how the agent behaves. θ are the learnable weights of the neural network.

The **critic** learns the value function $V(s, \phi)$, which measures how good a certain state s is to be in. The value function V is used to calculate the expected cumulative reward $Q(s, a)$ from following the policy π from state s. ϕ are the learnable weights of the neural network.

$$Q(s, a) = r_{t+1} + \gamma V^{\pi}(s_{t+1}) \tag{5}$$

To update the policy function, the **Advantage function** is used which tells the improvement of a certain action compared to the average action taken at state s. In other words, it estimates the improvement of the true reward compared to the expected reward of the current state s by using the temporal difference error:

$$A(s, a) = Q(s, a) - V(s) \tag{6}$$

The advantage function pushes up the probability of an action from a state s if this action was better than the expected value. This leads to more stability of the learning algorithm and reduces the variance of the estimate, which improves the sample efficience of the algorithm.

To learn an optimal policy, we minimize the **policy** $\pi(a|s, \theta)$ **loss** function

$$loss_{\pi} = -log(\pi_{\theta}(a|s)) * A \tag{7}$$

The **gradient** of the **policy** $\pi(a|s, \theta)$ loss is therefore

$$\nabla\theta = A\nabla_{\theta} log\pi_{\theta}(a|s) \tag{8}$$

To learn the value function we minimize the **value** $V(s, w)$ **loss**

$$loss_V = \sum (R - V(s))^2 \tag{9}$$

3 Imagination-augmented agent

The paper "Imagination-augmented agents for deep reinforcement learning" from Weber et al. [10] combines the advantages of model-free and model-based reinforcement learning to get an agent which is robust against model imperfections, but is able to use the advantages of model-based agents. The imagination-augmented agent learns therefore to combine information from a model-free and a model-based imagination-augmented path.

Figure 2 shows the network architecture of the imagination-augmented agent, which will be explained in the following.

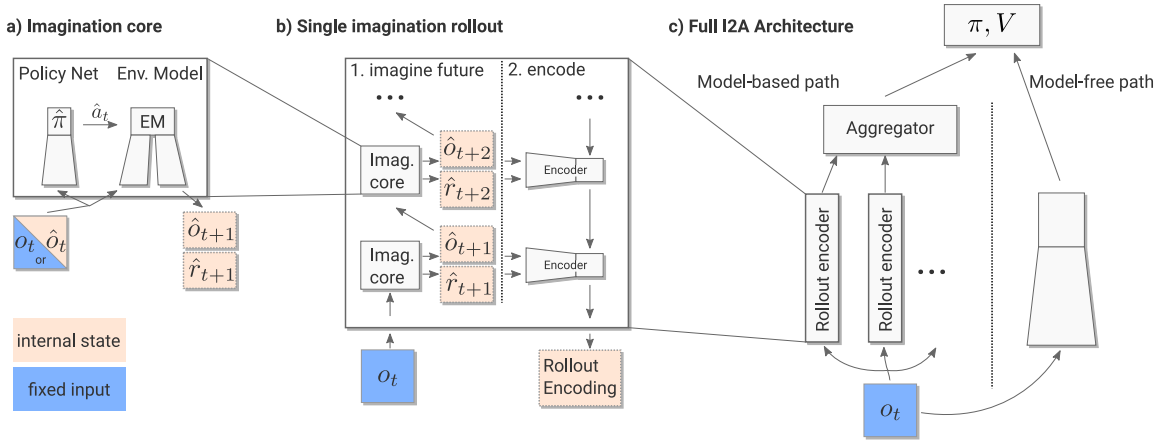


Figure 2: Network architecture for deep reinforcement learning which combines model free and model based reinforcement learning. a) imagination core (IC) predicts the next time step conditioned on an action sampled from the rollout policy $\hat{\pi}$. b) the imagination core imagines trajectories of features f encoded by the rollout encoder. c) the full i2a

3.1 Imagination core

The **imagination core** (Figure 2 a) imagines trajectories of features $\hat{f}_{t+1} = (\hat{o}_{t+1}, \hat{r}_{t+1})$, containing the next observation $\hat{o}_{t+1}$ and the next reward $\hat{r}_{t+1}$, given the observation o_t or $\hat{o}_t$, where o_t refers to the input i2a gets and $\hat{o}_t$ refers to an internal state of i2a, which is an output of a previous rollout with the imagination core. To do so the imagination core uses a rollout policy, which predicts an action given the current state, and an environment model (see chapter 3.2), which predicts the next observation and the next reward.

The **rollout policy** $\hat{\pi}$ (Figure 2 a) is a model-free reinforcement learning agent which should, given the same observation, predict the same action as the i2a policy π (Figure 2 c). To ensure this, the rollout policy needs to be similar to the i2a policy π, which can be ensured by minimizing the distillation loss, the cross entropy between the rollout policy $\hat{\pi}$ and the i2a policy π:

$$l_{dist}(\pi, \hat{\pi})(o_t) = \lambda_{dist} \sum_a \pi(a|o_t) log \hat{\pi}(a|o_t) \tag{10}$$

with scaling parameter λ_{dist}.

3.2 Environment Model

The environment model (Figure 2 a) predicts (imagines) the next observation $\hat{o}_{t+1}$ and next reward $\hat{r}_{t+1}$, given observation o_t or $\hat{o}_t$ and action $\hat{a}_t$. The environment model could be any kind of model that fulfills this condition, but in most cases no model of the environment is given. To solve this problem the environment model is also a trainable neural network. A trained environment model can not be assumed to be perfect, it might sometimes make wrong predictions, but this does not pose a problem, as the imagination augmented agent is robust to imperfect environment models.

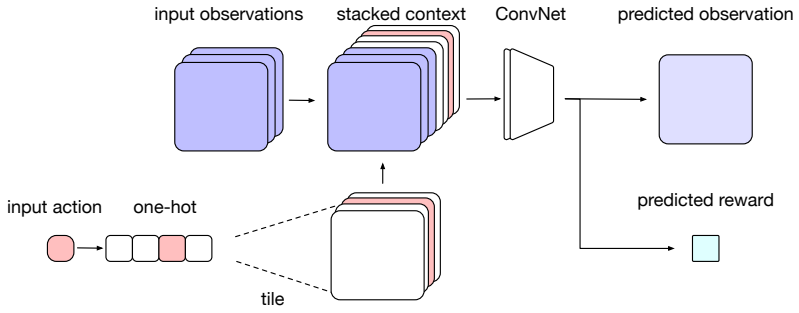


Figure 3: Environment model. The input action is broadcasted and concatenated to the observation. A convolutional neural network then processes the input and predicts the next observation and the expected reward.

Figure 3 shows the environment model architecture. It gets as input the last observation o_t as rgb image and an action a, which was selected from the rollout policy $\hat{\pi}$. The action is converted into a one-hot vector and tiled to the size of the observation. The tiled action is then concatenated with the input observations o_t and named stacked context. The number of channels of the stacked context is now equal to three rgb channels plus the number of possible actions. Given the stacked context as input a convolutional neural network with two heads predicts the next observation o_{t+1} and the expected reward r_{t+1}.

The neural network is trained with transitions of the form $(o_t, a_t) \rightarrow (o_{t+1}, r_{t+1})$ generated from a pretrained model-free advantage-actor-critic policy. This is necessary because a random agent is not able to generate a representative set of states, as it sees few rewards in some of the domains.

The observation head of the environment model is trained by maximizing the log-likelihood of the probability $p(o_t|a_{t-1}, o_{t-1})$, which is a bernoulli distribution given by

$$p(o_t|a_{t-1}, o_{t-1}) = \prod_{n \in I} p_n^{o_{t,n}} (1 - p_n)^{1 - o_{t,n}} \tag{11}$$

where p_n is the prediction of a single pixel $n \in I$ and I is the set of all pixels in the image corresponding to the true observation o_t at timestep t. Maximizing the log-likelihood of the probability $p(o_t|a_{t-1}, o_{t-1})$ is the same as the binary cross entropy loss between the true image o_t and the predicted image p:

$$L_{env}(p, o) = \frac{1}{N} \sum_{n \in I} o_{t,n} \log p_n + (1 - o_{t,n}) \log(1 - p_n) \tag{12}$$

To train the predicted reward a L2 loss is used. The environment model can either pretrained before embedding it within the i2a architecture, or jointly trained with the agent. For our experience we used a pretrained environment model.

3.3 Rollout Encoder

The rollout encoder processes the imagined rollout as a whole and learns to interpret it, by using any useful information and ignoring information when necessary, see Figure 2b. To do so each rollout encoder predicts a imagined trajectory $T = (\hat{f}_{t+1}, ...\hat{f}_{t+n})$, which is a sequence of features, by performing n rollouts, given the input observation o_t and a start action a_t. The encoder consists of a convolutional neural network (CNN) followed by an long-short-term-memory (LSTM) network which processes the information given by the imagined trajectory T. By using an LSTM network the encoder is able to learn long-term dependencies in the rollouts.

3.4 I2A Architecture

As described above the I2A architecture combines model-free and model-based reinforcement learning (figure 2c). The model-based path performs for each action a the agent can take an imagination rollout with the rollout encoder. The aggregator then concatenates all rollouts. The model-free path is just a neural network with CNN layers followed by fully connected (FC) layers. If only the model-free path would be used the architecture would be equal to a model-free reinforcement learning architecture.

The outputs of the model-free path and the model-based path are then simply concatenated and passed to an output policy network which consists of one FC layer and two heads, the policy head π and the value head V. As I2A is just an architecture design it can be trained with advantage-actor-critic as described in section 2.5.

3.5 Model-free baseline

For the model-free baseline an a2c architecture is used consisting of two convolutional layers followed by a FC layer and two FC output heads, one for the policy and one for the value function.

3.6 Copy Model

To verify that the improvement of the i2a agent does not rely on the higher model complexity, a copy model is used for comparison. The copy model has the same network architecture as the i2a model but instead of using an environment model a 'copy' model is used which simply returns the input observation.

4 Mini Pacman

The performance improvement of i2a over the model-free baseline is most obvious on planning intensive tasks, as for example the atari game MsPacman. In MsPacman, the player 'pacman' needs to avoid situations where he gets enclosed by 'ghosts'. Also the 'ghosts' are not always visible to the player, which results in the necessity to predict the current and the future positions of the 'ghosts' to avoid them. MsPacman is also a path planning problem as a level is successfully finished when 'pacman' has collected all pills.

Even though MsPacman would be a nice task to compare i2a with the model-free baselines, it is impossible to train the model with current hardware, due to the very long training time of i2a. The model-based path of i2a performs in each forward step for each possible action one rollout. Each rollout consists of n next observation predictions and each observation has the size of a MsPacman input observation, which is 210x160x3 pixel. Even if the observation size is downsampled to 80x80x3 pixel, the number of weights is way to high.

As our goal is to verify the performance improvement based on a planning problem and not on a vision problem, a mini version of pacman is used. The mini version is called 'MiniPacman' and has an observation size of 15x19x3 pixel, an image of MiniPacman is depicted in Figure 4. MiniPacman makes the vision problem easier but preserves the reinforcement learning problem.

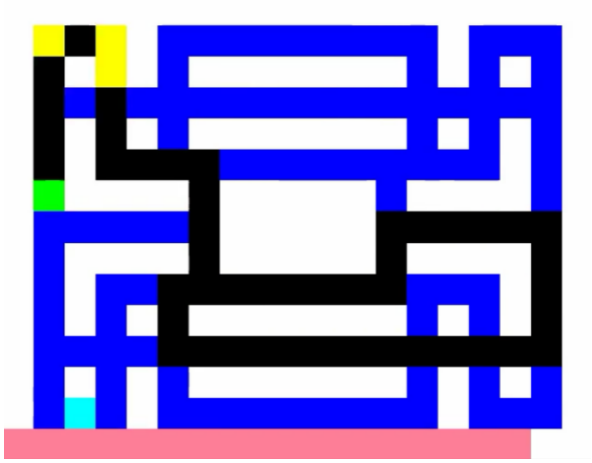


Figure 4: Mini Pacman

MiniPacman can be played in different modes. All modes share the same environment but vary in their reward structure and level termination. In the i2a paper they compared 5 different modes, we only trained i2a in 2 different modes, Regular and Hunt, due to restricted time and resources. In the mode Regular a level is cleared when all food is eaten, in Hunt when all ghosts are eaten or after 80 steps. The rewards associated with different events are listed in the table below:

Task	At each step	Eating food	Eating power pill	Eating ghost	Killed by ghost
Regular	0	1	2	5	0
Hunt	0	0	1	10	-20

The MiniPacman implementation we used is the same as the one used in the i2a paper and can be found on the github repository of Sébastien Racanière [7].

5 Results

This section shows the results of the environment model and the i2a implementation. Also it shows the performance of i2a over the model-free baseline and the copy model.

5.1 Environment Model Training

Our trained environment model is an imperfect model, which is in most cases able to predict a few steps into the future, but it is bad at predicting far into the future.

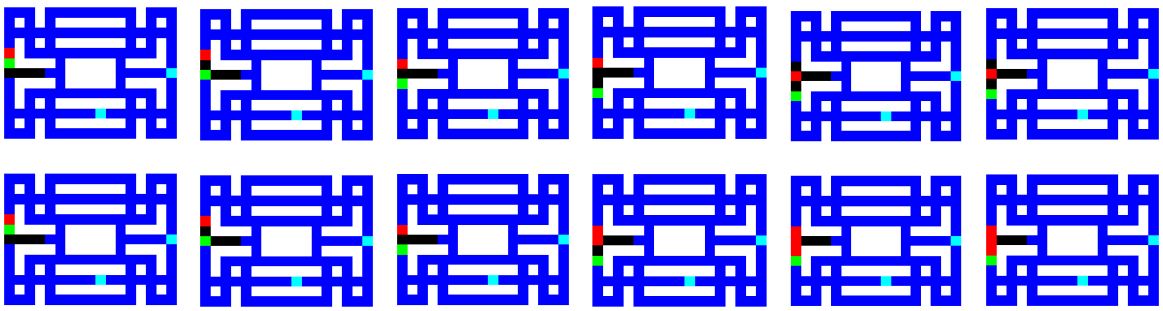
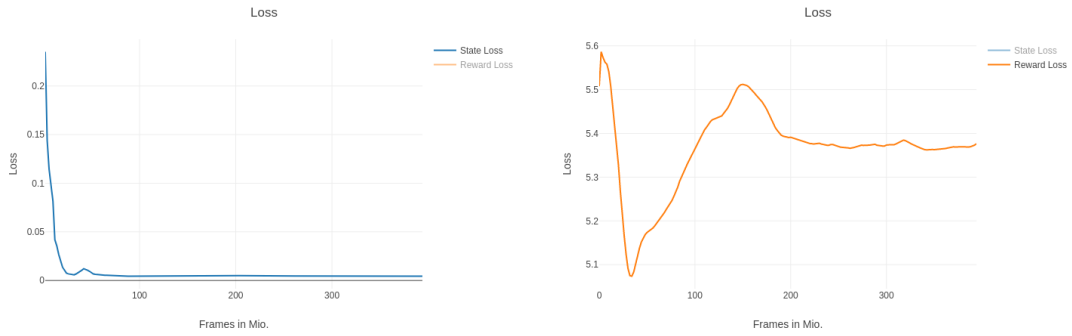


Figure 5: Environment model rollouts, top true observations, bottom predicted observations, rollout steps from left to right

Figure 5 shows the true next observations in the upper row and the predicted next observations in the lower row. From left to right it shows the rollout steps, the prediction is always based on the previous rollout step. In the beginning the prediction is quite good but the errors accumulate, leading to prediction errors like the one in rollout step 3 of figure 5. The environment model is unsure where the ghost, the red points, are moving and as a result it predicts the ghost at two positions.

The environment model was trained for 400 million frames, on a Titan Xp GPU for ∼35 hours with around 3100 frames per second. As optimizer we used Adam with a learning rate of 0.0007 and a weight decay of 0.01. The batch size was 2000 and we weighted the reward loss with a coefficient of 0.01 compared to the frame loss. To train the environment model with a representative set of observations, we used as described in section 3.2 a pretrained a2c policy to create samples of pairs of observation and reward.



(a) Environment model frame loss curve

(b) Environment model reward loss curv

Figure 6: Environment model learning curves

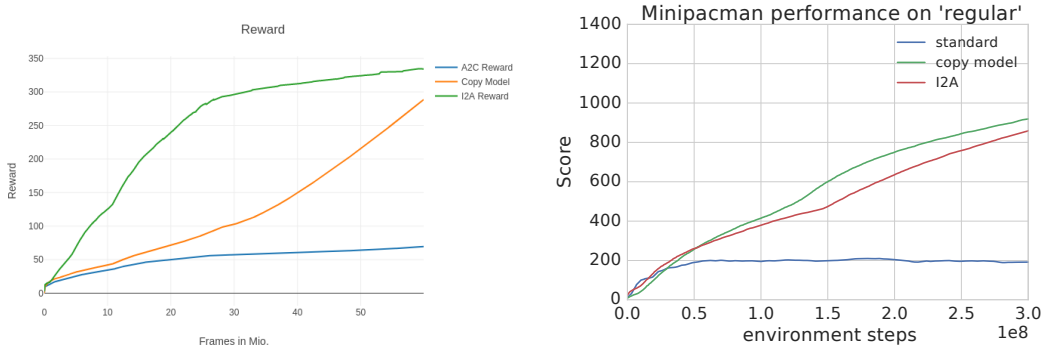
The figure 6a shows the loss curve of the environment model training. It seems as if the loss stagnates very soon, but the main advantage in correct predictions is learned in the part with small loss changes. To learn the general structure of the maze is easy but to predict the correct position of the player and

the ghosts is really hard, but has only a small impact on the loss. In figure 6b the training curve of the reward loss is shown.

5.2 I2A MiniPacman Results

In this section we demonstrate the performance of our reinforcement learning agent and compare our results with the results of the "imagination augmented agent for deep reinforcement learning" paper [10]. To do so we trained three different agents, an a2c model-free baseline, the copy model and the i2a agent on the reinforcement learning task MiniPacman in the modes "Regular" and "Hunt".

As baseline for our implementation we used the a2c reinforcement learning pytorch implementation of Ilya Kostrikov[3].

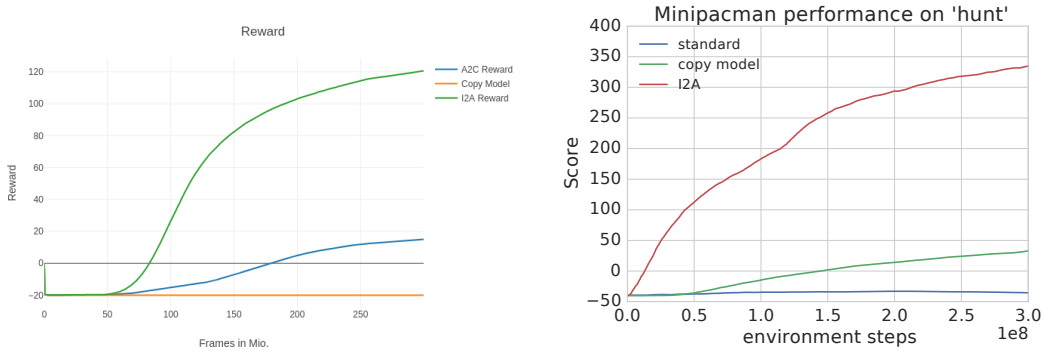


(a) Received reward of i2a run with 2 rollouts (green), model-free a2c baseline (blue) and copy model (orange).

(b) Received rewards from the "imagination augmented agent for deep reinforcement learning" paper [10].

Figure 7: Learning curves for different agents on the task MiniPacman Regular

Figure 7a shows our training results for the reinforcement learning task MiniPacman Regular. The a2c baseline is clearly outperformed by the copy model and the i2a model and in the begining the i2a model also outperforms the copy model, but after 60 million frames the difference between the i2a model and the copy model gets smaller. It is therefore possible that the performance improvement corresponds mainly on the parameter increasement and not on the imagined rollouts. Our results do not contradict the results of the original i2a paper which also have no improvement of i2a over the copy model in the mode "Regular" as shown in figure 7b. We trained our agents for 60 million frames on a Titan Xp which took for the a2c baseline ∼10 hours, for the copy model ∼22 hours and for the i2a ∼48 hours.



(a) Received reward of I2A run with 2 rollouts (green), model-free A2C baseline (blue) and copy model (orange).

(b) Received rewards from the "imagination augmented agent for deep reinforcement learning" paper [10].

Figure 8: Learning curves for different agents on the task MiniPacman Hunt

The results of the trained agents on MiniPacman in Hunt mode are shown in figure 8. As you can see i2a outperforms clearly the a2c baseline and the copy model shown in figure 8a. As before our results match the ones from the paper, see figure 8b, only the maximum reward we reached with our i2a agent is not as high as the results of the original paper. This difference could be caused by the fact that they use a rollout length of 5 and we only a rollout length of 2. We trained MiniPacman Hunt for 300 million frames on a Titan Xp which took for the a2c baseline ∼48 hours, for the copy model ∼94 hours and for the i2a ∼215 hours.

The reproduced results from the paper support the statement that the performance gap between i2a and baselines is high for tasks where the rewards are sparse and where planning ahead is important to avoid irreversible actions, as for example move pacman to a position where he gets enclosed by ghosts. Our pytorch impementation can be found on github [2].

6 Future Work

The main drawback of i2a is the very long training time, which leads to the problem that it can only be applied on toy problems. This problem is already addressed in the paper "Learning and Querying Fast Generative Models for Reinforcement Learning" [1] Buesing et. al., where they reduce the training time by working on a latent space representation of the observation, instead of the full observation. By doing this Buesing et al. where able to apply i2a to the atari game MsPacman without downsampling the input image.

i2a can only benefit from the model-based path if the environment model provides relevant information. If the used environment model is bad, i2a can only perform as good as its base line, so it should be worthwhile to pursue further research to create a good environment model. This can be done by using a different environment model architecture, by applying new training approaches or by training the environment model together with the i2a agent.

Also it might be valuable to find better policies for the rollout strategy. Currently for each action one rollout is performed and the used policy imagines the future, depending on the actions it thinks the agent will make. But it might be more valuable to imagine what will happen in extreme cases.

Another possible precision improvement is to improve the exploration by using a reward bonus as done in "Count-Based Exploration with Neural Density Models" [6].

References

[1] Lars Buesing, Theophane Weber, Sébastien Racanière, S. M. Ali Eslami, Danilo Jimenez Rezende, David P. Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. Learning and querying fast generative models for reinforcement learning. *CoRR*, abs/1802.03006, 2018.

[2] Angela Denninger Florian Klemt. Pytorch latent i2a. `https://github.com/FlorianKlemt/pytorch-latent-i2a.git`.

[3] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr`, 2018.

[4] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[6] Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *CoRR*, abs/1703.01310, 2017.

[7] Sébastien Racanière. I2a nips workshop. `https://github.com/vasiloglou/mltrain-nips-2017/blob/master/sebastien_racaniere`.

[8] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. pages 279–292, 1992.

[9] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. May 1989.

[10] Theophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.