

RL: Reinforcement Learning

Sample efficiency I: Model Based Reinforcement Learning (MBRL)

Mario Martin

CS-UPC

April 25, 2026

Motivation

RL efficiency

- We usually distinguish between wall-clock efficiency (on-policy PPO) and sample efficiency (off-policy SAC)
- In benchmarks, we usually compare reward with respect to steps or episodes (experiences)
- In RL usually we need a lot of interactions with the environment (order of millions of steps) in order to learn
- When a simulator is available this is usually not a problem, but when interactions are in the real world, it is a problem
- This fact limits the application of RL to real cases

Sample efficiency

- There can be a lot of reasons where a simulator is not used: Not reliable, not available, too slow, etc.
- Even in cases where simulator is available, it is desirable to learn as fast as possible
- So, we will focus in **sample efficient** methods, that is, methods that allow to learn with fewer interactions with the environment than standard RL algorithms (need less samples)
- Several ways to do that. We will focus today in Model based RL (MBRL) techniques.

Model based Reinforcement Learning

- RL methods seen until now are **Model-free**
- Model-free methods do not need to previously know or explicitly learn transition probabilities ($P_{s,a}^{s'}$) and reward function ($R(s, a)$)
- Why not?
- Q-values and policies implicitly incorporate transition probabilities

$$\begin{aligned} Q(s, a) &= \mathbb{E} \left[r(s, a) + \gamma \cdot \arg \max_{a'} Q(s', a) \right] \\ &= r(s, a) + \gamma \sum_{s'} P_{s,a}^{s'} \cdot \arg \max_{a'} Q(s', a) \end{aligned}$$

- Also notice that *Experience Replay buffer* has information about transition (s, a, r, s')

Model based Reinforcement Learning

- So, do we need models?
- Benefits of having a model of the world:
 - ▶ We can plan
 - ▶ We can learn from complete backups instead of samples
 - ▶ We can imagine trials and learn from them
 - ▶ We can reuse the info to other tasks in same environment
 - ▶ Help for Exploration
 - ▶ Speed to adapt to changing rewards
 - ▶ Speed to adapt to changing dynamics
 - ▶ ...
- Goal is sample efficiency

Model based Reinforcement Learning

- Other reasons to use a model:
 - ① To expensive or risky to use real world (f.i. robots)
 - ② Simulating complex physical dynamics is too expensive
 - ③ When interacting with humans
 - ④ ...

Model based Reinforcement Learning

- Other reasons to use a model:
 - ① To expensive or risky to use real world (f.i. robots)
 - ② Simulating complex physical dynamics is too expensive
 - ③ When interacting with humans
 - ④ ...
- So, questions:
 - ① What is exactly a model?
 - ② How can we learn a model?
 - ③ How can we take profit of a model?
 - ④ Which are the problems with models

Model definition

What is a model?

Model Definition

A model is a representation that **explicitly** encodes knowledge about the structure of the environment and task:

- **A transition/dynamics model:**

$$s_{t+1} = f_s(s_t, a_t)$$

- **A model of rewards:**

$$r_{t+1} = f_r(s_t, a_t)$$

- In some cases: An inverse transition/dynamics model:

$$a_t = f_s^{-1}(s_t, s_{t+1})$$

What is a model?

- Sometimes we know the ground truth dynamics or rewards. Might as well use them!
- In general not known and **must be learnt**
- This is a supervised learning problem: given a dataset:

$$\begin{aligned} S_1, A_1 &\rightarrow R_2, S_2 \\ &\vdots \\ S_{T-1}, A_{T-1} &\rightarrow R_T, S_T \end{aligned}$$

- learn applying any SL technique.

About the input to the model

- How do we obtain the data?
 - ▶ Random movements before building the model
 - ▶ From a initial fixed policy?
 - ▶ From Experience Replay while learning?
- Problem of coverage of the dataset!
- RL is interactive, use data collected to improve the model

About the input to the model

- How do we obtain the data?
 - ▶ Random movements before building the model
 - ▶ From a initial fixed policy?
 - ▶ From Experience Replay while learning?
- Problem of coverage of the dataset!
- RL is interactive, use data collected to improve the model
- Representation? Guess!...

About the input to the model

- How do we obtain the data?
 - ▶ Random movements before building the model
 - ▶ From a initial fixed policy?
 - ▶ From Experience Replay while learning?
- Problem of coverage of the dataset!
- RL is interactive, use data collected to improve the model
- Representation? Guess!... Yes DNNs... but sometimes other approaches as bunch of equations (and determination of parameters) or Gaussian mixtures, etc.
- Remember that model is an approximation (and it can fail!)

About the input to the model

- I know. I said state and action:

$$s_{t+1} = f_s(s_t, a_t)$$

$$r_{t+1} = f_r(s_t, a_t)$$

About the input to the model

- I know. I said state and action:

$$s_{t+1} = f_s(s_t, a_t)$$

$$r_{t+1} = f_r(s_t, a_t)$$

- ... but in some cases not possible or desirable

About the input to the model

- I know. I said state and action:

$$s_{t+1} = f_s(s_t, a_t)$$

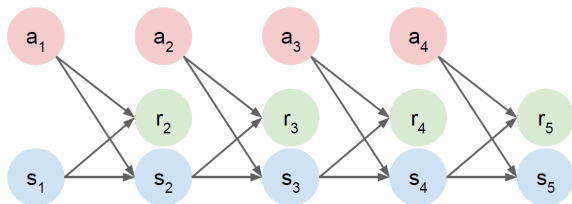
$$r_{t+1} = f_r(s_t, a_t)$$

- ... but in some cases not possible or desirable
- Usually three options:
 - ▶ States
 - ▶ Observations
 - ▶ Latent States

Input is state

- Usually we use a DNN to predict next state.

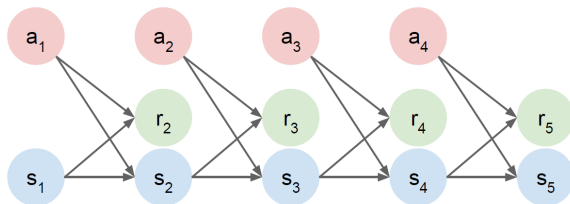
$$s_{t+1} = f_s(s_t, a_t), r_{t+1} = f_r(s_t, a_t)$$



Input is state

- Usually we use a DNN to predict next state.

$$s_{t+1} = f_s(s_t, a_t), r_{t+1} = f_r(s_t, a_t)$$



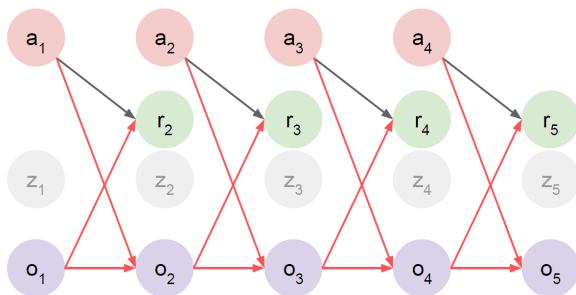
Practical trick: typically better to predict the derivative (change in s), and then integrate to obtain s_{t+1}

$$s_{t+1} = s_t + f(s_t, a_t)$$

Input is observation

- In some cases, agent has Partial Observability and no access to true state

$$o_{t+1} = f_s(o_t, a_t), r_{t+1} = f_r(o_t, a_t)$$

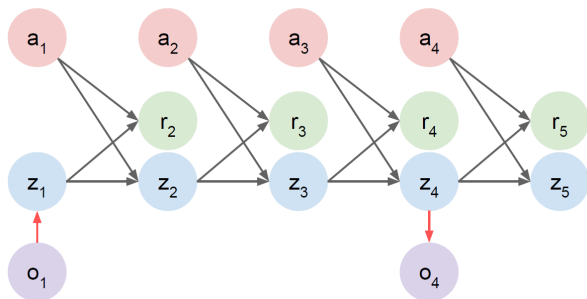


Input is latent state

- In some cases, observation is too complex to do predictions (f.i from pixels of an image)

$$z_{t+1} = f_s(z_t, a_t), r_{t+1} = f_r(z_t, a_t)$$

$$z_t = Enc(o_t), o_t = Enc^{-1}(z_t) = Dec(z_t)$$



Nice [interactive paper](#) on latent space

What does the model predict?

- Models that return the **expectation of next state**
- Goal: estimate model \mathcal{M}_η from experience $\{S_1, A_1, R_2, \dots, S_T\}$ that computes function $f(s, a) = r, s'$
- Pick loss function (e.g. mean-squared error), and find parameters η that minimise empirical loss
- This would give an expectation model If $f(s, a) = r, s'$, then we would hope

$$s' \approx \mathbb{E}[S_{t+1} \mid s = S_t, a = A_t]$$

$$r \approx \mathbb{E}[R_{t+1} \mid s = S_t, a = A_t]$$

Three kinds of uses

- Expectation models can have disadvantages:
 - ▶ Imagine that an action randomly goes left or right
 - ▶ The expectation model might interpolate and put you in the middle
- An alternative can be **stochastic models** (like generative models), that returns next state according some probability.

$$\hat{R}_{t+1}, \hat{S}_{t+1} = \hat{p}(S_t, A_t, \omega)$$

where ω is a noise term

- Stochastic models can be chained, even if the model is non-linear -
But they do add a lot of noise

How to learn a policy with a model

Three kinds of uses

- (Wang et al. 19) did a classification and comparison of MBRL algorithms. (web [here](#))
- They classify MBRL methods into three different approaches:
 - 1 Model Predictive Control (MPC) inspired
 - 2 Learn from simulated (imagined/hallucinated) trials
 - 3 End-to-end approaches

Subsection 1

MPC-like methods

Model Predictive Control (MPC) [1]

- Helps to improve the policy:
 - ① Plan *whole trial* from step s_t using the world model
 - ② Execute only *first action*
 - ③ (get data to update the model and the policy)
 - ④ Return to 1 if not ended experience
- Why?

Model Predictive Control (MPC) [1]

- Helps to improve the policy:
 - ① Plan *whole trial* from step s_t using the world model
 - ② Execute only *first action*
 - ③ (get data to update the model and the policy)
 - ④ Return to 1 if not ended experience
- Why?
- Things can go wrong specially in stochastic environments or bad model
- Widely used in robotics and Control systems

PETS Example (1)

- Probabilistic Ensembles with Trajectory Sampling (PETS) is an example of MPC-like approach (Chua et. al 18)
- Authors Cite two kinds of uncertainty in models:
 - ▶ *Aleatoric uncertainty* which is the inherent stochasticity of the environment.
 - ▶ *Epistemic uncertainty* which reflects the model's confidence regarding different input state-actions.
- To solve aleatoric uncertainty they propose to use **probabilistic neural networks** as models that output parametrized gaussian distributions over the next state and reward
- They are trained using negative log likelihood of the true next state under the distribution predicted by our model

PETS Example (2)

- For solving epistemic uncertainty problem, they propose an **ensemble** of models

PETS Example (2)

- For solving epistemic uncertainty problem, they propose an **ensemble** of models
- Planning is done by generating different trajectories until the horizon allowed is reached
- They propose a trajectory sampling method that combines, by bootstrapping, the actual model of the ensemble that will be used in the prediction of one step.
- From all trajectories, the best one is selected

PETS Example (2)

- For solving epistemic uncertainty problem, they propose an **ensemble** of models
- Planning is done by generating different trajectories until the horizon allowed is reached
- They propose a trajectory sampling method that combines, by bootstrapping, the actual model of the ensemble that will be used in the prediction of one step.
- From all trajectories, the best one is selected and *only the first action* is executed

PETS Example (2)

- For solving epistemic uncertainty problem, they propose an **ensemble** of models
- Planning is done by generating different trajectories until the horizon allowed is reached
- They propose a trajectory sampling method that combines, by bootstrapping, the actual model of the ensemble that will be used in the prediction of one step.
- From all trajectories, the best one is selected and *only the first action* is executed
- Learning of the policy is done using cross entropy method (CEM)
- Achieve good in several domains in few trials (but more clock-time), specially in stochastic environments!
- Competitive [results](#)

Subsection 2

Learning from simulated trials

Learn from simulated trials [2]

- Idea: Instead of learning only from experiences in the environment, use the model to generate trials and learn from them!
 - ① Choose action from current state and observe results
 - ② Update the policy using this data
 - ③ Update the model with data collected
 - ④ Simulate trial with the model
 - ⑤ Update the policy with results from simulated data
 - ⑥ Return to 1!
- Some people use word *simulated*, others *imagined* or, when talking about visual states, *hallucinated*
- Some example methods:
 - ▶ Dyna
 - ▶ World Models paper
 - ▶ Model Based Policy Optimization (MBPO)

Dyna (Sutton 90)

- Based on Q-learning before DNNs. Sample based (not whole trajectories)

Tabular Dyna-Q

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

- $S \leftarrow$ current (nonterminal) state
- $A \leftarrow \varepsilon$ -greedy(S, Q)
- Take action A ; observe resultant reward, R , and state, S'
- $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
- $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- Loop repeat n times:
 - $S \leftarrow$ random previously observed state
 - $A \leftarrow$ random action previously taken in S
 - $R, S' \leftarrow Model(S, A)$
 - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

Dyna (Sutton 90)

- Can be done better
- Random exploration of experiences is not optimal
- Let's sample the experiences stored according to the imprecision on the prediction (Bellman error!)
- Ring a bell?
- Prioritized sweeping (Moore, Atkenson, 93)

Prioritized sweeping (Moore & Atkenson, 93)

Prioritized sweeping for a deterministic environment

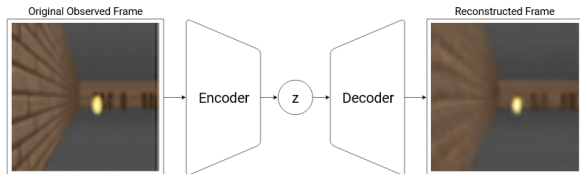
Initialize $Q(s, a)$, $Model(s, a)$, for all s, a , and $PQueue$ to empty

Loop forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow policy(S, Q)$
- (c) Take action A ; observe resultant reward, R , and state, S'
- (d) $Model(S, A) \leftarrow R, S'$
- (e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
- (f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
- (g) Loop repeat n times, while $PQueue$ is not empty:
 - $S, A \leftarrow first(PQueue)$
 - $R, S' \leftarrow Model(S, A)$
 - $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$Loop for all \bar{S}, \bar{A} predicted to lead to S :
 - $\bar{R} \leftarrow$ predicted reward for \bar{S}, \bar{A}, S
 - $P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.
 - if $P > \theta$ then insert \bar{S}, \bar{A} into $PQueue$ with priority P

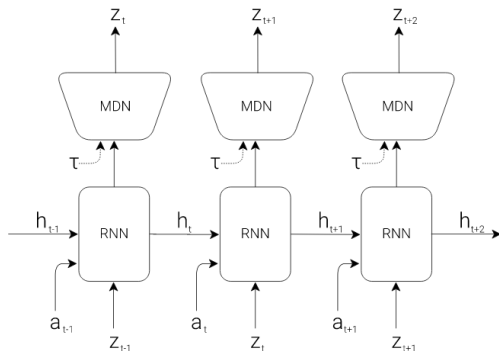
World Models 1 (Ha & Schmidhuber, 18)

- A more modern paper with nice interactive [web](#).
- Learn from pixels. To decrease dimensionality, creates a latent representation of observations.
- Latent representation is learnt using Variational Auto Encoder (VAE) before training the policy



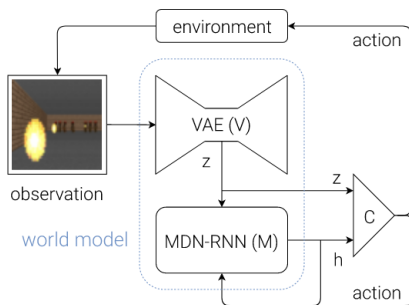
World Models 2 (Ha & Schmidhuber, 18)

- A model in the latent space is learnt using RNNs and Mixture Density Network (generative model of state controlled with noise and temperature)
- Model is trained collecting 10,000 rollouts from a random policy.



World Models 3 (Ha & Schmidhuber, 18)

- Finally CEM is used to train the policy
- At this point Model is only used as features to help the learning of the policy



World Models 4 (Ha & Schmidhuber, 18)

- So, if we have a model, can we use to learn a policy without going to the true environment?
- That is what they call *learning in a Dream*

World Models 4 (Ha & Schmidhuber, 18)

- So, if we have a model, can we use to learn a policy without going to the true environment?
- That is what they call *learning in a Dream*
- Problem about transferring the policy learnt to the true environment:
- Model was not perfect and policy take profit of that to increase *dreamed reward*
- Solved increasing noise in MDN
- Illustrative exploration in WM

Model Ensemble TRPO (Kurutahc et al. 18)

- (ME-TRPO) Learn an ensemble of Models and train a policy (with TRPO) collecting episodes from all of them
- When performance of policy does not increase for most of the ensembles, go back to env. for more data

Algorithm 2 Model Ensemble Trust Region Policy Optimization (ME-TRPO)

- 1: Initialize a policy π_θ and all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, \dots, \hat{f}_{\phi_K}$.
 - 2: Initialize an empty dataset \mathcal{D} .
 - 3: **repeat**
 - 4: Collect samples from the real system f using π_θ and add them to \mathcal{D} .
 - 5: Train all models using \mathcal{D} .
 - 6: **repeat** ▷ Optimize π_θ using all models.
 - 7: Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^K$ using π_θ .
 - 8: Update the policy using TRPO on the fictitious samples.
 - 9: Estimate the performances $\hat{\eta}(\theta; \phi_i)$ for $i = 1, \dots, K$.
 - 10: **until** the performances stop improving.
 - 11: **until** the policy performs well in real environment f .
-

SLBO (Luo et al. 19) variation that separates learning Model from policy and train models on differences (derivative)

Model Based Policy Optimization (Janer et al. 19)

- After analysis of error in MBRL they conclude that collecting whole trajectories and learning from them lead to accumulate error
- MBPO proposes a Dyna-like procedure for sampling and learn with SAC
- You can take larger samples because you learn not from episodes but from transitions

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

- 1: Initialize policy π_ϕ , predictive model p_θ , environment dataset \mathcal{D}_{env} , model dataset $\mathcal{D}_{\text{model}}$
 - 2: **for** N epochs **do**
 - 3: Train model p_θ on \mathcal{D}_{env} via maximum likelihood
 - 4: **for** E steps **do**
 - 5: Take action in environment according to π_ϕ ; add to \mathcal{D}_{env}
 - 6: **for** M model rollouts **do**
 - 7: Sample s_t uniformly from \mathcal{D}_{env}
 - 8: Perform k -step model rollout starting from s_t using policy π_ϕ ; add to $\mathcal{D}_{\text{model}}$
 - 9: **for** G gradient updates **do**
 - 10: Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$
-

Subsection 3

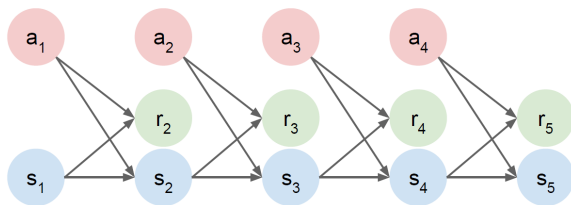
End-to-end approaches

End-to-end approaches [3]

- Accumulating errors are a source of problems in MBRL ([example](#))
- Previous approaches only use first action (MPC) or use ensembles or small rollouts (Dyna like) to diminish the problem

End-to-end approaches [3]

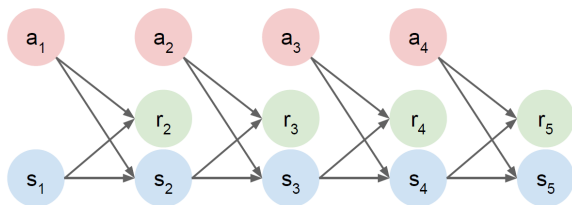
- Accumulating errors are a source of problems in MBRL ([example](#))
- Previous approaches only use first action (MPC) or use ensembles or small rollouts (Dyna like) to diminish the problem
- But wait, lets look in detail a imagined episode (assuming sequence is generated using the model)



- Notice that $s_5 = f(f(f(f(s_1, a_1), a_2), a_3), a_4)$

End-to-end approaches

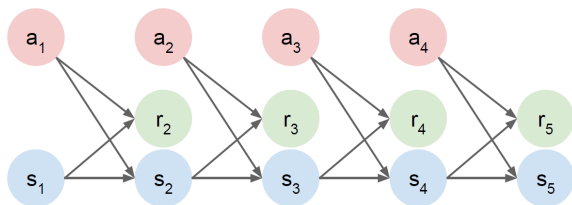
$$s_5 = f(f(f(f(s_1, a_1), a_2), a_3), a_4)$$



- Also notice that f is a DNN (usually)

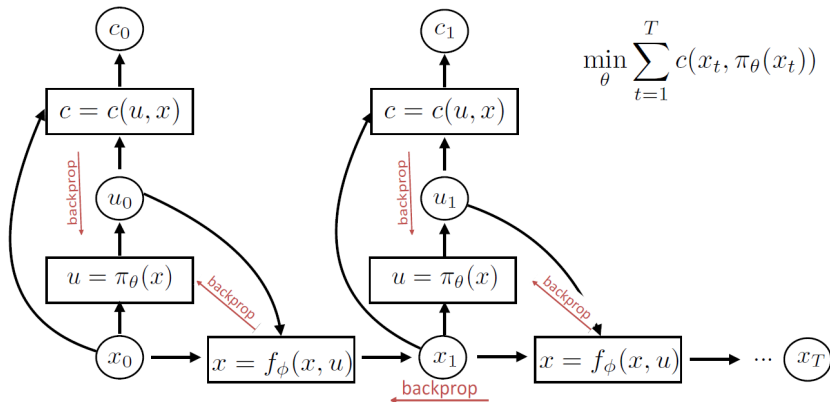
End-to-end approaches

$$s_5 = f(f(f(f(s_1, a_1), a_2), a_3), a_4)$$



- Also notice that f is a DNN (usually)
- So, you can apply back-propagation from any state backwards to the initial state in order to learn the model!
- This is called *Back-propagation through time* (BTT)

End-to-end approaches



1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$

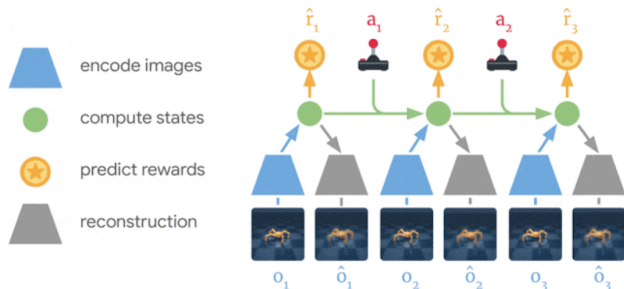
2. learn dynamics model $f_\phi(x, u)$ to minimize $\sum_i \|f_\phi(x_i, u_i) - x'_i\|^2$

3. backpropagate through $f_\phi(x, u)$ into the policy to optimize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

4. run $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, appending the visited tuples $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ to \mathcal{D}

End-to-end approaches: DREAMER

- Dreamer algorithm (evolution of previous Planet)



- Learns on latent space
- Uses BTT to learn the models (they call multi-step prediction)

- Other things not in image:
 - ▶ It uses RNN to deal with partial observability
 - ▶ Used TD(λ) to learn the policy
 - ▶ Use stochastic models to solve aleatoric uncertainty

- Other things not in image:
 - ▶ It uses RNN to deal with partial observability
 - ▶ Used $TD(\lambda)$ to learn the policy
 - ▶ Use stochastic models to solve aleatoric uncertainty
- Lately, they have proposed Dreamer v2 ([Haffner et al. 20](#)) that claim it's better than Rainbow in Atari Games
- In general End-to-end approaches have problems with local minima and gradient vanishing (similar problems than RNNs)

Subsection 4

Conclusions

Conclusions

- Different ways to deal with uncertainty in models
- Learning the policy and the model is done at the same time
- (Wang et al. 19) did a classification and comparison of MBRL algorithms (web [here](#))

Conclusions

- Different ways to deal with uncertainty in models
- Learning the policy and the model is done at the same time
- (Wang et al. 19) did a classification and comparison of MBRL algorithms (web [here](#))
- They do reduce number of interaction with the env.
- ... but usually don't achieve model-free methods performance

Do models help in another way?

- Other uses:
 - ▶ Classical planning and learning a policy (Monte Carlo Tree search and uses in AlphaGo, Muzero, etc.)
 - ▶ Help to explore better the domain [Plan2Explore](#)
 - ▶ As auxiliary loss
 - ▶ Transfer between RL tasks
 - ▶ ...