Implementing a (DNN) Controller with Proximal Policy Optimization using a learned World Model to solve the CarRacing_v0 task

Eva Kuth, Pia Stermann and Friedrich Heitzer

April 2021

Abstract

As our final project for the 'Deep Reinforcement Learning' course we attended in the semester break after WS 2020/21 at the University of Osnabrück, we decided to attempt a reimplementation of the World Models approach by Jürgen Schmidhuber and David Ha from 2018 [HS18] to solve the CarRacing_v0 gym Reinforcement Learning environment. We made slight modifications to the implementation proposed by the authors. In this document we describe mainly the motivation for, background and training (and training outcomes) of the Controller part of the agent architecture, for which we used a Proximal Policy Optimization algorithm instead of - as originally done with - genetic or evolutionary algorithms, and present an outlook on other existing approaches that use learned world models in the context of Deep Reinforcement Learning. *Disclaimer*: We did and will not focus on, or expect to get comparable results to the original paper but rather on establishing a working version of the models that constitute the RL agent and their respective training algorithms. Moreover, we based our attention on understanding the theoretical framework and background of the World Models approach as well as different variations of related (more or less) model-based approaches and present this here in our documentation.

Introduction

We chose to attempt a reimplementation of the "World Models" approach as presented by David Ha and Jürgen Schmidhuber in 2018 in the same-named paper [HS18], which uses different modules such as a Variational Autoencoder (VAE, the "Vision" module) to compress image frames (state observations of the environment), which are further used to train an (LSTM-)RNN with a Mixture Density Network output layer (together accounting for the “Memory” module) to make predictions about possible future frame compression codes, and a “Controller” module on top which is responsible for policy learning based on the other agent components’ outputs. See figure 1 for a visualization of the structure and organization of the different component models that make up the RL agent.

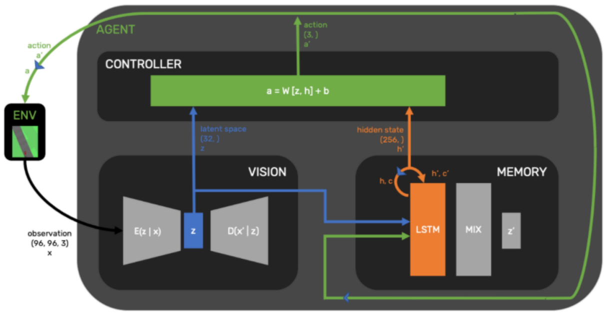


Figure 1: Structure of the World Models agent (image taken from [Gre19]

As one can see, the world models approach makes use of a variety of ANN architectures, concepts and techniques. This comes in handy, because we could well split the work of reimplementing it into two projects: For the first project we focused rather on the implementation and training of the Vision and Memory modules (as this is more related to what we learned in the IANNwTF course), while this second project deals with the implementation of the Controller and training thereof (for the DRL course).

Here, in the second project, we want to elaborate more on how the different modules are combined to make up for a working RL algorithm and discuss it in the context of related research on RL algorithms. Furthermore, we will assess its performance (compared to existing alternative approaches), possible shortcomings or limitations, and make suggestions for future adaptations.

World Models for RL

In Reinforcement Learning one can distinguish between two major classes of algorithms, namely model-based and model-free algorithms.

Generally, model-based algorithms explicitly make use of and are informed by a model of the underlying Markov Decision Process (MDP) dynamics (either learned or known a priori). Whereas model-free approaches to RL do not require and use this additional information, or at least not explicitly. World models can generally be categorized as belonging to the family of model-based RL algorithms. However, the agent here does not have access to the full description of the MDP, but can only observe the environment via interactions and from this experience construct a learned model of its environment. This is typically achieved by an augmentation of the computational agent model's internal structure with a module that can predict future states based on previous experience from interactions with the MDP.

One of the authors of the World Models paper gave a nice and compact (~14 min) description of their approach, possible extensions, limitations and related work in a talk at the Neural Information Processing Sciences Conference in 2018.

Motivation

Firstly, in section 'Personal Interest', we state how we personally came to the topic, and what made it so appealing to us that we chose to try things out ourselves. Secondly, the general motivation for the World Models approach can be regarded as twofold: One the one hand, if implemented reasonably and applied to suitable problems, the approach has certain computational advantages over its model-free alternatives (in RL) and generally addresses some common problems of model-free and model based RL algorithms. This is what we will discuss further in section 'Computational Motivation'. On the other hand, it also seems to get close to how one might believe humans to learn, which we elaborate further on in section 'Cognitive Plausibility'.

Personal Interest

The topic of world models is dominated by two researchers, namely David Ha and Jürgen Schmidhuber. Schmidhuber mentioned the concept of world models for the first time in his report "Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments" in the year 1990 [Sch90]. His recent work with David Ha of Google (2018) [HS18] can be seen as probably the first or at least highly influential implementation of a World Models agent. We personally touched upon this topic in the last part of the course "Deep reinforcement learning" for the first time. From the very moment we saw the introduction video about this topic, we wanted to work on world models in our final project. How exactly is it possible to build software agents to behave like a human? Can computers build something like common sense or proper real-life knowledge? Furthermore, the video encourages thoughts about our own mental model, emphasizing its existence and trying to get to the bottom of the workload done internally in everyone's brain. Thinking about the vicinity this reinforcement learning approach has to real life makes it tangible but still unapproachable. The way three fairly simple networks are able to mimic this complex interaction is really fascinating.

Computational Motivation

Solving common RL problems - a computational perspective World Models are an approach to tackle some major flaws which algorithms belonging to either of the two classes of RL algorithms mentioned above typically exhibit. Which problems these are specifically and how world models can help to overcome them is what we will elaborate on in the following.

Model-based problem: We do not have access to complete knowledge of the environment

On the one hand, traditional model-based RL algorithms - i.e. those that have access to a complete description of the environment dynamics - might exhibit extraordinary performance over the domain of problems where the environment dynamics can be described and accessed by a computational agent completely. However, they are not generally applicable to any given problem, i.e. when the models are too complex to define explicitly or simply not known. World Models can be regarded as belonging to the class of model-based RL algorithms, but instead of requiring complete a priori information about the environment dynamics, an environment dynamics model, the so called world model, is learned iteratively during or prior to training the RL agent as a part of the computational agent's architecture. The task of this world model - typically implemented via a neural network (NN) which incorporates some recurrent neural network (RNN) variant - is then to predict a representation of the future environment state(s) given an action taken by the agent at the current state of the world (or a compressed representation thereof) and the past state(s) of the RNN. Thereby, i.e. by introducing the task of constructing or approximating a proper model of the environment to the algorithm, the World Model approach is more generally applicable to RL problems than other traditional model-based approaches.

Model-free problem: We want the agent to learn quickly, i.e. generalize well from few real interactions with the environment

On the other hand, agent models for model-free RL algorithms such as PPO or Actor-Critic Methods do not contain structures which are intended to explicitly capture knowledge about the environment dynamics, making them at least as generally applicable as the (learned-world-)model-based alternatives. However, a major downside of these algorithms is that they usually require lots of interaction (trajectories) data from within the actual problem-specific environment (much more than humans would) in order to enable an agent to perform well over a given task (Kaiser et al., 2020 [Kai+20]). One possible reason for our human ability to learn so fast might be that we do not only make decisions about actions based on current and immediate observations of our surroundings, but more or less explicitly also use our memory of past experiences to predict the outcomes of an action on our surroundings, thereby predicting observations we will likely make in the future (c.f. Hafner et al., 2018 [HS18]). In most of the approaches which make use of learned World Models, the problem of sampling-data inefficiency can be overcome, because World Models enable to capture state-action-transition probabilities over time (Weber et al., 2018). Thus, a World Model can be used to train the RL agent within a simulation of the real environment, just as humans might form decision making policies by imagination or when dreaming (cf. [Haf+20a]). Simulation of the environment in World Model approaches is typically accomplished by using representations of future state observations (given an initial input observation) generated by the learned world model only (cf. [Haf+20a],[HS18]). Hybrid approaches have also been proposed, in which model-free RL and learned world model networks are combined with model predictive control (MPC) strategies to increase learning efficiency, e.g. by Nagabandi et al. in 2017 [Nag+17].

General problem: Dealing with partially observable environments (given limited computational resources)

Another more general issue in RL comes with partially observable MDPs (POMDP). In POMDPs, a single observation of the environment does not capture all information about the actual environment state. According to Jürgen Schmidhuber, "realistic environments are not fully observable". Thus, according to Jürgen Schmidhuber, "General learning agents need an internal state to memorize important events in case of POMDPs". And exactly this is what the world model approach tries to accomplish. We know that we have only limited computational resources to train a RL agent to behave intelligently in an initially unknown environment (i.e. to learn a good policy), but we also know that the given task is a POMDP problem. Having an agent learn a policy as with standard policy-gradient based methods from - in this case - raw spatial data (state observations) and information from previous timesteps would require lots of computational resources due to large numbers of trainable parameters in and end-to-end training of the agent model. In the world models approach, learning a good compression of inputs over space and time (which might require a decent amount of parameters in the model to be tuned) is the main purpose of the world model module in the agent. By decoupling this task from learning the actual policy, the whole problem can be solved more efficiently (regarding the required number of sampled interactions of the agent with the real environment).

Cognitive Plausibility

Apart from computational aspects, world models are also of theoretical interest regarding their similarities to how we believe humans to make decisions in real life. As humans we are able to perceive our surroundings through many different senses. By interacting with the environment, we further learn to predict the consequences of our actions. Everything we perceive from observations or through third parties is taken in, understood and reaches our brain. With that ability, we create a mental model which represents this environment, how it works and behaves in certain situations. Importantly, the mental model is based on beliefs and interests, not facts. It can and does get updated to simplify the complexity of the whole world into understandable chunks. Usually, all this happens within seconds and with the agent - i.e. ourselves - not even being aware of it.

In addition, this mental model now can be used to learn from it. Because the mental model knows the behaviour and can predict several states, the agent does not need to perceive and consciously plan every action. Think about the following two examples.

First, imagine standing on skis for the first time in your life. It will most likely feel really weird, unstable and just new for you. In that very moment, your mental model of skiing has just started to develop. The first couple of times when you get on your ski you will have to learn how the snow behaves in interaction with your ski, what will further happen when you ski at that specific speed and angle. During this process of learning, your mental model becomes clearer and more extensive. Finally, after some time you will not have to consciously think about what will happen in which specific moment. Often, there might not even be time to think things through, you will just have to react and interact with the environment. Now, this environment is fully represented within your mental model and you gained the ability to use that imagined model to learn your behaviour.

The second example is concerned with artificial curiosity, which helps to learn how the real world works. To build an extensive mental model, humans need new challenges and new input, which is one of the goals of curiosity. However, while building a mental model other goals are present as well. Although one might be curious what will happen when touching a high voltage line and has never seen anybody do it, the already existing mental model will prevent doing it. One does not exactly know how it would feel, but at the same time does not want to know. The mental model already helps with predicting the consequences of certain actions and becomes more complete through curiosity.

World models are now exactly this, including two major components. Firstly, the vision part, for instance the human's eye to perceive, encode and reduce complexity of the environment. And secondly, the memory part to integrate the knowledge and create the mental model, which could be represented by the human's brain. The internal world model is now followed by a controller to select the best action based on knowledge from the world model, in real-world applications the controller would be the human's body.

Implementation and Results

We will proceed by describing the proposed implementation of the world models approach (task, agent structure and training procedure) by Schmidhuber and Ha [HS18], and more specifically what we effectively did for our reimplementation. Firstly, we briefly describe what the task was for the agent. Afterwards, we will elaborate more on the agent structure, where we briefly describe the World Model part (Vision and Memory Module). For more detailed description of these, please refer to the documentation of our first project, where we described V and M in more detail. Contrary, we will elaborate here more on the Controller architecture and training thereof using Proximal Policy Optimization. Furthermore, we will discuss the final performance of the trained agent on the task against the background of related literature.

The Task

The world models agent was originally tested on a (continuous control) Car Racing and Doom task, both of which are available as environments in OpenAI's gym framework [Ope]. For the scope and purpose of our project(s), we will only focus on the specific application of the approach to the CarRacing-v0 environment. In this environment, the agent has to steer a race car over a road as fast as possible within a fixed time. The tracks are randomly generated for each trial, and the agent is rewarded for visiting as many tiles as possible in the least amount of time. The agent controls three continuous actions: steering left/right, acceleration, and brake.

The Agent

Generally, the procedure of training an RL agent (as shown previously in figure 1 and in figure 2 with the world models approach is as follows:

1. Sample trajectories (i.e. raw spatial and temporal information) from random interactions with the environment.
2. Train the "Vision" module (a Variational Autoencoder) to compress image pixel-frames into latent vector representations.

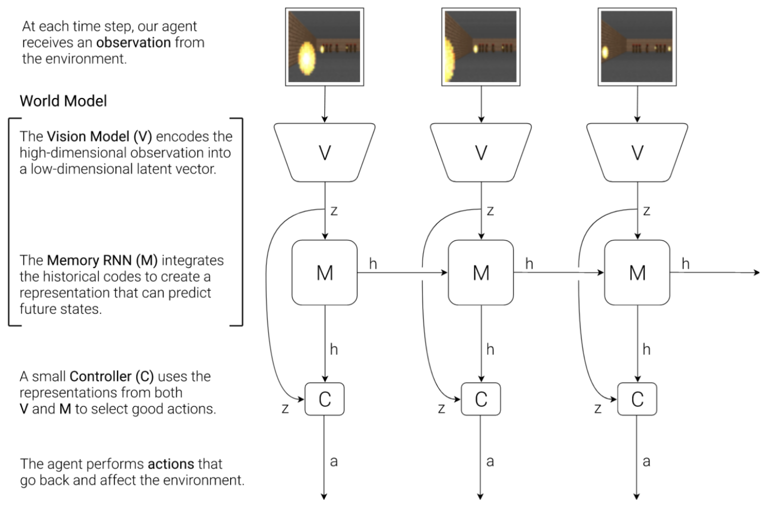

At each time step, our agent receives an **observation** from the environment.
World Model
The **Vision Model (V)** encodes the high-dimensional observation into a low-dimensional latent vector.
The **Memory RNN (M)** integrates the historical codes to create a representation that can predict future states.
A small **Controller (C)** uses the representations from both **V** and **M** to select good actions.
The agent performs **actions** that go back and affect the environment.
V
z
M
h
C
a

Figure 2: Training of the World Models agent (taken from [HS18])

3. Train the "Memory" module (a MDN with LSTM head) to learn environment dynamics and predict the next state's latent vector representation given the VAE-encoded state observations from the trajectory samples, and the corresponding action taken by the random agent.

4. Train the "Controller" module (originally using evolutionary algorithms). This can be done in the real or simulated environment using the Memory module to predict state-action transitions.

We will describe how we dealt with the these steps - especially the first and last step- in the following paragraphs using the previously described car racing task. The second and third steps are described in more detail in our final project documentation for the "Introduction to Artificial Neural Networks with TensorFlow" course.

Step 1: Generating Trajectories The goal is to here learn a good spatial and temporal representation of data that we can pass as input to the Controller (the policy-learning) module of the agent. To do so, we need to collect raw data of state-action-transition observations from interactions with the environment first. This data is used to train the world model's (=compression) modules (VAE and Memory) on. Therefore, the world models approach starts with having an initial agent repeatedly take random actions in the environment (until a total of 10000, these so-called "random-rollouts" of the environment are collected). Each raw state observation is a frame of 64x64 pixels, which is quite a lot of data to process at each single timestep. In fact, it is possibly more than effectively needed to capture the features of a state that are relevant or decisive for solving a given task. Passing them as input information to a policy-learning module might be overwhelming for the agent assuming it stays unfiltered.

Recognizing that our agent cannot be trained on the full-sized input observations, we next need a method for spatial compression of the raw input observations (i.e. pixel frames).

Step 2: World Model - Vision The authors' method of choice here was to make use of a variational Autoencoder (VAE) NN model architecture, which we got to know in the IANNwTF course as well. We adopted the implementation details about the specific encoder-decoder architecture structure as it was described in the paper.

The VAE/Vision/V model as the first part of the agent also constitutes the first part of what is called the World Model. A raw input observation enters the final agent model through V, which builds a compressed form of the multi dimensional input. As stated above, the necessity arises due to the agent not being able and usually not needing to cope with too high dimensionality. The VAE allows our agent (i.e. the memory and controller parts) to receive the lower dimensional, abstract and latent representation of the observation constructed by the VAE. Moreover, the V model never receives any rewards, thus learns only from observations of the environment and in interaction with the other parts of the agent.

Step 3: World Model - Memory Secondly, we want to incorporate data from the history of previous timesteps to account for the POMDP aspect of the problem. Because it would not be computationally and memory-wise feasible (and neither conceptually desirable) to keep all previous input observations in memory and pass them as input to the "decision-making part" of the agent, aka Controller, at each timestep, we also want to compress the data along the time dimension.

Thus, the second component of the agent is the memory module (M), which consists of a recurrent neural network (RNN), more specifically an LSTM, with a mixture density network (MDN) output layer. During training, the RNN gets sequential input in the form of a compressed state observation (z) (sampled from the VAE training results) and a corresponding action a which had been taken by the agent in that state, and outputs a set of values that can further be passed to the MDN layer to produce the parameters of $|z|$=32 Gaussian Mixture probability distributions with 5 kernels each, from each of which (i.e. technically for each dimension of z) we can sample a single value to get a prediction for a possible next compressed state observation for each dimension of the next latent state representation (z').

The goal of M is thus to learn a probabilistic model capturing the environment dynamics, which should allow the agent to make better (informed) decisions. Specifically, it should learn to predict the consequences of the agent's actions on the environment and encode transition probabilities for given states and actions, given the sequence of previous states and actions in a trajectory. The hidden layer of the recurrent neural network should finally contain all relevant information about the current situation and many future steps to facilitate policy learning for the Controller.

The second input to the Memory module as described in the paper is a temperature parameter, which is able to control the certainty of the network over the predicted future states by introducing noise to the network's predictions. Increasing the parameter when sampling the next latent representation leads to higher uncertainty, making the environment less predictable. For sampling from the MDN output parameters during training of the Controller network later, we used a temperature value of 1.15. However, this is not used for training the Memory itself, because we do not need to sample predictions for this but just take the (negative-log-)likelihood derivable from the "raw" gaussian mixture parameters (i.e. the memory model's outputs) to calculate the memory network's error. The parameters for the Gaussian Mixtures as output of the Memory model can afterwards be used to sample a possible z' from the distribution they define if we want to train an agent inside a simulated or "dream" environment, and also pass a temperature parameter to the mdn layer's sampling function we use to do so when training the Controller. Theoretically we could even adjust it dynamically to vary the amount of stochasticity or noise in the memory model's predictions and avoid the Controller from learning behavior that exploits imperfections of the World Model and only thereby achieve high scores or rewards in the simulated environment.

The final Memory model and training code was then derived from our model definition in our development and testing notebook, the proposals from the original paper by Ha and Schmidhuber[HS18], and a well documented reimplementation of the world models approach by [Gre19]. We trained our memory model on the results from the sampled trajectories and the respective VAE outputs as described by Ha and Schmidhuber [HS18] and done by Green [Gre19].

Figure 3 shows some decoded state observation representations from predictions of future latent vectors for a single step into the future. We recognize that our model predictions seem quite imperfect when compared to the corresponding actual decoded state representations by the VAE but also seem somewhat reasonable as state observations.

Step 4: Controller - PPO The third component of the agent is the Controller C, which however is not part of the world model. Its purpose is to select a specific task in interaction with the environment. Which action vector the

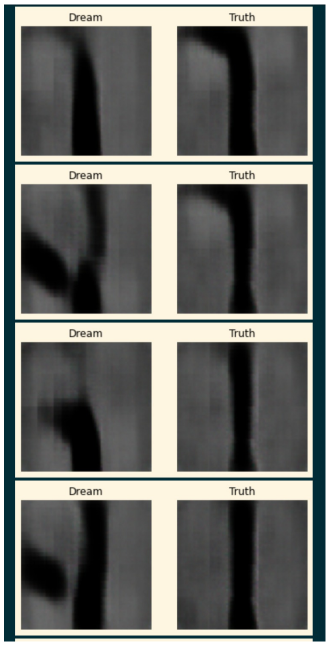


Figure 3: Memory visualizations

controller is going to produce, i.e. which action the agent is going to take is dependent on the world model consisting of V and M. Here, we receive the latent representation vector from the Vision Model and the hidden state of the Memory, an internal representation of what the M model has learnt. Only the Controller is able to access any rewards from the environment, which it is expected to maximize in each time step. However, even the C model can be trained without taking the action in the real world, due to the internal world model built. In the original paper C is represented as a small, one-layer linear model which maps input to output for every new time step.

Proximal Policy Optimization Different from the covariation matrix adaptation evolution strategy utilized to optimize Controller Model in WM, we make use of PPO based Actor Critic (AC) algorithm in the Controller. Furthermore, an attention mechanism is considered in the critic network for estimates of state value function. In AC algorithms, the critic network generally works prior to the actor network, since precise estimation of value function can better accelerate policy learning.

Proximal policy optimization algorithm is an advancement of the normal Trust Region Policy Optimization algorithm. However, it still tries to solve the same objective, namely the changes of the stationary distribution not changing too much. To realize this, one uses the so-called trust region, defined as the region where the new updated policy cannot change too much during one step. The usability of a PPO as compared to TRPO algorithm increases, as it is more stable and tries to apply multiple update steps from samples created from the current policy. It manages to remove the Kullback-Leibler divergence, which led to trouble in the TRPO approach. In addition, PPO eliminates the request to make adaptive updates. In PPO algorithms, the probability ratio $r_t(\theta)$ between the new updated policy outputs and the outputs of the previous version is therefore used to measure the difference between the two policies.

$$r_t(\theta) \text{ is defined as } \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\right]$$

If the new policy is too far away from the old one, i.e. it falls out of the trust region, the advantage function A will be clipped. So, one can decide on a proper trust region in which the update can be done. Furthermore, PPO algorithms work with taking the minimum of

1. Objective of normal policy gradients, responsible to push policy action that yield a high positive advantage over the regular baseline

   –> probability ratio multiplied with advantage

2. Similar to first, but truncated probability ratio results by clipping

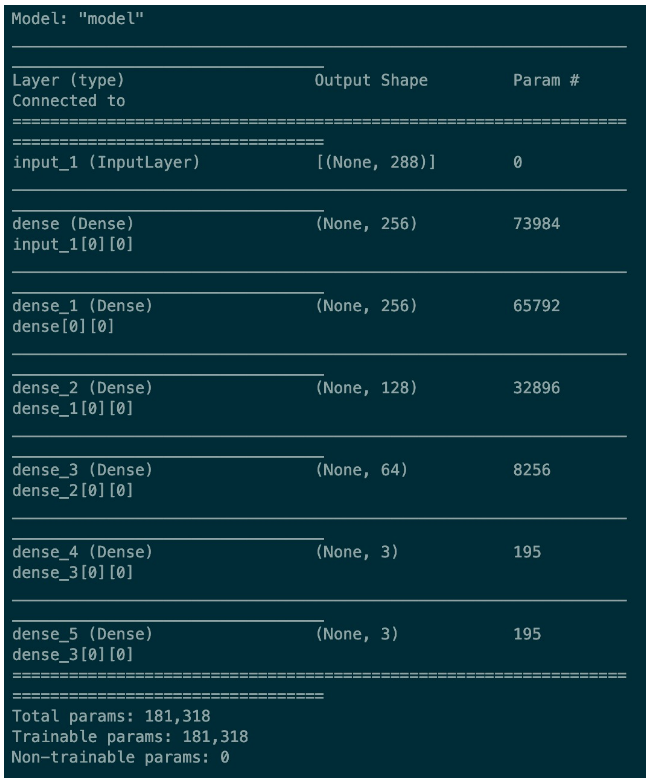
   –> clipped probability ratio multiplied with advantage

The corresponding formula for the objective function looks like this (usually epsilon is 0.2):

$$L^{CLIP}(\theta) = E[\,\min(\,r_t(\theta)\,A_t(s,a),\; clip(\,r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t(s,a)\,)\,] \qquad (1)$$

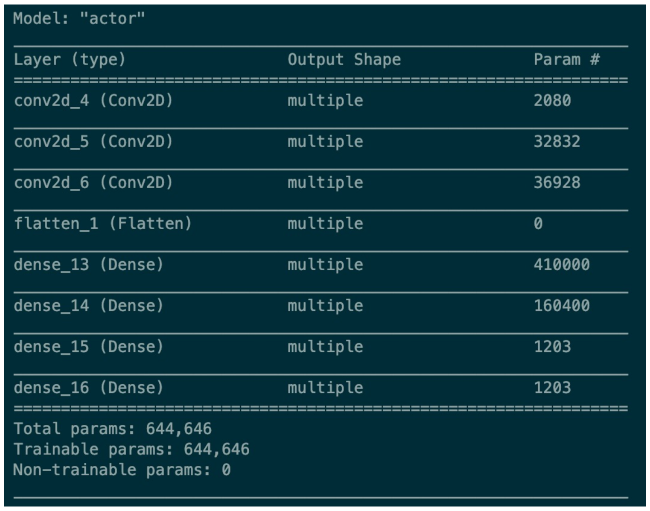
Controller Model Summaries

Finally, we present a summary of the number of parameters in our implementation of the Controller networks in figures 4, 5, 6. Figure 4 shows the summary of the actor Controller, the actor is responsible to select good actions. One can see that due to several dense layers, exactly 181,318 trainable parameters are involved. The whole model code can be seen in the "controller" section of our world model implementation. There, you can also find the so-called vanilla PPO, which is trained directly in the environment without the use of a previous world model. Summary of its actor (figure 5) and critic (figure 6) together consist of almost 1,300,000 trainable parameters.



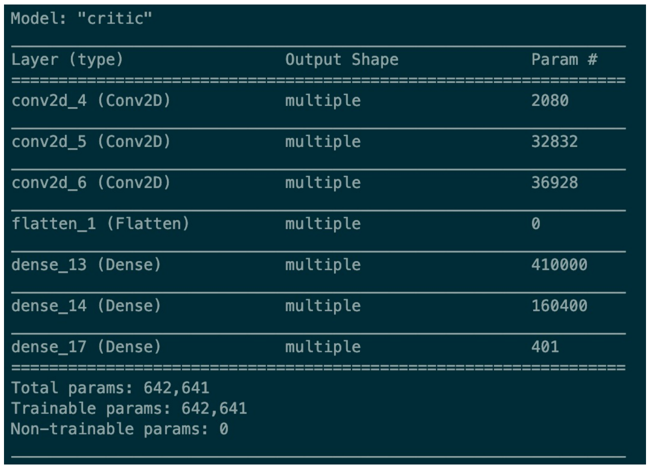
```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 288)]        0
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          73984       input_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          65792       dense[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 128)          32896       dense_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 64)           8256        dense_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 3)            195         dense_3[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 3)            195         dense_3[0][0]
==================================================================================================
Total params: 181,318
Trainable params: 181,318
Non-trainable params: 0
```

Figure 4: Actor Controller world model summary



```
Model: "actor"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            multiple                  2080
_________________________________________________________________
conv2d_5 (Conv2D)            multiple                  32832
_________________________________________________________________
conv2d_6 (Conv2D)            multiple                  36928
_________________________________________________________________
flatten_1 (Flatten)          multiple                  0
_________________________________________________________________
dense_13 (Dense)             multiple                  410000
_________________________________________________________________
dense_14 (Dense)             multiple                  160400
_________________________________________________________________
dense_15 (Dense)             multiple                  1203
_________________________________________________________________
dense_16 (Dense)             multiple                  1203
=================================================================
Total params: 644,646
Trainable params: 644,646
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Vanilla PPO actor model summary



```
Model: "critic"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            multiple                  2080
_________________________________________________________________
conv2d_5 (Conv2D)            multiple                  32832
_________________________________________________________________
conv2d_6 (Conv2D)            multiple                  36928
_________________________________________________________________
flatten_1 (Flatten)          multiple                  0
_________________________________________________________________
dense_13 (Dense)             multiple                  410000
_________________________________________________________________
dense_14 (Dense)             multiple                  160400
_________________________________________________________________
dense_17 (Dense)             multiple                  401
=================================================================
Total params: 642,641
Trainable params: 642,641
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Vanilla PPO Critic model summary

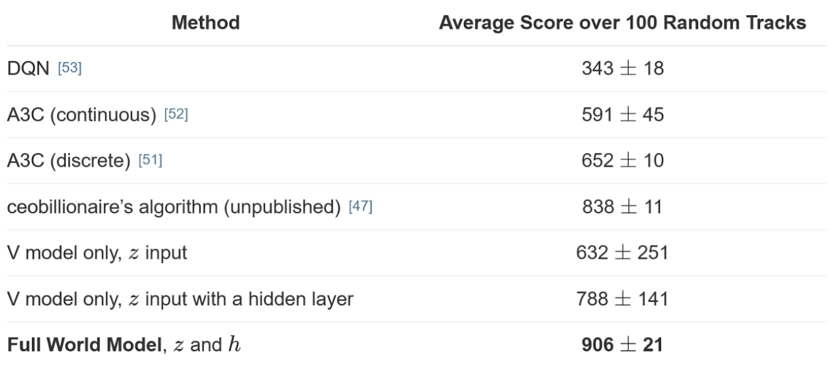
Discussion of Historical Background, Optimization, Related Work and Outlook

Historically, the notion of world models in the context of Reinforcement Learning(RL) algorithms dates back to the 1990s. In "Making the World Differentiable" by Schmidhuber [Sch90], RL and planning are based on a combination of a Controller RNN and a World Model RNN, where the world model learns to predict the consequences of actions taken by the controller. The controller in turn should learn to use these predictions to plan several timesteps ahead and select the best future action sequences based on the predicted reward.

At the time the paper was published, the authors declared that the approach was the first one - to their knowledge - that can solve the CarRacing_v0 task, reaching an average score of about 900 over 100 random tracks (cf. figure 7).

For more complex environments, where initial state observations from random actions might not be enough to explore and approximate the whole environment dynamics, Schmidhuber and Ha propose an iterative training procedure as "Iterative training could allow the C—M model to develop a natural hierarchical way to learn. Recent works about self-play in RL and PowerPlay also explores methods that lead to a natural curriculum learning", according to the interactive version of the "World Models" paper. An iterative training procedure, adapted from "Learning To Think" [Sch15], should work as follows:

1. Initialize M, C with random model parameters.
2. Rollout to actual environment N times. The agent may learn during roll-



Method	Average Score over 100 Random Tracks
DQN [53]	343 ± 18
A3C (continuous) [52]	591 ± 45
A3C (discrete) [51]	652 ± 10
ceobillionaire's algorithm (unpublished) [47]	838 ± 11
V model only, z input	632 ± 251
V model only, z input with a hidden layer	788 ± 141
Full World Model, z and h	**906 ± 21**

Figure 7: World Model agent performance against alternative approaches for the CarRacing_v0 task (table taken from [HS18])

outs. Save all actions a_t and observations x_t during rollouts to storage device

3. Train M to model P(t_1,r_{t+1},a_{t+1},$d_{t+1}x_t$, a_t,h_t).
4. Train C to optimize expected rewards inside of M.
5. Go back to (2) if the task has not been completed yet.

After engaging with the topic of world models, we came across some other recent papers building up on the work by Schmidhuber and Ha. Due to the breakthrough by those two authors taking place only three years ago in 2018, the field is still very dynamic and subject to major changes. Several variants of learned-world-model-based RL algorithms or such that interpolate between model-free and model-based policy learning strategies (so called "hybrid" approaches) to solve problems from different domains more efficiently were introduced within the past decade, with promising results as well. Hybrid strategies include alternating between using world model predictions and model-free methods for policy learning as in [Pan+20], or using short model-generated rollouts branched from real data for increased sampling-efficiency [Jan+19]. In 2018, Chua et al.[Chu+18] introduced an approach in which a dynamics model is trained to predict the distribution over state-trajectories resulting from applying a sequence of actions. By computing the expected reward over state-trajectories, multiple candidate action sequences are evaluated, and the optimal action sequence to use is selected. Other recent work also showed that using world models for solving Atari, Chess, Go and similar (also previously unsolved) RL problems/environments can lead to state-of-the-art or even better performance compared to existing model-free alternatives like Rainbow or PPO based learning algorithms in the Atari game domain (figure 8, cf. Hafner et al., 2020 [Haf+20b] which we will elaborate on further in the following paragraph; or

Schrittwieser et al., 2020 [Sch+20]). There exist many more interesting extensions to the world model based approach to RL, with more being published the moment we speak.

One of these variants we deemed to be worth mentioning is the Dreamer v2 world model which was published in the paper "Mastering Atari with discrete World Models"[Haf+20b] in December 2020. The authors Hafner, Lillicrap, Norouzi and Ba work for instance for Google Brain and DeepMind. Here, a model-based and single-GPU reinforcement learning algorithm is able to successfully outperform many relevant model-free RL algorithms in Atari games (figure 8).

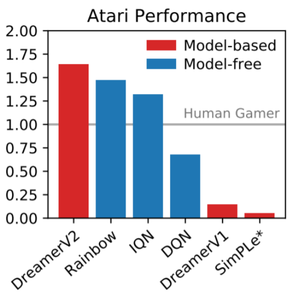


Figure 8: DreamerV2 Performance (taken from [Haf+20b])

As any world model, Dreamer v2 learns how the world acts and uses that model to learn what actions to perform. This model learns the world model from past steps, by starting to feed each new observation into an autoencoder. But here is already what depicts the major difference, as the latent state representation between encoder and decoder has a special constraint. Given an observation, a latent state and an action, we are trying to predict a reward and the image of the game after performing the action. Special about Dreamer v2 is now the definition of the hidden states h representing the latent representation vector. Usually, the hidden states of a world model are fixed as containing gaussian latent variables. In the present approach, those hidden states are fixed as a combination of a deterministic state and a stochastic state (see figure 9). Together, each hidden state and input image x is going to predict a stochastic description of the current state, defined as z. This z is foremost predicted from the hidden state h. Z is a collection of categorical variables, each with 32 possible classes, whereas values of variables and classes are to be chosen freely by the model. Categorical distributions are other than gaussian dynamics very flexible and thus able to accurately predict multimodal distributions. As mentioned, the input image contributes to building stochastic state z. By implication, this means that every observation which enters the network is going to be described by categorical variables. Leading to the input observations being compressed and thus resolving in better latent variables. Furthermore, apart from the stochastic state z being predicted from the deterministic state h, it also touches upon h later. Specifically, samples from the categorical distribution of the stochastic state are concatenated with the deterministic hidden state to build the final latent state.

Because in world model reinforcement learning one wants to detach from the real world, there is a tiny addition to make the observation predictable. To do so, each hidden state is used to construct a second, desirably same $\hat{z}$ variable, without the help of the current observation (see both figure 9 and figure 10). In addition, a loss function is introduced between posterior stochastic state z and prior stochastic state $\hat{z}$. It controls the tradeoff between training the prior $\hat{z}$ and checking how much information the posterior z incorporates from the observation. Effectively, the world model is now able to be learned and improved by receiving one observation only (as shown in figure 10). The rest of the agent's reinforcement learning occurs in the completely imaginary space. In the present paper, the agent learns its behavior using an advanced actor-critic algorithm.

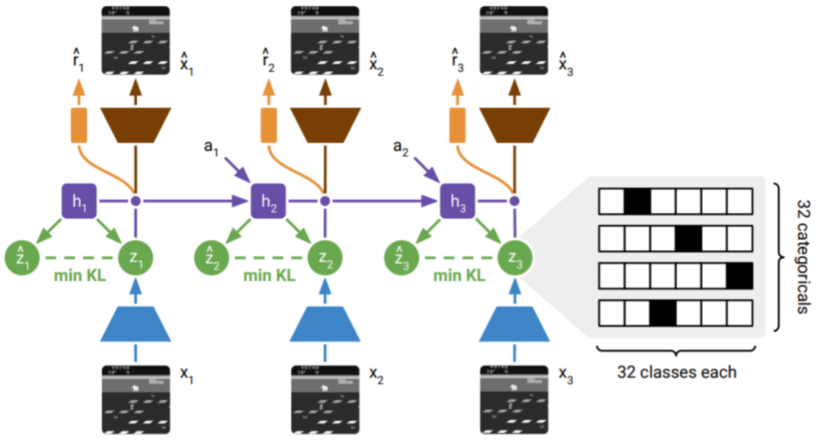


Figure 9: Dreamer v2 World Model from [Haf+20b]

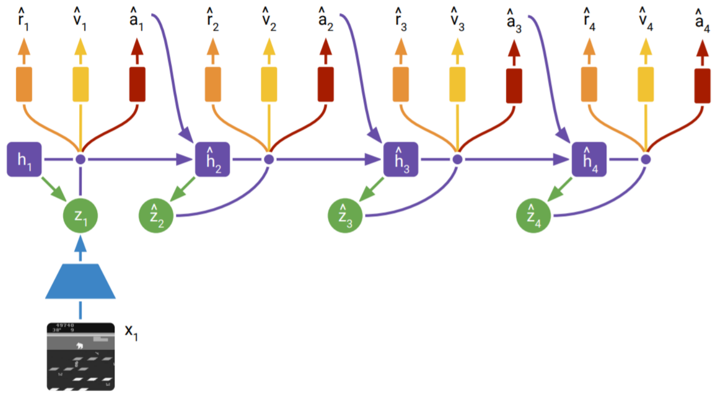


Figure 10: Actor critic of Dreamer v2 World Model from [Haf+20b]

References

[Sch90] Jiirgen Schmidhuber. “Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environm~nts”. In: (1990), p. 28.

[Sch15] Juergen Schmidhuber. “On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models”. In: *arXiv:1511.09249 [cs]* (Nov. 30, 2015). arXiv: 1511.09249. URL: http://arxiv.org/abs/1511.09249 (visited on 03/05/2021).

[Nag+17] Anusha Nagabandi et al. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *arXiv:1708.02596 [cs]* (Dec. 1, 2017). arXiv: 1708.02596. URL: http://arxiv.org/abs/1708.02596 (visited on 04/03/2021).

[Chu+18] Kurtland Chua et al. “Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models”. In: *arXiv:1805.12114 [cs, stat]* (Nov. 2, 2018). arXiv: 1805.12114. URL: http://arxiv.org/abs/1805.12114 (visited on 04/03/2021).

[HS18] David Ha and Jürgen Schmidhuber. “World Models”. In: *arXiv:1803.10122 [cs, stat]* (Mar. 28, 2018). DOI: 10.5281/zenodo.1207631. arXiv: 1803.10122. URL: http://arxiv.org/abs/1803.10122 (visited on 03/02/2021).

[Gre19] Adam Green. *World Models (the long version)*. The intersection of energy & machine learning. Dec. 21, 2019. URL: https://adgefficiency.com/world-models/ (visited on 04/17/2021).

[Jan+19] Michael Janner et al. “When to Trust Your Model: Model-Based Policy Optimization”. In: *arXiv:1906.08253 [cs, stat]* (Nov. 4, 2019). arXiv: 1906.08253. URL: http://arxiv.org/abs/1906.08253 (visited on 04/17/2021).

[Haf+20a] Danijar Hafner et al. “Dream to Control: Learning Behaviors by Latent Imagination”. In: *arXiv:1912.01603 [cs]* (Mar. 17, 2020). arXiv: 1912.01603. URL: http://arxiv.org/abs/1912.01603 (visited on 03/15/2021).

[Haf+20b] Danijar Hafner et al. “Mastering Atari with Discrete World Models”. In: *arXiv:2010.02193 [cs, stat]* (Dec. 22, 2020). arXiv: 2010.02193. URL: http://arxiv.org/abs/2010.02193 (visited on 03/15/2021).

[Kai+20] Lukasz Kaiser et al. “Model-Based Reinforcement Learning for Atari”. In: *arXiv:1903.00374 [cs, stat]* (Feb. 19, 2020). arXiv: 1903.00374. URL: http://arxiv.org/abs/1903.00374 (visited on 03/15/2021).

[Pan+20] Feiyang Pan et al. "Trust the Model When It Is Confident: Masked Model-based Actor-Critic". In: *arXiv:2010.04893 [cs]* (Oct. 9, 2020). arXiv: 2010.04893. URL: http://arxiv.org/abs/2010.04893 (visited on 04/17/2021).

[Sch+20] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *Nature* 588.7839 (Dec. 24, 2020), pp. 604–609. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-020-03051-4. arXiv: 1911.08265. URL: http://arxiv.org/abs/1911.08265 (visited on 03/15/2021).

[Ope] OpenAI. *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. URL: https://gym.openai.com (visited on 04/16/2021).