Word Models
https://arxiv.org/pdf/1803.10122.pdf

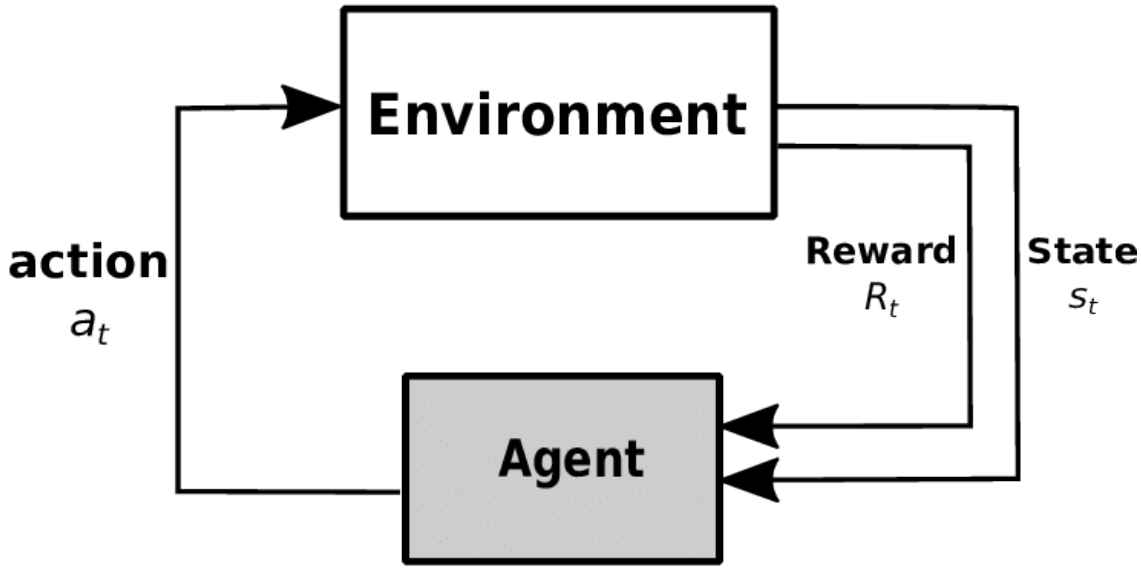
by Vecoven Nicolas

April 2020

Content

- Reinforcement learning
- Introduction to model-based reinforcement learning
- Tabular model-based
- Model-based in continuous state-action space
- Model-based in a stochastic environment
- Model-based in a partially observable environment
- World Models with images
- Using the model to learn a policy
- Cheating the model
- What about more complex environments ?

Reinforcement learning : A reminder



Goal : to maximize the sum of (discounted) rewards obtained by the agent in an environment modeled by a transition and reward functions.

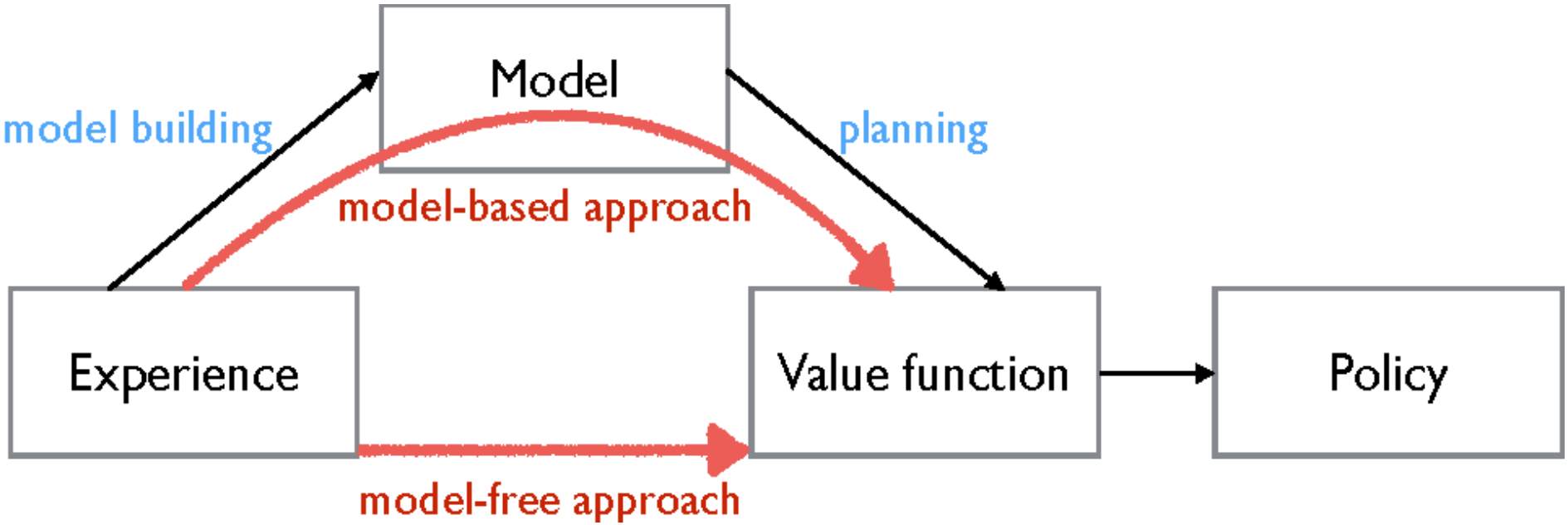
$$s_{t+1} \sim p(s_{t+1}|s_t, a_t) \tag{1}$$

$$r_t = R(s_{t+1}, s_t, a_t) \tag{2}$$

Reinforcement learning paradigms

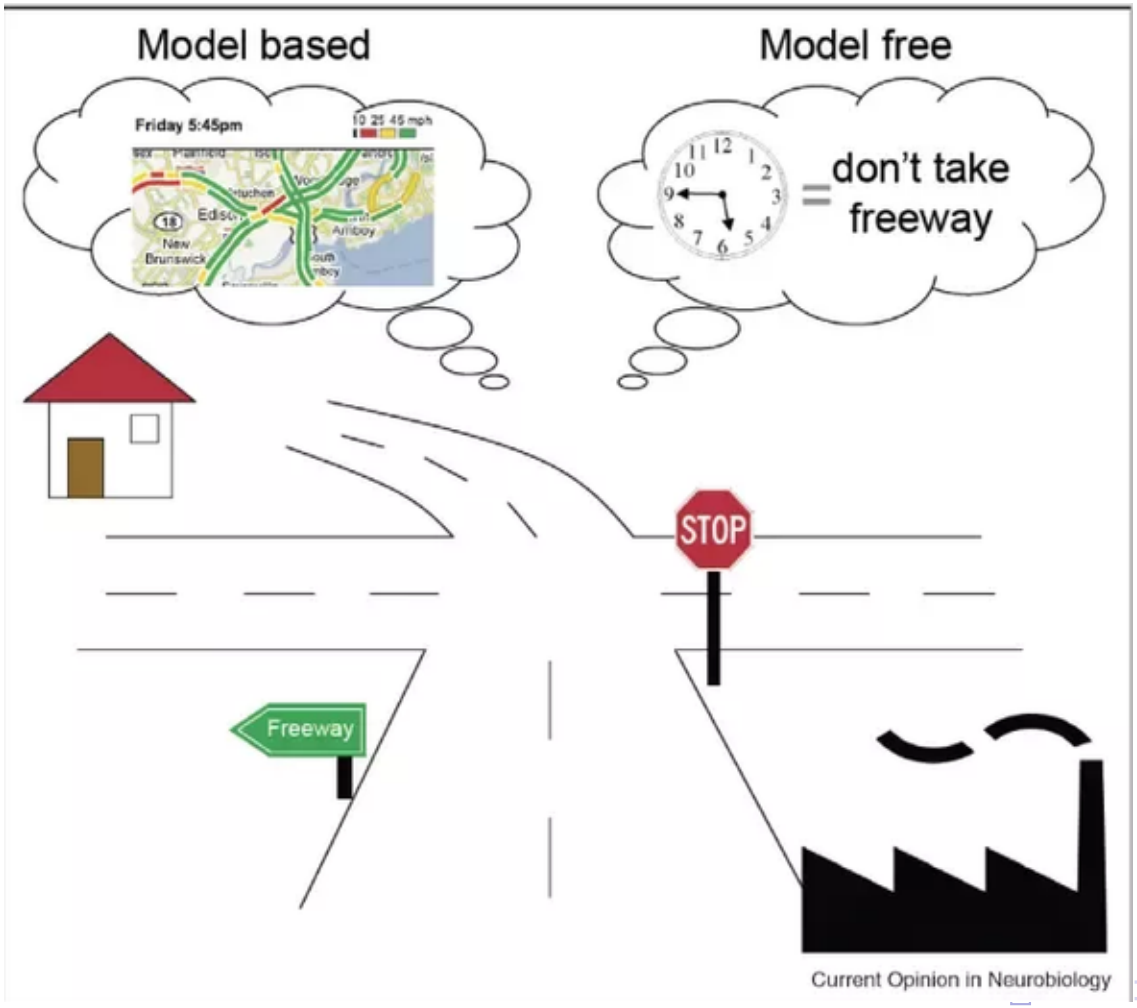
- Model-free : Directly learns a policy, without any effort trying to learn the environment's characteristics. Examples of this are Q-learning, Policy gradients, Reinforce,...
  $\rightarrow$ Usually very sample inefficient (lots and lots of interactions with the environment needed).
- Model-based : First, **try to model the environment**, then uses that model to learn or compute a policy.
  $\rightarrow$ Usually requires much less interactions with the environment (use of supervised learning).
  $\rightarrow$ Very useful when it is expensive to interact with the environment !

Reinforcement learning paradigms



World models

Sometimes the environments are complex, or unknown, and thus can't be modeled easily. This leads to the need of **learning** a model of the environment, that is, a **world model**.



Minimal procedure

There are plenty of ways to learn a world-model, one of which we'll see in details here. However, the minimal procedure is as follows:

- Use a random policy to interact with the environment for multiple episodes.
- Use the gathered samples (s_{t+1}, s_t, a_t, r_t) to approximate the transition and reward functions.

Most simple environment

Let's first take a look at the most basic environment, one for which:

- The transition function is deterministic. Thus $p(s_{t+1}|s_t, a_t)$ degenerated in diracs which positions are a function of s_t and a_t.
- The state-space is discrete and the agent can only take a discrete number of actions.

In such a setting, if the state space is not too large, it is very easy to learn a world-model.

Tabular model-based

Using the procedure previously mentioned : start with a random world-model.

	$s_t = 0$	$s_t = 1$	$s_t = 2$
$a_t = 0$	?	?	?
$a_t = 1$	?	?	?
$a_t = 2$	?	?	?

Tabular model-based

Play a random policy in the environment. In this case, it could be $\pi(a_t|s_t) = \mathbb{U}\{0, 1, 2\}$. Observe interaction with the environment and fill the table.
For example:
$(s_0 = 0, a_0 = 0, s_1 = 1, r_0 = 0) \rightarrow (s_1 = 1, a_1 = 1, s_2 = 1, r_1 = 0) \rightarrow (s_2 = 1, a_2 = 2, s_3 = 2, r_2 = 20) \rightarrow (s_3 = 2, a_3 = 2, s_4 = 2, r_3 = 0)$

	$s_t = 0$	$s_t = 1$	$s_t = 2$
$a_t = 0$	$s_{t+1} = 1$ $R_t = 10$	?	?
$a_t = 1$	?	$s_{t+1} = 1$ $R_t = 0$	?
$a_t = 2$	?	$s_{t+1} = 2$ $R_t = 20$	$s_{t+1} = 2$ $R_t = 10$

Model-based in a continuous state-action space

What if s and a are continuous, or very large ? It is impossible to fill the table (as in Q-learning).
$\rightarrow$**We can use a neural network to approximate the transition and reward functions.**

Neural network as a simple world-model

Let θ denote the parameters of the neural network $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with n and m the state and action spaces dimensions respectively, we have:

$$\hat{s}_{t+1}, \hat{r}_t = f(s_t, a_t; \theta) \quad .$$

Let $\mathcal{T}$ denote the set of (s_{t+1}, s_t, a_t, r_t) tuples observed while interacting randomly with the environment. The world-model can then simply be trained using a standard MSE loss:

$$\mathcal{L}_\theta = \sum_{(s_{t+1}, s_t, a_t, r_t) \in \mathcal{T}} ([s_{t+1}, r_t] - f(s_t, a_t; \theta))^2 \quad .$$

Stochastic environments

Now, what if the environment is stochastic ? Think of doom where a teleporter can take you to two different places at random.
For the world model to be accurate, it needs to be able to model stochasticity.
$\rightarrow$ **Make the neural network model a distribution over possible next states and rewards**

NN to model the world-model as a distribution

In this case we have:

$$\hat{s}_{t+1}, \hat{r}_t \sim \hat{p}(s_{t+1}, r_t | s_t, a_t; \theta) \quad .$$

Training is done by maximizing the log-likelihood of the samples observed while interacting with the environment:

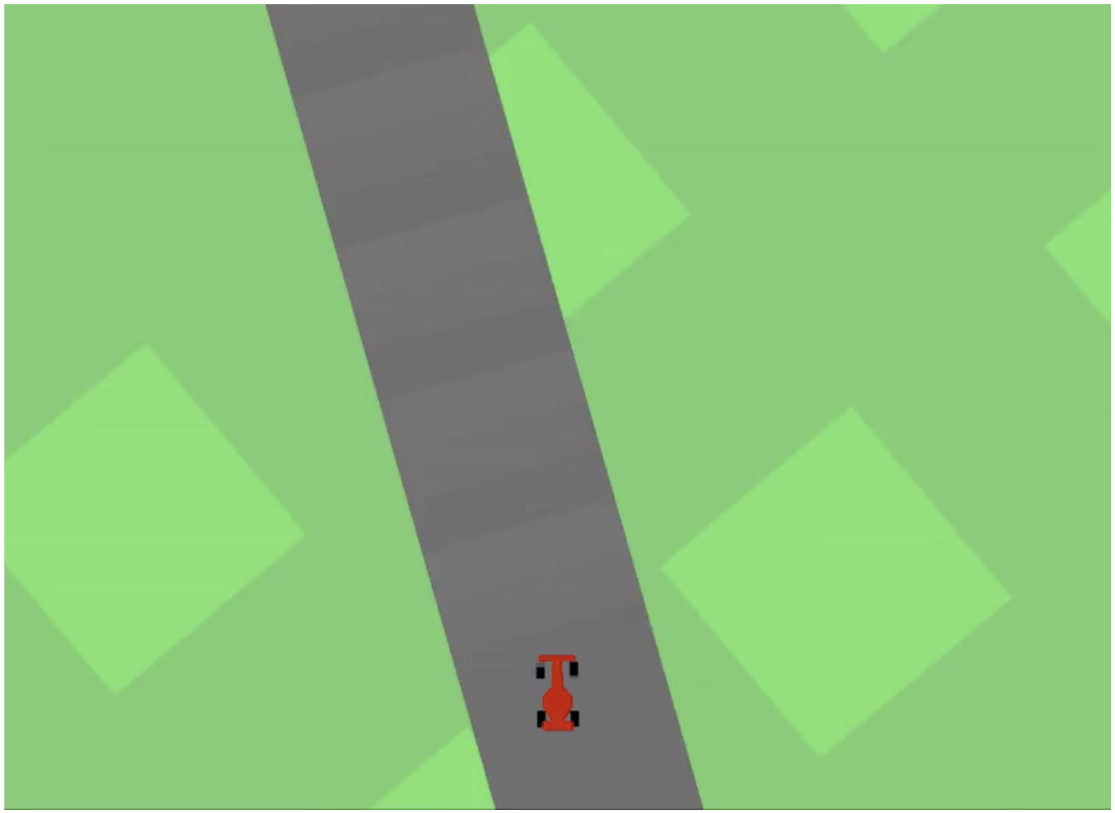
$$\mathcal{L}_\theta = \mathbb{E}_{(s_{t+1}, s_t, a_t, r_t) \in \mathcal{T}} - \log(\hat{p}(s_{t+1}, r_t | s_t, a_t; \theta)) \quad .$$

What distribution to use ?

Rethink of doom's teleporter. Is a Gaussian suitable, if the two places where you can be teleported are far one from another ?
Use a multivariate Gaussian mixture $\rightarrow$ Mixture density network (MDN)
(Or a normalizing flow, as presented by Antoine Wehenkel)

What if the agent can't observe the state of the world ?

Now, think of a racing game. The agent can only observe a frame of the game at a time. $\rightarrow$ No information on the speed of the vehicle !



The agent receives a noisy version o_t of the environment's state through

$$o_t \sim O(o_t|s_t) \quad .$$

The tuples observed are now (o_{t+1}, o_t, a_t, r_t) and the model learned is:

$$\hat{o}_{t+1}, \hat{r}_t \sim p(o_{t+1}, r_t|o_t, a_t; \theta) \quad .$$

This model might be very imprecise if o_t is a very noisy version of s_t !

Using MDN-RNN as a world-model

One way to filter more information on the current true-state of the environment s_t is to use the observations history:

$$\hat{o}_{t+1}, \hat{r}_t \sim p(o_{t+1}, r_t | o_t, o_{t-1}, \ldots, o_0, a_t; \theta) \quad .$$

To this end, one can use a recurrent neural network, that is a network that maintains a hidden state h_t through time.

$$h_{t+1} = f(o_t, a_t, h_t; \theta) \tag{3}$$

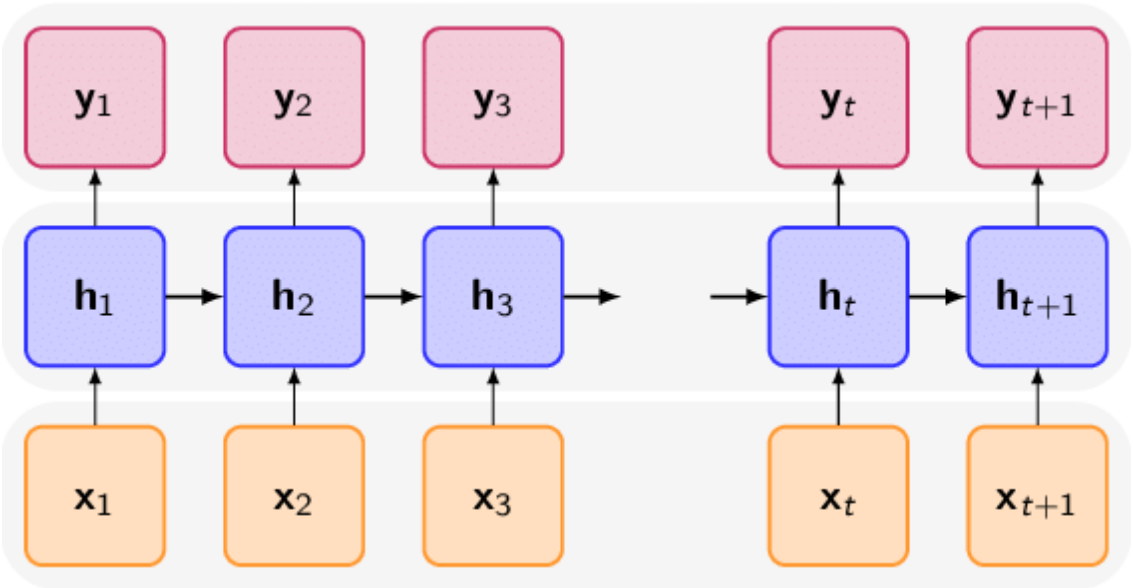
$$\hat{o}_{t+1}, \hat{r}_t \sim p(o_{t+1}, r_t | o_t, a_t, h_t; \theta) \tag{4}$$

Brief reminder on RNN layers

Most simple way to build a recurrent layer:

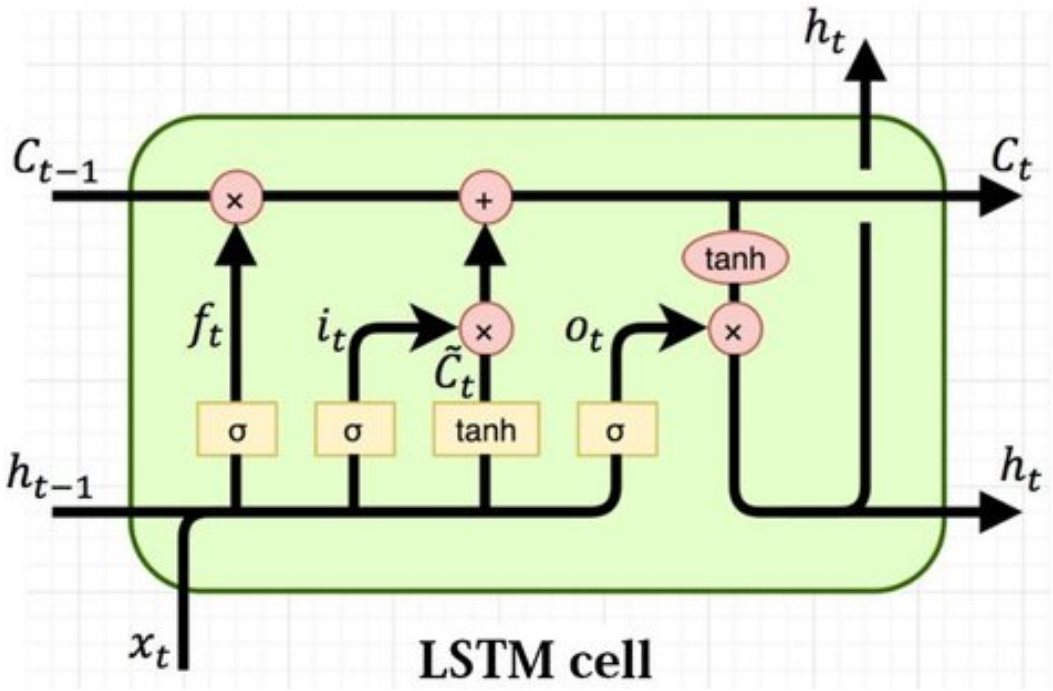
$$h_t = \tanh(W_x x_t + W_h h_t + b) \quad .$$

But dependencies can span over many time-steps ! Think of the gradient of h_{200} with respect to x_0 ?
$\rightarrow$ Vanishing / exploding gradients



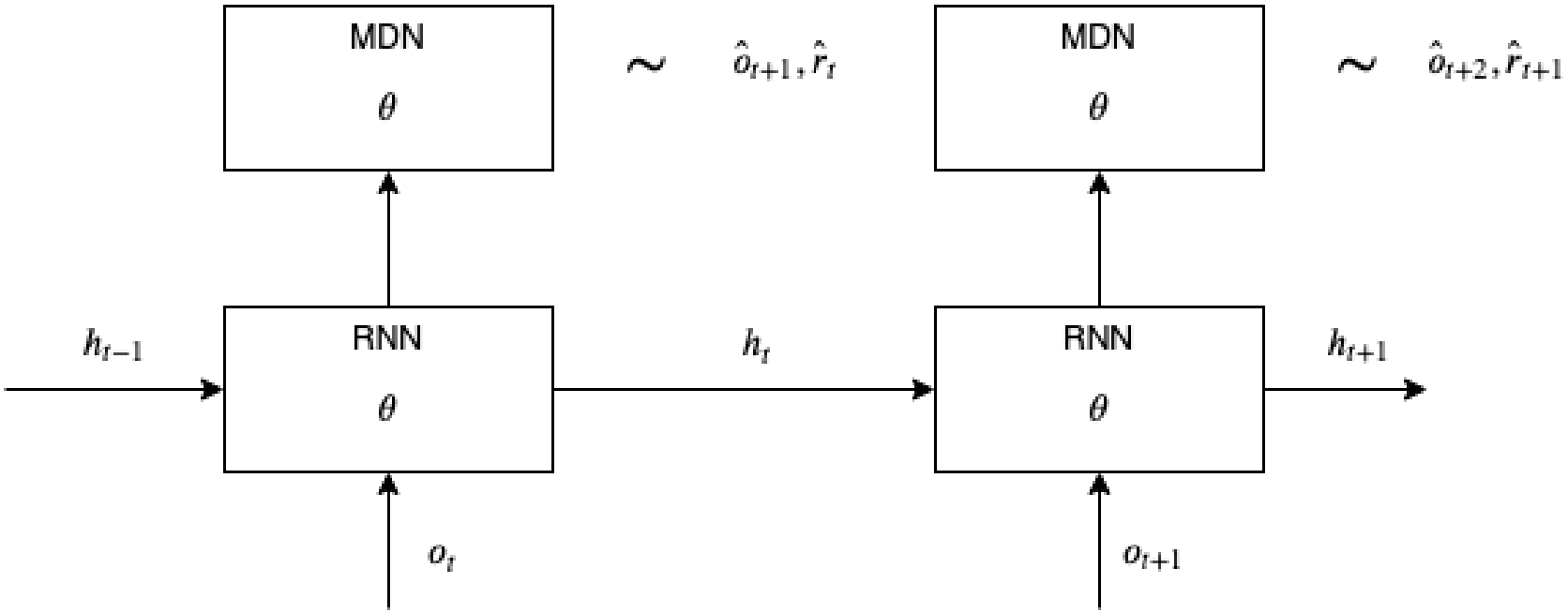
What can we do ?

LSTM



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

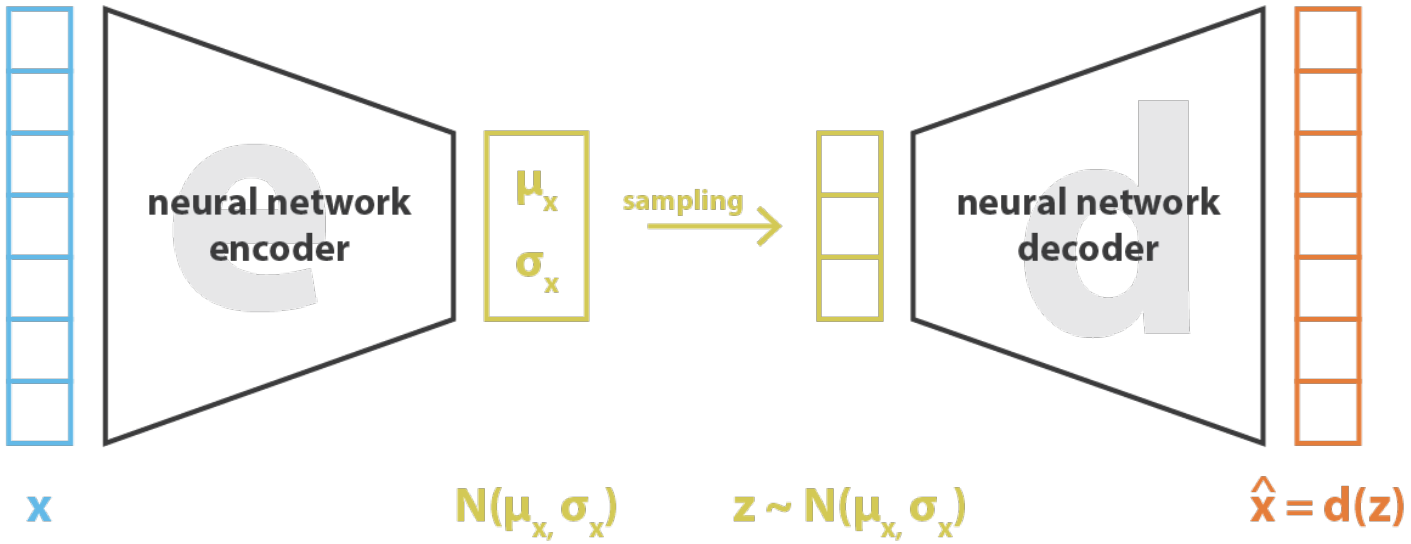
Until now


MDN
θ
$\sim$
$\hat{o}_{t+1}, \hat{r}_t$
MDN
θ
$\sim$
$\hat{o}_{t+2}, \hat{r}_{t+1}$
h_{t-1}
RNN
θ
h_t
RNN
θ
h_{t+1}
o_t
o_{t+1}

Working with images

Working in more complex environments, it might be interesting to encode the observations into more compact and meaningful features (for examples if o_t is an image), as done in the world-models paper.
We can use a **variational auto-encoder** to compress the observation space into a latent space.

VAE



$$\text{loss} = \| x - \hat{x} \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,] = \| x - d(z) \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,]$$

We are then mostly interested by the encoder part ! $z_t \sim e(z_t | o_t; \psi)$

A few examples of the latent space of a VAE

Goto github

Including a VAE in world models

The procedure for building the world-model is now three fold:

- Use a random policy to interact with the environment to gather tuples (o_{t+1}, o_t, a_t, r_t) for multiple episodes.
- Use those samples to train a VAE to encode o_t into z_t.
- Use those samples, and the encoder, to train a MDN-RNN that models

$$z_{t+1}, r_t \sim p(z_{t+1}, r_t | z_t, a_t; \theta) \quad .$$

Learning a policy from the world models

Once an accurate world-model has been built, there are many ways to learn / compute a policy. One could for example use search algorithms such a MCTS on the world-model to compute an action.
Here, we'll look at a way to use the features learnt by the world model to build a simple, yet very efficient controller.

Controller

Thanks to the supervised training of the world-model, it is expected to have learned relevant and compact features to model the environment. Namely z_t has been built to encode the current observation into a compact meaningful latent space (as shown before) and h_t is built to represent a relevant history of such states.
From these features, it is thus very easy to build a controller, with parameters α such that

$$a_t = \pi(z_t, h_t; \alpha) \quad .$$

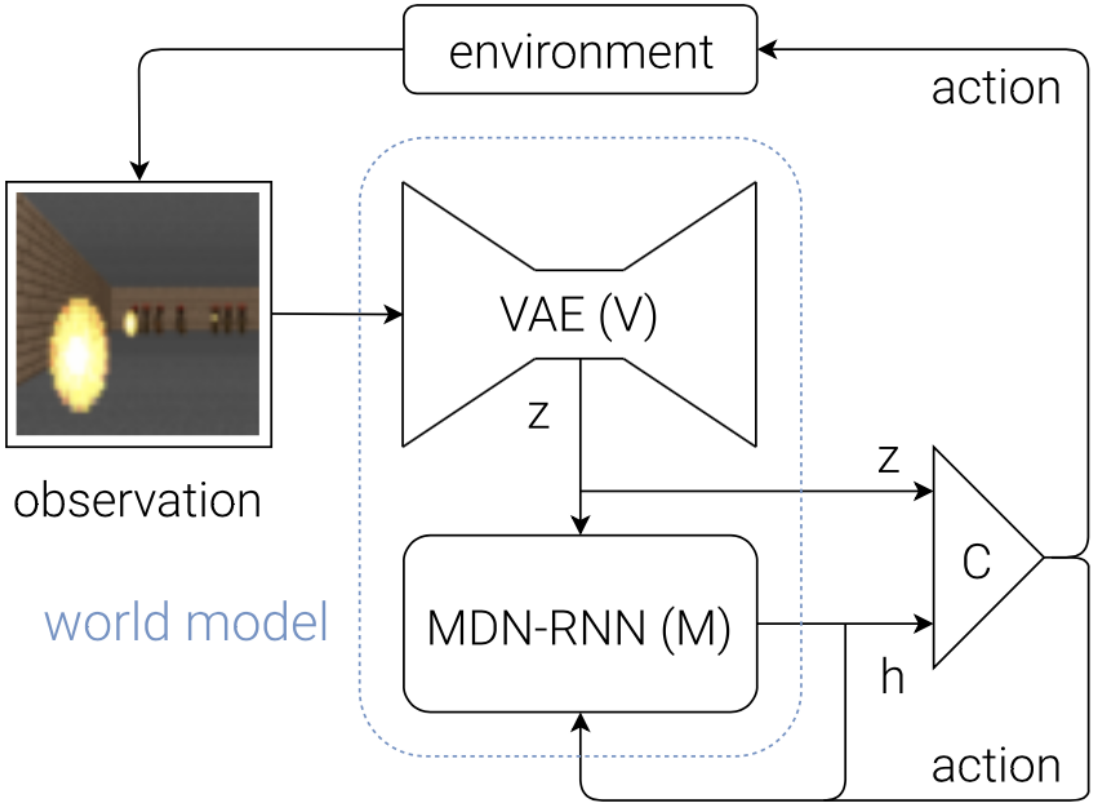
The main advantage of world models is that π can be made very small, usually as small as a single linear layer such that

$$a_t = W_h h_t + W_z z_t + b \quad .$$

Training the controller

Thanks to the power of the features learned by the world-models, π can then be trained very efficiently, with standard RL techniques such as Q-learning, PPO,... by playing the environment and using the features encoded by the world model.
In fact, in the paper π is so simple that it can even be trained through Covariance-matrix adaptation evolution strategy (CMAES), not even requiring stochastic gradient descent.

Full model



Flow diagram of our Agent model. The raw observation is first processed by V at each time step t to produce z_t. The input into C is this latent vector z_t concatenated with M's hidden state h_t at each time step. C will then output an action vector a_t for motor control. M will then take the current z_t and action a_t as an input to update its own hidden state to produce h_{t+1} to be used at time $t + 1$.

But, we have a world-model representing the environment. Can't we do better than interact with the environment to train the controller, as this might be expensive ?

Learning in the dream !

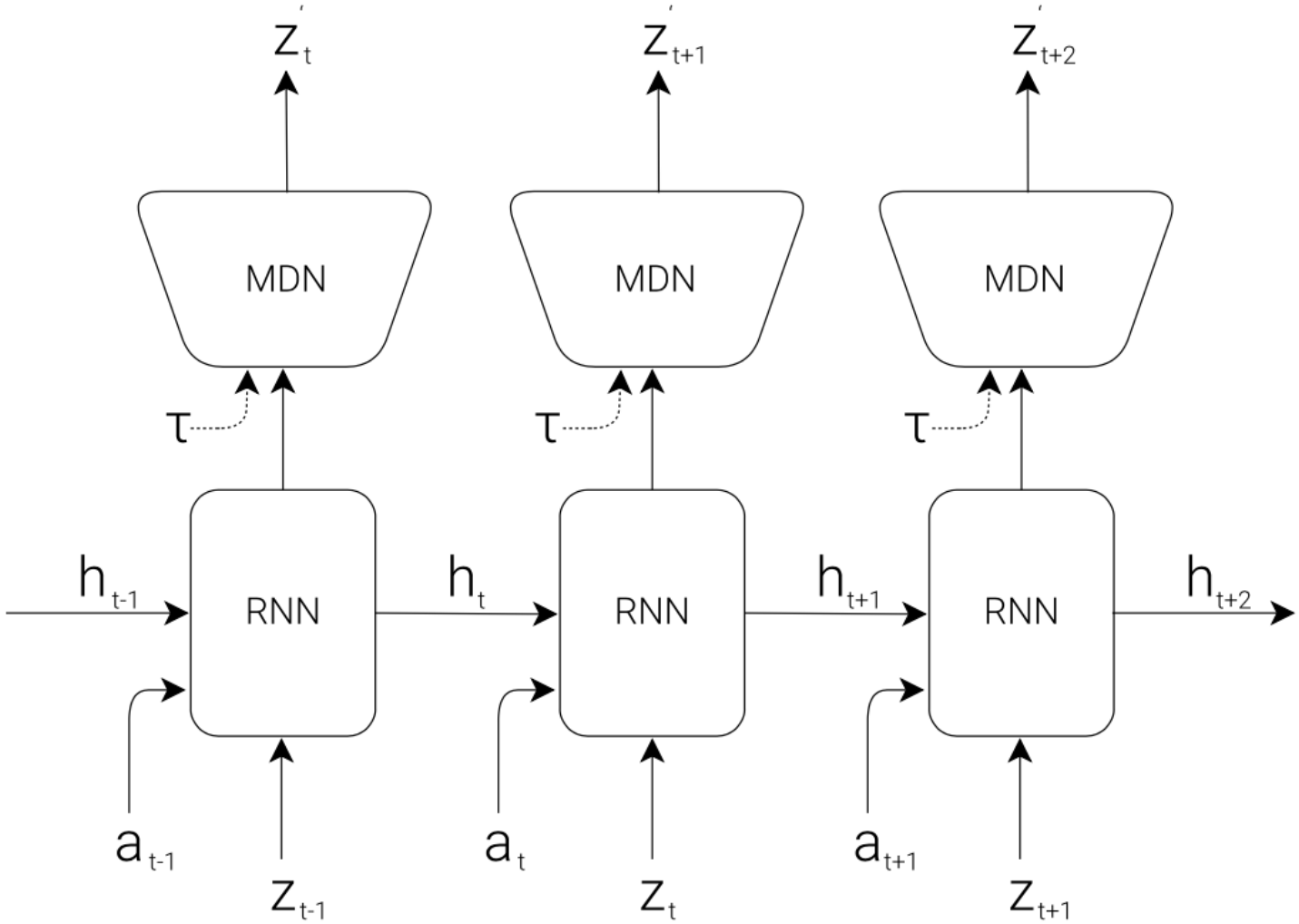
As we have a fully functional world-model, we can in fact use it to **train our controller directly by simulating the environment.**

Let's take a look at DOOM ! Game engine fully modelled by the world model !

Cheating !

The world-model is never perfect. So the agent can learn to exploit some of the model's imperfections !

Introducing a temperature parameter



RNN with a Mixture Density Network output layer. The MDN outputs the parameters of a mixture of Gaussian distribution used to sample a prediction of the next latent vector z.

Back to DOOM

Let's look at the effect of the temperature parameter

What if a random policy is not enough to build a good world-model ?

In complex environments, the samples obtained by acting randomly will not suffice to learn a good world-model. $\rightarrow$ Use an iterative procedure.

As a conclusion

1. Initialize M, C with random model parameters.

2. Rollout to actual environment N times. Agent may learn during rollouts. Save all actions a_t and observations x_t during rollouts to storage device.

3. Train M to model $P(x_{t+1}, r_{t+1}, a_{t+1}, d_{t+1} \mid x_t, a_t, h_t)$ and train C to optimize expected rewards inside of M.

4. Go back to (2) if task has not been completed.

We have shown that one iteration of this training loop was enough to solve simple tasks. For more difficult tasks, we need our controller in Step 2 to actively explore parts of the environment that is beneficial to improve its world model. An exciting research direction is to look at ways to incorporate artificial curiosity and intrinsic motivation [59, 60, 61, 62, 63]

Sources for images

- https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
- https://worldmodels.github.io
- https://www.researchgate.net/figure/Reinforcement-Learning-Agent-and-Environment_fig2_323867253
- https://www.researchgate.net/figure/Structure-of-the-LSTM-cell-and-equations-that-describe-th fig5_329362532
- https://www.semanticscholar.org/paper/Strengths%2C-Weaknesses%2C-and-Combinations-of-and-Asadi/5f936eae11c88150c1874fc0615f95aa435a9567
- https://towardsdatascience.com/google-open-sources-dreamer-a-reinforcement-learning-agen
- https://www.researchgate.net/figure/An-RNN-unrolled-in-time_fig4_325385139