Latent Space Imagination Augmented Agents for Deep Reinforcement Learning

Florian Klemt

Munich, 18.10.2018

Dynamic Vision and Learning Group

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

1 Abstract

Recent developments in combining model-based and model-free reinforcement learning lead to the development of Imagination Augmented Agents (I2A) [13]. We will briefly explain the basic concepts of I2As, before focusing on their central problems. Specifically we will analyse complexity problems, that arise from the way I2As imagine future states and evaluate ways to circumvent these problems, based on imagining states in an abstract state space. These approaches are based on the results of Buesing et al. [3]. We provide an implementation of I2A, as well as an implementation of the version using abstract spaces [1] [4] in the python deep-learning framework PyTorch [9].

[1] `https://github.com/FlorianKlemt/pytorch-latent-i2a.git`

Contents

2 Preliminary Work

While reinforcement learning (RL) was introduced as early as the 1950s with the first appearance of the Bellman equation, the last two decades brought a multitude of advances in the area of function approximation, especially deep reinforcement learning. In the applied field of computer vision and control, Mnih et al. published breakthrough papers in 2013 [7] and 2015 [8], managing to solve several Atari games via Deep-Q-Learning at a human or even super-human level of play. Since then the Atari Games domain has advanced to a standard application in the field, which was the basis for many publications introducing new reinforcement learning algorithms for solving vision based tasks.
The basic premise of reinforcement learning is an agent interacting with an environment, by taking actions and getting a numeric reward signal from the environment, telling him how desirable the current state of the environment is. The agent optimizes his choices, resulting from his policy π, with the goal of getting the maximum amount of long-term reward. Usually the information included in states is chosen, such that they have the Markov property, meaning each state encompasses all information relevant for future states. The environment can than be described by a Markov Decision Process (MDP). The MDP can be expressed as a 5-tuple (S, A, T, R, γ), where S is the set of possible states, A is the set of actions, $T(s'|s, a)$ the transition probability function specifying the probability of landing in state s' given a state-action pair (s, a), $R(s'|s, a)$ the reward function specifying the reward received when transitioning from state s to state s' via action a and γ a discount factor weighting immediate reward against future reward.

2.1 Model-free and model-based RL

In model-based RL methods, the agent has access to a model of the environment, providing the agent with an approximation of the MDP describing the intrinsics of the environment. Having access to such an environment model simplifies the task of learning an optimal policy, as the agent knows the effects of an action without the necessity of actually performing it. However, for many tasks it is hard, if not outright impossible to construct a perfect model of the environment. An alternative to this approach, that does not need initial information about the environment, is realized by model-free RL methods, that do not have access to a model of the problem underlying MDP.
Reinforcement learning methods include value-based approaches like Q-Learning [12] and policy-based approaches like Policy Gradient [11]. Q-Learning maximizes the so-called Q-Value $Q(s, a)$ indicating the expected long-term reward, when the agent is in state s and taking action a, via the optimality equation

$$Q^*(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a')|s, a] \tag{1}$$

where r is the immediate reward of taking action a in state s. From the resulting optimal Q-Values an optimal policy π can easily be derived. In the case of deep RL, which we use for the rest of the paper, the Q-Value function is learnable and is approximated with a neural network.
In contrast policy gradient methods directly optimize the policy by computing a gradient on the current policy by

$$\nabla \mathbb{E}_\pi[r(\rho)] = \mathbb{E}_\pi \left[\Big(\sum_{t=1}^{T} R_t \nabla \log \pi(a_t|s_t) \Big) \right] \tag{2}$$

$$R_t = \gamma^t Q(s_t, a_t) \tag{3}$$

where $r(\rho)$ denotes a trajectory of T consecutive states and actions and R_t the discounted expected reward in step t when following policy π. When using deep RL, the policy depends, as before, additionally on the model parameters θ.

2.2 Advantage Actor Critic

A variant of policy gradient is the recently developed Asynchronous Advantage Actor Critic (A3C) algorithm [6] of which we use a synchronous variant (A2C) in our work. It consists of an actor specifying the agent's policy and a critic predicting the state values. It replaces the expected discounted reward by an advantage term $A^{\pi_\theta}(s_t, a_t)$ resulting in

$$\nabla \mathbb{E}_{\pi_\theta}[r(\rho)] = \mathbb{E}_{\pi_\theta} \left[\Big(\sum_{t=1}^{T} A^{\pi_\theta}(s_t, a_t) \nabla \log \pi_\theta(a_t|s_t) \Big) \right] \tag{4}$$

where the advantage term is defined by

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{5}$$

This leads to an increase of the probability of choosing a state-action pair, if the reward following it is larger than expected, and vice versa. This substraction of the learned value function, which is the output of the critic head of the network, makes the learning more stable and reduces the variance of the estimate. This drastically improves the sample efficiency of the algorithm.

2.3 Imagination Augmented Agents

This work is based on the concepts of Imagination Augmented Agents (I2A), first introduced by Weber et al. in 2017 [13]. We explain the basic ideas and the reasoning behind I2As in this section. For a more comprehensive view we refer to the paper itself.
Essentially I2As novelty is based on giving the agent a way to predict the future development of the environment's state and combining this model-based reinforcement learning approach with model-free reinforcement learning. The central element, allowing the agent to predict the future is the imagination core, consisting of a pretrained environment model and a rollout-policy network (see Fig. 1a). When given an observation o_t as input, the policy net decides on an action $\hat{a}_t$ and the environment model predicts the next observation $\hat{o}_{t+1}$ and the corresponding reward $\hat{r}_{t+1}$. The input observation is logically equivalent to what in previously mentioned algorithms is called state s_t, but we rename it here to avoid confusing notation in the context of I2As.

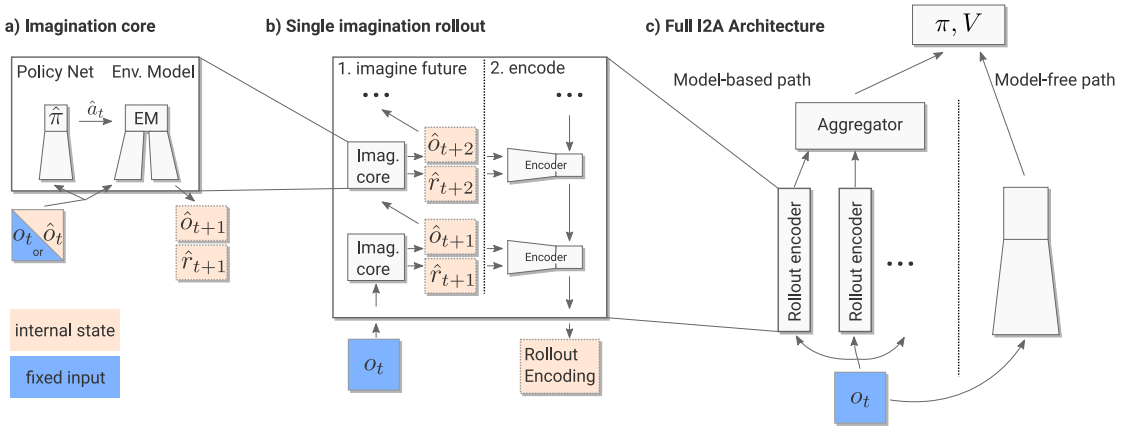


Figure 1: Architecture of the basic I2A model. (Image taken from Weber et al. [13])

Using the imagination core, the agent can imagine future trajectories, starting from the current state. In a single imagination rollout, each state-reward pair from such an imagined trajectory is fed into a convolutional network and subsequently into an LSTM aggregating the temporal information (see Fig. 1b). The I2A model does one rollout for each possible initial action and concatenates the resulting rollout encodings, before feeding them into an output policy network predicting policy π and state-value V (see Fig. 1c). This procedure is called the model-based path, due to its use of the environment model. As mentioned in the section on model-based RL, a perfect model is often hard to obtain in real-world problems. However an imperfect model often proves to have a very negative effect on learning. The algorithm might not converge anymore, even on a local minimum of the true task, but rather on an erroneous minimum of the imperfect model. Additionally the agent will focus on actions, that falsely seem good according to the model, leading to little exploration, while at the same time following actions with a small true reward, in turn leading to slow learning. To overcome these negative effects that can arise from using an imperfect model, the I2A model additionally features a model-free path, simply feeding the observations through a convolutional network and uses them as an additional input to the output policy network (see Fig. 1c). With this approach the network can learn to ignore parts of the model that prove to be wrong and learn the corresponding mechanics of the environment via the model-free path, while still being able to use parts of the model that prove to be correct. This ability makes I2A models highly functional on complex problems.
The architecture exhibited very promising results on learning the tasks of the puzzle game Sokoban and a custom 15x19 pixel implementation of Pacman (MiniPacman), as presented by Weber et al [13],

outperforming a baseline A3C algorithm [6] significantly in the task of Sokoban and in sparse-reward versions of MiniPacman.

3 Limits of standard I2A

While the I2A model as described by Weber et al. [13] is a very promising step in tackling complex problems and using imperfect models of an environment, it has several drawbacks that limit its use on real-world problems. First and foremost the issue is scalability. The imagination rollout shown in Figure 1b uses the imagination core to predict the imagined next state $\hat{o}_{t+n+1}$ from the previous state $\hat{o}_{t+n}$, with $n \in [0, T]$ where T denotes the rollout length. It should be noted, that the size S of all $\hat{o}_{t+n}$ is the same as the size of the observation o_t, on which the rollout is based. For large observations, which can arise easily, the rollout therefore needs a lot of computational resources. Among other reasons this is why Weber et al. [13] used a low resolution custom implementation of MsPacman.
The environment model inside the imagination core, feeds the data through a bottleneck layer of size $l << S$ before upsampling the data to the original size S, for the prediction of the next observation. This means the actual expressiveness of the predicted observation $\hat{o}_{t+n}$, is not depending on S but rather on the bottleneck size l. The central idea behind Latent Space I2A is to intelligently downsample the input observation o_t once and do the rollouts in the downsampled latent space. The implementation of this idea, as well as the evaluation of approaches to it, are the core topics of this work.
Another problem, that also falls under the category of scalability, is the performance of I2A under the premise of large action-spaces. As I2A does a rollout for each possible action a_t in the current state s_t, it scales linearly with the size of the action-space. While this does not pose a problem for the tasks chosen by Weber et al. [13], Sokoban and a custom minimal implemenation of MsPacman, which both have an action-space of 5, it is easy to find tasks with much larger action-spaces. In the extreme case of continuous action-spaces standard I2A does not work at all. We will not go deep into this problem in this paper, but it is important to know about this limitation.

4 Environment Model

Changes to the environment model, I2A uses to predict the future in its model-based path, are central to implementing a version of I2A, that uses latent space representations in its rollouts. The architecture of the environment model using the latent space can be broken down into three main parts. First of all, an encoder for transforming observations into compact latent space representations (LSR) and a decoder for reconstructing the corresponding observation given an LSR of it and predicting the reward of that step from the LSR. To predict the LSR of the next observation given the previous LSR, we use a third part of the architecture called the state transition. All three parts are realized by neural networks and will be described in detail in this section.
We study two different versions of environment models, deterministic State Space Models (dSSM) and stochastic State Space Models (sSSM). When using dSSMs, the state transition is a determinisic function (see Figure 2: dSSM-DET and dSSM-VAE), whereas sSSMs model uncertainty in the transition, by additionally basing the state transition $s_{t-1} \rightarrow s_t$ on a latent variable z_t, itself depending on s_{t-1} and a_{t-1} (see Figure 2: sSSM). dSSMs are further divinded into dSSMs using a deterministic decoder (dSSM-DET) and dSSMs using a stochastic decoder (dSSM-VAE), which takes the latent variable z_t as additional input. For sSSMs we always use the stochastic decoder. Figure 2 visualizes a single prediction step in each model.
When using stochastic models two additional network parts predicting the prior and the posterior of the latent variable are used.

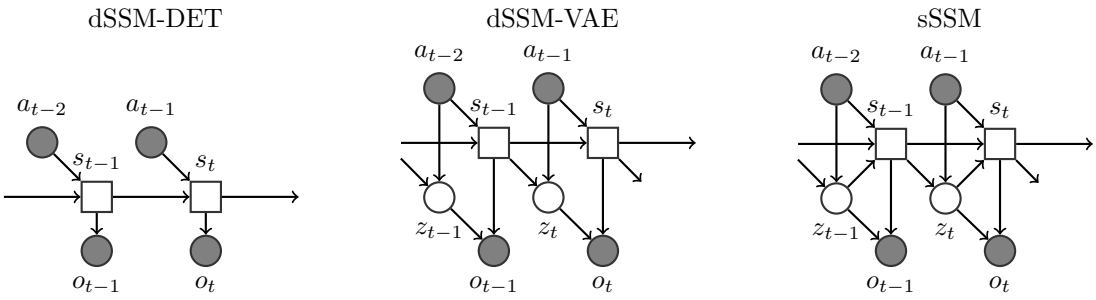
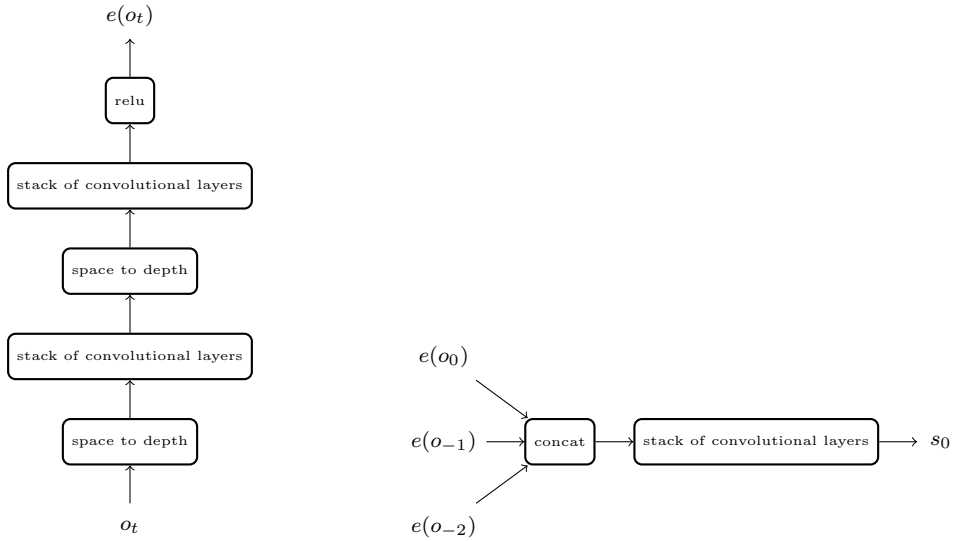


Figure 2: The graphical models representing the architectures of different environment models. Boxes are deterministic nodes, circles are random variables and filled circles represent variables observed during training. (Images and caption taken from Buesing et al. [3])

4.1 Encoder and Initial State Module

The purpose of the initial state module is to produce the initial LSR s_0, which is the input to the imagination rollout. This is analogous to the input observation o_t in the standard I2A model. An observation is downsampled into an encoding via a combination of stacks of convolutional layers and space-to-dapth operations, that rearrange pixel-blocks in the image into channels (see Figure 3a). As some environment dynamics cannot be reliably predicted from a single observation (eg. the movement direction of symmetric objects or the existence of objects that are not visible in that observation) we base s_t on encodings of the last 3 observations as depicted in Figure 3b. The encodings are concatenated and fed through another stack of convolutional layers to produce s_0, in order to allow the network to choose which information to use from which encoding.



a) Encoder module

b) Initial State module

Figure 3: Schematic representations of the encoder module, encoding a single observation, and the initial state module, computing the first initial state s_0 as a function of the embedding $e(o_i)$ for $i = -2, -1, 0$ of three previous observations. [3]

4.2 Prior and Posterior module

To construct the distribution from which the latent variable z_t is sampled, we use a further network part, called the prior module. It processes the inputs of s_{t-1} and a_{t-1} by a stack of convolutional layers with two network heads, predicting mean μ_{z_t} and variance σ_{z_t} of a gaussian distribution p.

Subsequently we can sample the latent variable

$$z_t \sim p(z_t|s_{t-1}, a_{t-1}) \tag{6}$$

The posterior module has a similar structure as the prior module and predicts mean $\hat{\mu}_{z_t}$ and variance $\hat{\sigma}_{z_t}$ of the gaussian distribution $q(z_t|s_{t-1}, a_{t-1}, o_t)$ approximating the true posterior $p(z_t|s_{t-1}, a_{t-1}, o_t)$. The use of the posterior module will be explained further in the next chapter.

4.3 State Transition

For the stochastic State Space model (sSSM) we first sample the latent variable z_t from the prior module, as described in the previous section.
The next state is then computed by the transition function

$$s_t = \begin{cases} g(s_{t-1}, z_t, a_{t-1}) & \text{for sSSMs} \\ g(s_{t-1}, a_{t-1}) & \text{for dSSMs} \end{cases} \tag{7}$$

that is defined by the state transition network layers show in figures 4, 5 and 6. The pool-and-inject module, first introduced by Weber et al. [13], captures global connections.

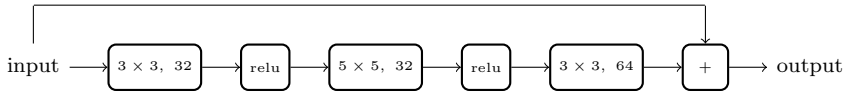


Figure 4: Definition of the residual convolutional stack res_conv.[3]

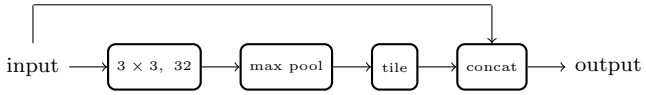


Figure 5: Definition of the Pool & Inject layer.[3]

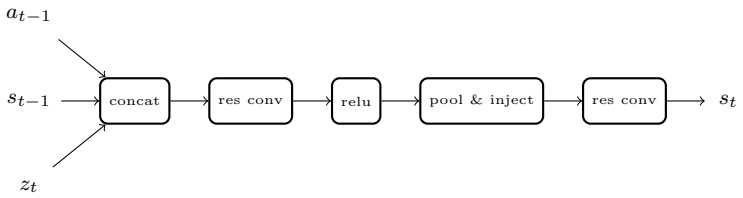


Figure 6: Transition module for computing the state transition function $s_t = g(s_{t-1}, z_t, a_{t-1})$.[3]

4.4 Decoder

We can decompose the decoder into two networks, one for reconstructing the observation from an LSR and one for predicting the reward that was received in the environment step leading to that observation (see Figure 7). For the observation reconstruction part, we use the current LSR as input, optionally concatenated with the latent variable z_t, that was sampled in the state transition leading to this LSR, when using a dSSM-VAE or sSSM model, constructing

$$o_t \sim p(o_t|s_t, z_t) \tag{8}$$

The final layer of the observation reconstruction network has 3 channels, that are interpreted as the logits of the color channels of the reconstructed observation. The output of the reward prediction network is interpreted as logits of bits of a binary representation of the reward. This is explained in more detail in the next section.

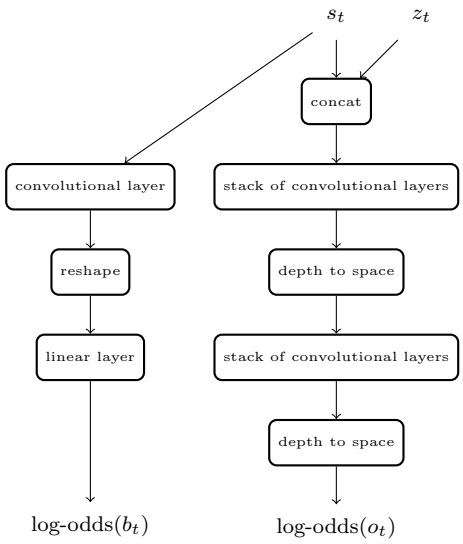


Figure 7: Decoder Module for reconstructing observations from LSRs and predicting rewards.[3]

5 Training the Environment Model

We train our environment models on training samples sampled from a queue, that is periodically updated with observation-reward pairs generated from playing the game. The sampling mitigates the conditioning of observations on the previous observations, which can lead to slow learning due to local minima. As the observations that are generated when using a random policy are not representative, we use a policy that is pretrained with a standard A2C [6]. In a training episode we go through all sampled training samples and compute the rollouts as described by Weber et al. [13], but using the environment model working in latent space. The loss computation differs depending on the algorithm.

5.1 dSSM-DET Loss

The total loss can be decomposed into the frame prediction loss and the reward prediction loss.

5.1.1 Frame prediction loss

Let

$$p_{i,k} := p_\theta(o_{i,k}|a_{0:i-1}, \hat{o}_{-2:0}) \tag{9}$$

be the prediction of a single pixel $k \in I$ where I is the set of all pixels of the frame corresponding to the true observation $o_{i,k}$ at timestep $i \in [1, T]$. The prediction is based on the actions a_n $n \in [0, i-1]$ and an initial context $\hat{o}_{-2:0}$ consisting of the last 3 observations on which the initial state s_0 is based via the encoder and the initial state module.
We use Bernoulli distributions to model pixels of frames. This leads to the following formulation of the likelihood function:

$$p_\theta(o_{1:T}|a_{0:T-1}, \hat{o}_{-2:0}) = \prod_{i=0}^{T} \prod_{k \in I} p_{i,k}^{o_{i,k}} (1 - p_{i,k})^{1-o_{i,k}} \tag{10}$$

For the purpose of maximizing with respect to θ we take the negative logarithm

$$L(\theta) = -\log p_\theta(o_{1:T}|a_{0:T-1}, \hat{o}_{-2:0}) \tag{11}$$

$$= -\sum_{i=0}^{T} \sum_{k \in I} o_{i,k} \log p_{i,k} + (1 - o_{i,k}) \log(1 - p_{i,k}) \tag{12}$$

leading to a binary cross-entropy loss.

5.1.2 Reward prediction loss

We represent the true reward at timestep t via the binary representation

$$\sum_{n=0}^{N-1} b_{t,n} 2^n = \lfloor r_t \rfloor \tag{13}$$

and add a bit indicating the sign, as well as another bit indicating wether the reward is 0. Although 0 reward can be represented without this bit, it makes it easier for the agent to deal with the very common occurrance of 0 reward.
When predicting the reward the bits are treated as independent Bernoulli variables

$$\hat{b}_{t,n} \sim Ber(p_{t,n}) \tag{14}$$

which are optimized using the binary cross-entropy loss described above. This approach has empirically been shown by Weber et al. [13] to be better suited for this application than standard regression.

5.2 Variational Models Loss

When using models that are based on latent variables, namely the dSSM-VAE and the sSSM models, we encounter the problem that the posterior $p(z_t|s_{t-1}, a_{t-1}, o_{t:T})$ is intractable as s_{t-1} depends on the latent variable. To deal with this we use the Kullback-Leibler-Divergence to measure the similarity between approximate posterior $Q = q(z_t|s_{t-1}, a_{t-1}, o_{t:T})$ and the true posterior $P = p(z_t|s_{t-1}, a_{t-1}, o_{t:T})$, which is defined as

$$KL(Q||P) = \int_{z_t} Q \log \frac{Q}{P} \tag{15}$$

To get rid of the intractable true posterior P in the denominator, we perform a series of transformations leading to the Evidence Lower Bound (ELBO).

$$\begin{aligned}
\int_{z_t} Q \log \frac{Q}{P} &= -\int_{z_t} Q \log \frac{P}{Q} \\
&= -\left(\int_{z_t} Q \log \frac{p(z_t, s_{t-1}, a_{t-1}, o_{t:T})}{Q} - \int_{z_t} Q \log p(s_{t-1}, a_{t-1}, o_{t:T}) \right) \\
&= -\int_{z_t} Q \log \frac{p(z_t, s_{t-1}, a_{t-1}, o_{t:T})}{Q} + \log p(s_{t-1}, a_{t-1}, o_{t:T}) \int_{z_t} Q \\
&= -\int_{z_t} Q \log p(z_t, s_{t-1}, a_{t-1}, o_{t:T}) + \int_{z_t} Q \log Q + \log p(s_{t-1}, a_{t-1}, o_{t:T}) \\
&= -\mathbb{E}_q[\log p(z_t, s_{t-1}, a_{t-1}, o_{t:T})] + \mathbb{E}_q[\log Q] + \log p(s_{t-1}, a_{t-1}, o_{t:T}) \\
&= -ELBO + \log p(s_{t-1}, a_{t-1}, o_{t:T})
\end{aligned} \tag{16}$$

In this expression the part $\log p(s_{t-1}, a_{t-1}, o_{t:T})$ is independent of q and the latent variable z_t and can therefore be omitted for the goal of maximizing with respect to q. This means we can maximize the ELBO instead of minimizing the KL-Divergence which in turn is equivalent to maximizing the likelihood.

$$\begin{aligned}
ELBO &= \mathbb{E}_q[\log p(z_t, s_{t-1}, a_{t-1}, o_{t:T})] - \mathbb{E}_q[\log Q] \\
&= \mathbb{E}_q[\log \left(p(o_{t:T}|s_{t-1}, a_{t-1}, z_t) p(z_t|s_{t-1}, a_{t-1}) p(s_{t-1}, a_{t-1}) \right)] - \mathbb{E}_q[\log Q]
\end{aligned} \tag{17}$$

Via the aforementioned state transition function 4.3 and assuming a Markovian environment we can simplify

$$p(o_{t:T}|s_{t-1}, a_{t-1}, z_t) = p(o_{t:T}|s_t) = p(o_t|s_t) \tag{18}$$

Using this simplification together with $p(s_{t-1}, a_{t-1}) = 1$ leads to our ELBO formulation that resembles the variational frame loss

$$ELBO = \mathbb{E}_q[\log p(o_t|s_t) + \log p(z_t|s_{t-1}, a_{t-1}) - \log Q] \tag{19}$$

The second term in the expectation is the output of the aforementioned prior module and the third term the output of the posterior module.
The reward loss works the same way as in the deterministic case, as its prediction does not depend on the latent variable.

6 Latent Space I2A architecture

When using an environment model based on latent spaces, we need to change the input to the I2A model from observations to abstract representations. Figure 8 shows the conceptual changes compared to the standard I2A model (see Figure 1). We encode the last three observations into the first latent state, which is the input to the rollouts. The model-free path, as with standard I2A, gets the full observation o_t as input. The core of the I2A model stays unchanged, making the latent space approach very powerful as it can be easily utilized on any I2A model.

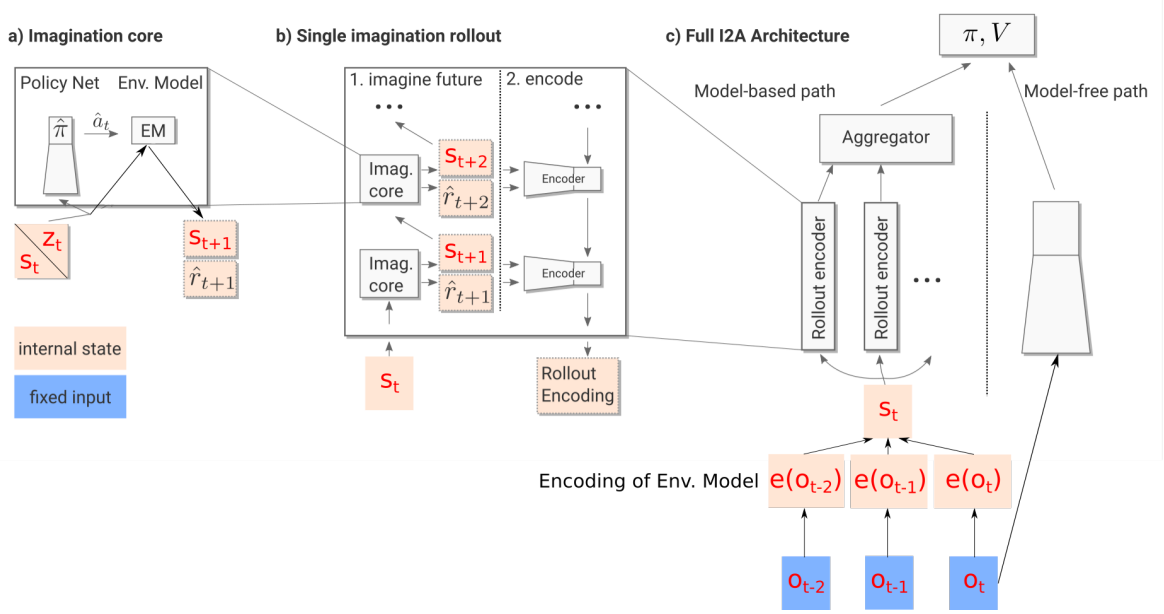


Figure 8: Architecture of the Latent Space I2A model. Differences from the basic I2A model are marked in red.

7 Results

In this section we show results from our implementation of the Latent Space I2A model described above.

7.1 MsPacman

To ensure the possibility of a meaningful comparison to the results of Buesing et al. [3] we focused like them on the atari game MsPacman in our experiments. We use the environments provided by OpenAI Gym [2], which features a wrapper around the Arcade Learning Evironment (ALE) [1], a library for emulating Atari 2600 games. Specifically we use the environment "MsPacman-v0", which features MsPacman with in-game frames of size 210,160,3 as input.
Learning to play MsPacman based on pixel input is a very challenging task, even for modern state-of-the-art RL algorithms, due to several reasons.

1. The agent needs to plan far into the future to avoid getting ambushed by ghosts.
2. Initially the agent has no knowledge of objects and has to construct a representation of the abstract concepts of ghosts, food, power pills, walls and the pacman itself from the input pixels. This is additionally complicated due to to the ingame objects not always having a rigid form.
3. Not all relevant information can be seen in every frame. Ghosts and power pills are blinking and the agent often needs to interpolate the ghosts' movement outward to know where they are.
4. While the rewards for eating food are dense at the beginnning of a game, allowing the agent to easily figure out the concept of moving around the map to eat food, the more advanced mechanics of the game, namely eating power pills and subsequently hunting ghosts, are hidden behind sparse rewards.
5. The size of states is very large.

By using the I2A model and an A2C [6] [5] algorithm built around it, we can solve problems 1 and 2 to some extend. We counter problem 3 by defining a state as the concatenation of the last four observations. Problem 4 is a central problem of RL that is not attacked in this work. Using the Latent Space I2A approach allows us to handle problem 5 by internally working on a much smaller problem, putting the MsPacman environment in the realm of practically solvable tasks.

7.2 Runs on MsPacman

In this section we will exhibit the results achieved by our implementation of the Latent Space I2A algorithm on the previously mentioned MsPacman environment, in different configurations. Due to limited computation power and the fact that even when using the Latent Space I2A model, learning MsPacman from visual output remains a very challenging task, we were not able to run MsPacman with all environment models on a sufficient number of training samples (in the order of 100s of million). After the results of Buesing et al. [3] suggesting the dSSM-DET model to perform best in combination with I2A, even though the sSSM predictions are generally more accurate, we decided to use the dSSM-DET model.
For training we used an A2C algorithm using the Latent Space I2A model. Although I2A works with any architecture and we could have used improvements of A2C like Proximal Policy Optimization (PPO) [10], we choose the standard A2C variant to produce results comparable to Buesing et al. We performed 2 runs, one with an I2A rollout length of 2 and one with an I2A rollout length of 5, where rollout length denotes the number of observations the agent predicts into the future. We compare ourselves against an A2C baseline, using a model consisting of two convolutional layers followed by a fully connected layer and two fully connected output heads, one for the policy and one for the value function. Figure 9 shows the mean reward development over samples the agent has seen, for both Latent Space I2A variants, as well as the A2C baseline.
Mean and median of the reward result in similar curves, indicating that the agent does not only manage to play a few very high reward games, but instead features a rather consistent level of gameplay. Figure 10 exhibits the mean and median reward curves for the case of Latent Space I2A with a rollout length of 2.

Figure 9: Mean reward curves of Latent Space I2As, using a dSSM-DET model, with rollout lengths of 2 and 5, compared to the A2C baseline.

Figure 10: Mean and median reward curves of a Latent-Space I2A agent using a dSSM-DET environment model with a rollout length of 2.

For comparison Figure 11 shows the rewards for different agents as reported by Buesing et al. [3]. Our agent resembles the "I2A, distillation, dSSM" agent.

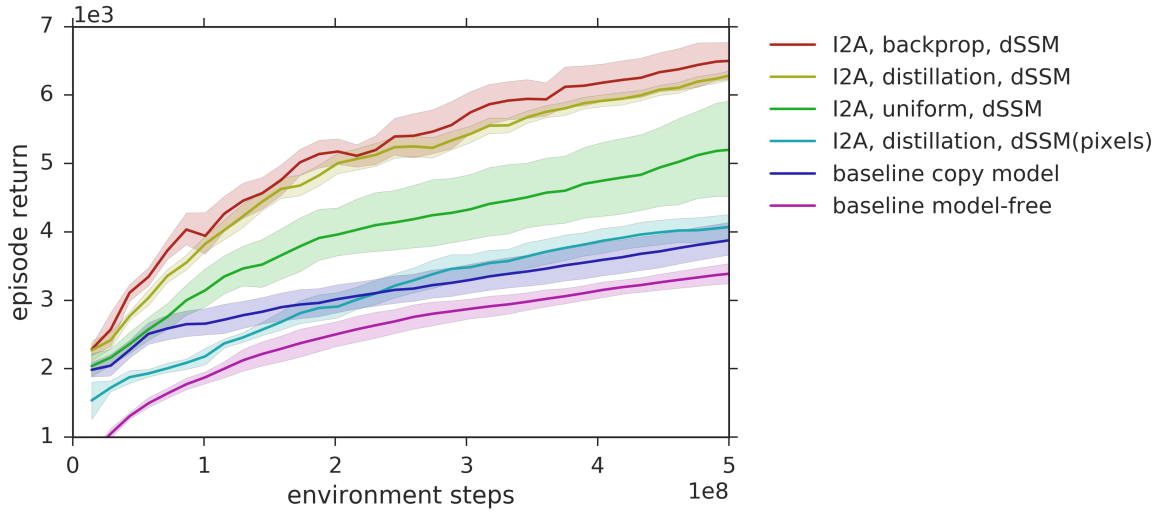


Figure 11: Results for deterministic I2As as reported by Buesing et al. [3]

7.3 Environment Model Example

We trained several environment models (dSSM-DET, dSSM-VAE, sSSM) on the MsPacman environment using the methods described in chapter 4 and 5. We show results for the dSSM-DET model here, as this model is the one that was used in our experiments with the Latent Space I2A model. Figure 12 shows the loss curves for both the cumulative loss of predicting the next 10 observations, as well as the cumulative loss of predicting the next 10 rewards. The state loss drops very fast in the beginning of the training, but soon does a very sharp bend and only learns slowly after that. This phenomenon can be explained by the agent easily learning about the static parts of the environment, but having a harder time predicting the movements of ghosts and pacman.

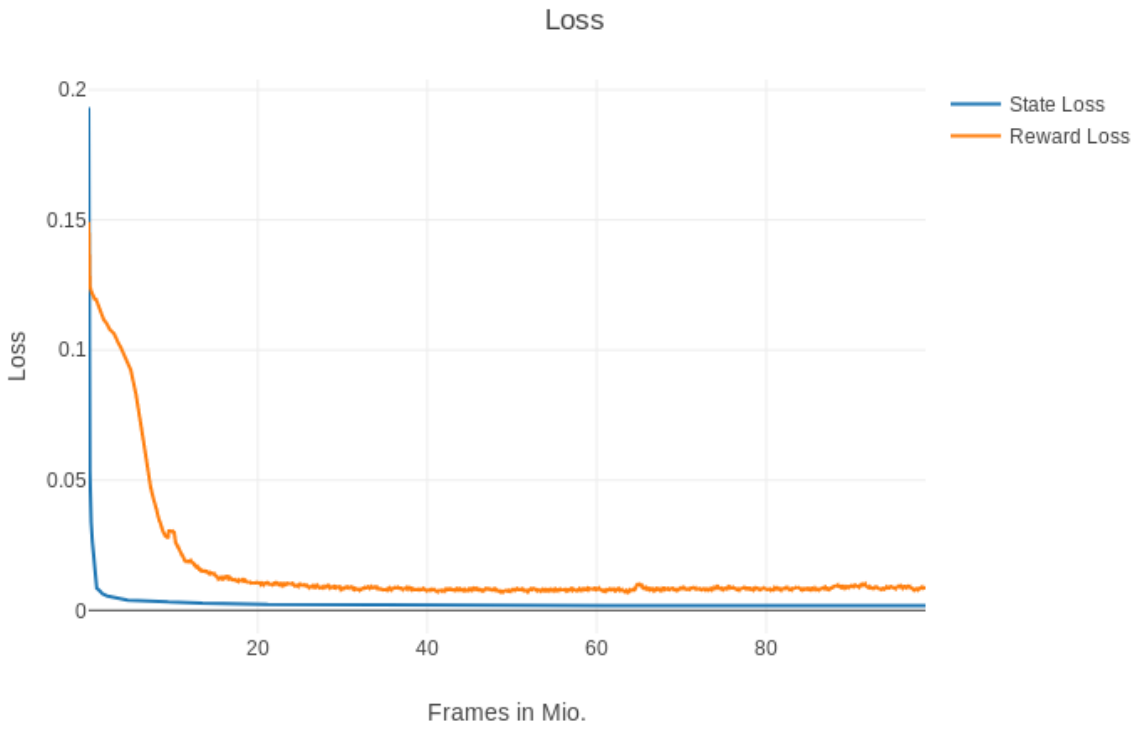


Figure 12: Loss curves of frame and reward prediction when training a dSSM-DET model.

After training the environment model with 100 million observations, its predictions look quite promis-

ing. Figure 13 exhibits an example of the prediction capabilities of the environment model. Here it correctly predicted the movement direction of both initially visible ghosts (red, purple), as well as pacman eating a food element. Interestingly the model predicts in frame 5 the existence and rough position of the blue and orange ghosts, that were not visible in the ground truth frames 1-4. This is possible due to the agent basing his latent representation of the first frame additionally on the encodings of the last 3 frames. Another interesting detail is the agent correctly predicting the visual in-game representation of the reward changing from 40 to 50 (albeit it is predicted a frame too early).

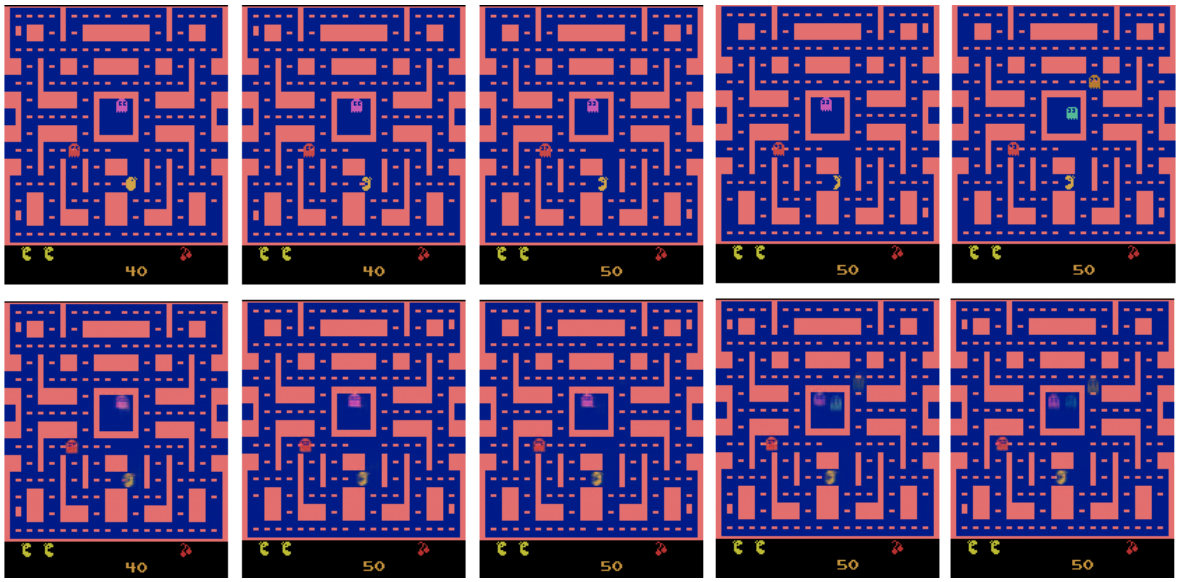


Figure 13:
Upper row: 5 consecutive observations (left-to-right) of the MsPacman environment produced by a pretrained policy.
Lower row: Predictions of the same 5 observations by a trained dSSM-DET environment model.

8 Outlook and future work

There are several approaches that seem promising to further increase the capabilities of Latent Space I2As. A few are mentioned by Buesing et al. [3] as parts of their paper that are not included in our implementation.

- Jumpy models: sub-sampling observations to model environment transitions at a coarser time scale, leading to a linear increase in computation time and a temporal abstraction.
- Learning to Query by Backpropagation: training the rollout policy jointly with the other parameters.
- Modulation Agent: let the agent imagine rare or even impossible trajectories, from which it can learn, eg. especially adversarial movements of ghosts.

One very important improvement is to not only pretrain the environment model, but to continue training the environment model together with the I2A model. When training the environment model, it is implicitly assumed that the environment observations used for training are representative of the environment. However in praxis these training samples are only representative of the reactions of the environment to the policy with which they were generated. This policy generally does not do well on the given task, which is why we need I2As for it in the first place. Due to this, the training samples contain few samples of situations that only arise for a *sufficiently good* agent. Exactly these situations are however in many tasks central elements of the problem. In our experiments on MsPacman, hunting ghosts is an example of such a situation. While the policy, generating the training samples, does hunt

ghosts every so often, such samples are far too seldom compared to the importance of hunting ghosts for a more sophisticated agent. Once the agent manages to clear the first level, the algorithm runs into a problem. The second level has different static features and the environment model has *never seen* this level during training, as the training sample generation agent is simply not good enough to reach the second level, meaning that environment model predictions are completely false at this point. While I2A has in theory the capabilities to overcome any environment model mistakes, it is very detrimental to learning. To solve this problem, the increasingly good I2A agent could be used during its training to generate new samples for continued training of the environment model. It would have to be evaluated if possible feedback loops pose a problem.

Another idea is to use the latent space as input for both paths, further reducing the size and increasing the training speed of the model, however possibly at the cost of frame efficiency depending on the quality of the latent space compression.

Overall latent space imagination augmented agents seem like a significant step in the direction of combining model-based and model-free approaches for large-scale problems, taking the best from both worlds. Our results on the MsPacman environment confirm the improvements of Latent Space I2A over the A2C baseline in terms of sample efficiency reported by Buesing et al. [3], as well as a significant reduction in computation complexity and runtime.

The code of our implementation in PyTorch can be found on github [4]. Included are implementations of classic I2A, Latent Space I2A and all environment models mentioned in this work.

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[3] 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.

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

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

[6] 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.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[8] 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.

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[11] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

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

[13] 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.