Google DeepMind

Model-based Reinforcement Learning

Matteo Hessel

Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L

www.decoding-intelligence.com

Google DeepMind

01 - Model-free vs Model-based

Matteo Hessel

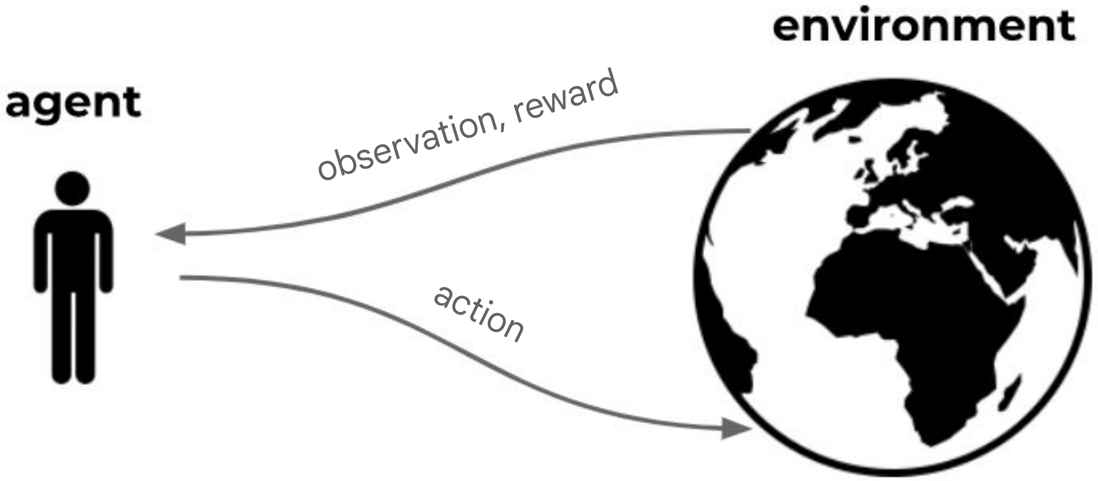
Research Engineer @ Google DeepMind

Honorary Lecturer @ UCL

Organizer @ M2L

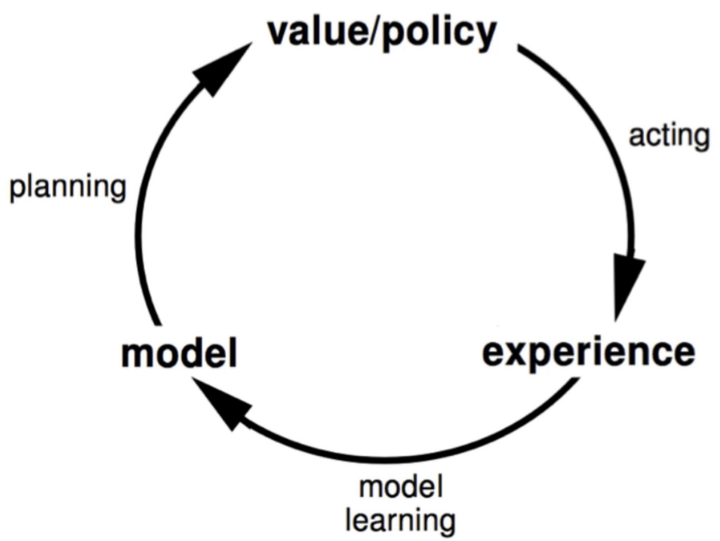
www.decoding-intelligence.com

Model-free RL



Google DeepMind

Model-based RL



Google DeepMind

02 - What is a model?

Matteo Hessel

Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L

www.decoding-intelligence.com

What is a model?

A model approximates the dynamics of the environment by modelling

- State transitions
- Rewards

$$R_{t+1}, S_{t+1} \sim \hat{p}_\eta(r, s' \mid S_t, A_t)$$

Google DeepMind

Model learning

The nice thing about model learning is that this reduces to a simple **supervised learning** problem

The agent can collect a dataset of transitions

$$\begin{gathered} S_1, A_1 \to R_2, S_2 \\ \vdots \\ S_{T-1}, A_{T-1} \to R_T, S_T \end{gathered}$$

And then we only need to pick a parametric function, a loss function

Then any standard SL algorithms can be used to find an approximation of the dynamics of the environment

Expectation models - I

If we use mean squared error as loss function we will learn an **expectation model**

where if $f(s, a) = r, s'$ then $s' \approx \mathbb{E}[S_{t+1} \mid s = S_t, a = A_t]$

Expectation models can have disadvantages:

- Imagine that a (high-level) action randomly goes left or right past a well
- The expectation model might interpolate and put you **in the wall**

As a result the

- **Values** of expected states are not necessarily well defined
- **Iterating** a model multiple times to predict the environment state N steps ahead can also be "weird"

Expectation models - II

Interestingly as long as model & values are a linear in the the state representation this is actually fine

Consider a linear model $\mathbb{E}[\phi_{t+1}] = P\phi_t$ and a linear value function $v_\theta(S_t) = \theta^\top \phi_t$

$$\begin{aligned}
\mathbb{E}[v_\theta(S_{t+n}) \mid S_t = s] &= \mathbb{E}[\theta^\top \phi_{t+n} \mid S_t = s] \\
&= \mathbb{E}[\theta^\top P\phi_{t+n-1} \mid S_t = s] \\
&= \ldots \\
&= \mathbb{E}[\theta^\top P^n \phi_t \mid S_t = s] \\
&= \theta^\top P^n \phi(s) \\
&= v_\theta(P^n \phi(s)) \\
&= v_\theta(\mathbb{E}[\phi_{t+n} \mid S_t = s]) \, .
\end{aligned}$$

Google DeepMind

Stochastic models

Alternative: stochastic models (i.e. generative models):

- These return samples of the next state and rewards rather then expected values

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

  where ω is a noise term

- Stochastic models can be chained (but they do introduce noise)

Full distribution models

Finally you can model explicitly the **entire distribution**

- In this case the model that takes as input tuples (S, A, S′)
  And returns you the probability of observing that transition for any S, A, S′

Table look-up model

The simplest form of model is a table look-up

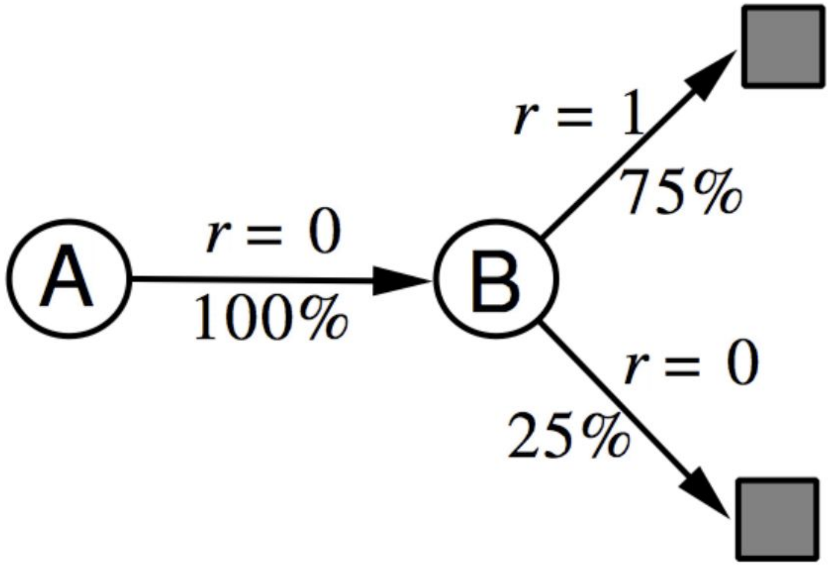
$$\hat{p}_t(s' \mid s, a) = \frac{1}{N(s, a)} \sum_{k=0}^{t-1} I(S_k = s, A_k = a, S_{k+1} = s')$$

$$\mathbb{E}_{\hat{p}_t}[R_{t+1} \mid S_t = s, A_t = a] = \frac{1}{N(s, a)} \sum_{k=0}^{t-1} I(S_k = s, A_k = a) R_{k+1}$$

Google DeepMind

Model learning example

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Deep model-based RL

In most problems of interest the model itself will have to be a parametric function

- Typically a neural network
- So that we can generalise what we learn about the environment's dynamics across states

Google DeepMind

03 - *Global* Planning

Matteo Hessel
Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L
www.decoding-intelligence.com

Sample based planning

A simple but powerful approach to planning

- Use the model only to generate samples
- Sample experience from model

Apply model-free RL to samples, e.g. Q-learning

Advantages of model-based RL

Sometimes the model of the environment is very simple to learn

- E.g. in a board game taking an action in a state typically results in simple predictable consequences

But figuring the right way of acting in that environment might still be exceedingly hard

In this case, learning a model and then using it to plan

- allows you to learn from a huge number of simulations
- without interacting with a potentially slow or expensive real environment

Disadvantages of model-based RL

There are also disadvantages

- Sometimes the model is still hard to learn
- The model learning is not aware of what matters to the planning algorithms
  - Small errors according to the model loss function might lead to big errors on the values and policy
  - Planning might result in very sub-optimal values and policy

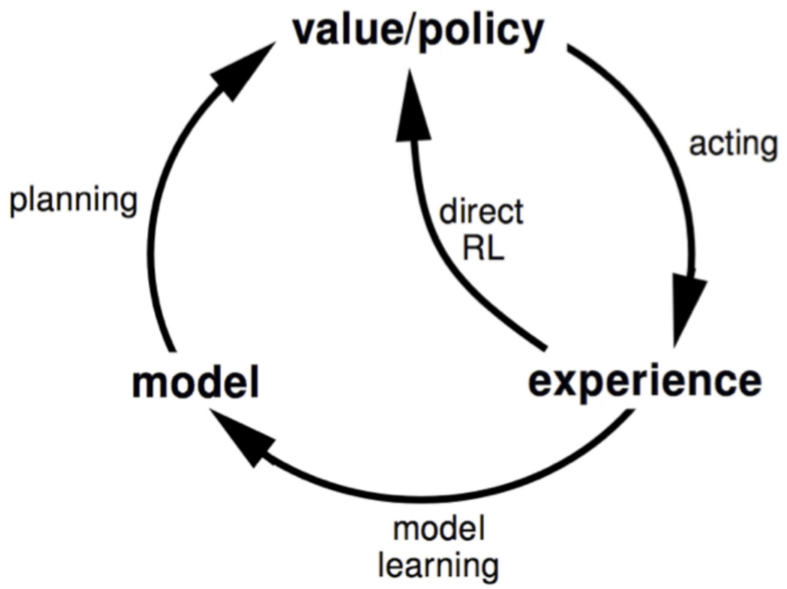
Three possible approaches:

- If model is inaccurate just use model-free RL
- Reason explicitly about inaccuracies / uncertainties in the model when planning (e.g. bayesian approach)
- Combine model free and model based RL in a safe way

Integrated learning and planning

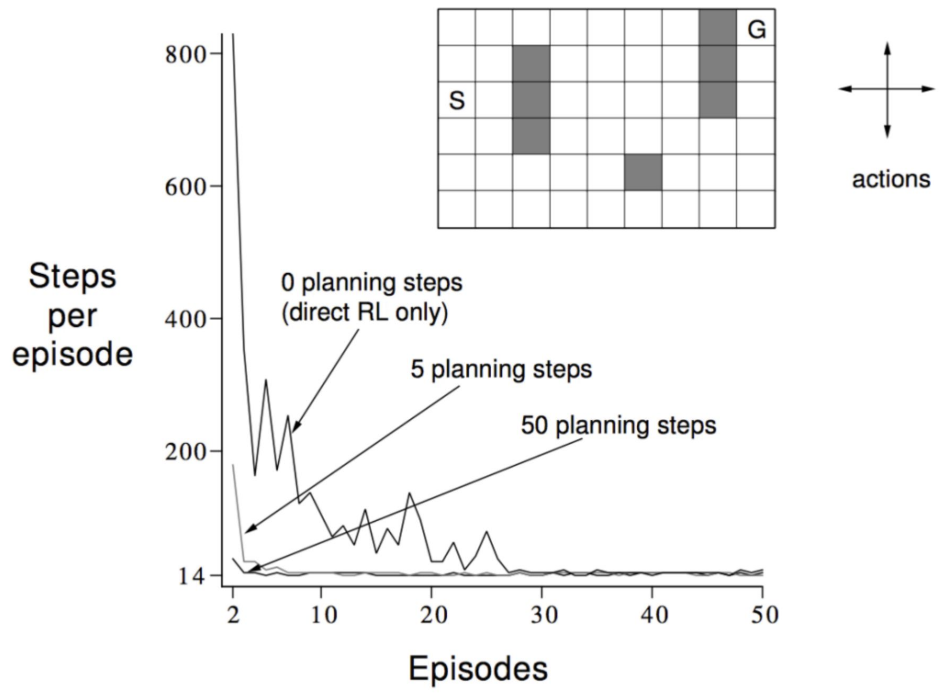
- Model-free RL
  - No model
  - Learn a value function (and/or a policy) from real experience
- Model-based RL
  - Learn a model from real experience
  - Plan a value function (and/or policy) from fictitious data
- Dyna
  - Learn a model from real experience
  - Learn AND plan a value function (and/or policy) from both real and simulated experience
  - Treat real and simulated experience equivalently.
    Conceptually, the updates from learning or planning are not distinguished.

Dyna - I

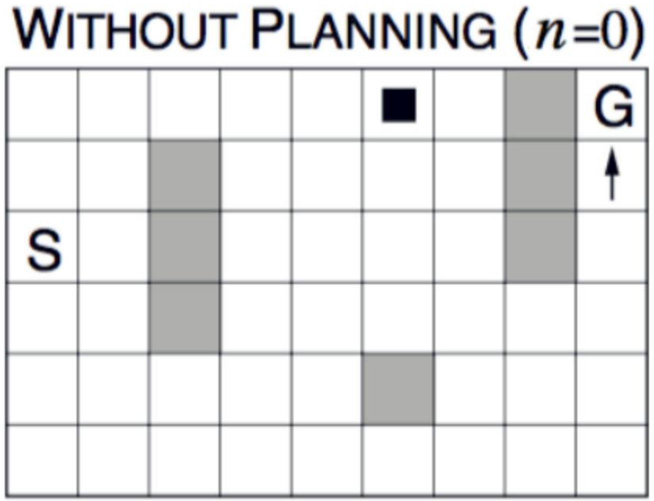

value/policy
acting
planning
direct RL
model
experience
model learning

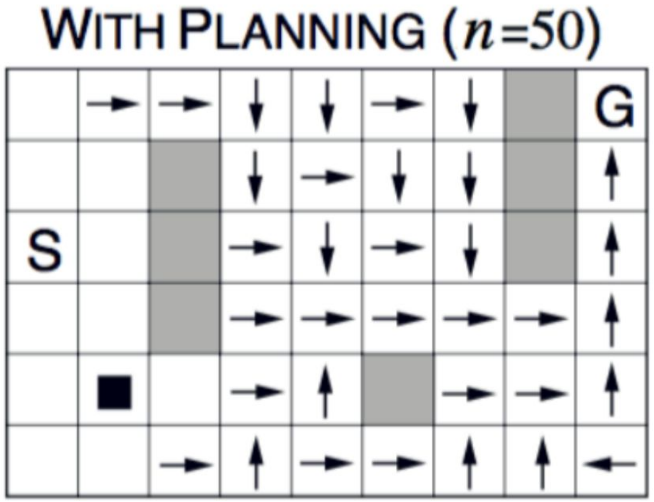
```
$S = $ env.initial_state()
$Q(s, a) = $ random() $\quad \forall s, a$
While True:
    $A = \epsilon - \text{greedy}(Q, S)$
    $R, S' = $ env.step($A$)
    $Q(S, A) \to R + \gamma \max_{a'} Q(S', a')$
    Repeat N times:
        $S = \text{random previous state}$
        $A = \epsilon - \text{greedy}(Q, S)$
        $R, S' = \text{model.step}(S, A)$
        $Q(S, A) \to R + \gamma \max_{a'} Q(S', a')$
```

Dyna-Q on a simple maze



Dyna-Q on a simple maze



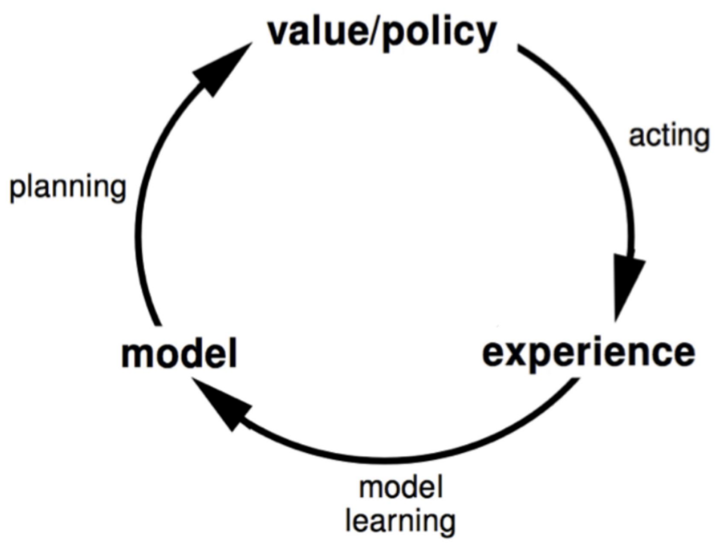


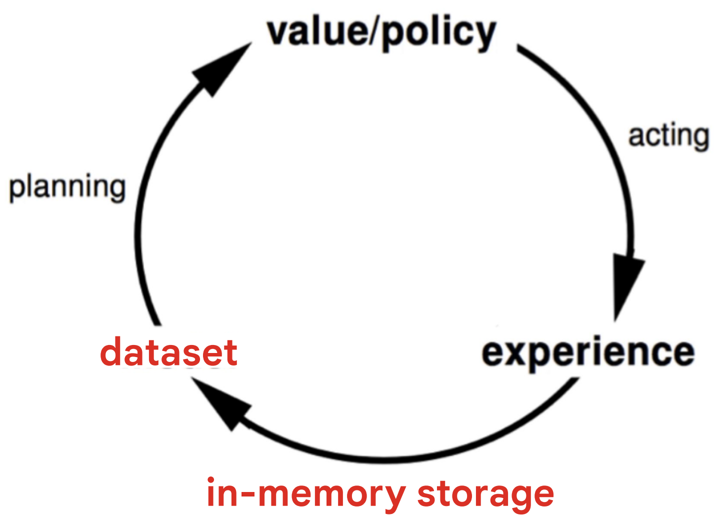
Experience Replay

A popular solution is to store experience in an experience replay buffer

- then compute updates on data sampled uniformly from the buffer
- this allows to break correlations between consecutive updates
- allows to improve data efficiency by allowing to reuse samples more than once

Experience Replay vs Model Based RL





Google DeepMind

Model learning as an auxiliary task

Last lecture we also discussed the challenges of state representation

- the agent will need to summarise the past history into a *state* $S_t = f(H_t)$
- this state representation $S_t = f(H_t)$ has to be learned jointly with the values
- the quality of the state representation is critical
    - but at the start of training when the agent doesn't see many rewards the signal is limited
- it is useful to share the representation with other auxiliary tasks
    - predicting different aspects of the stream of experience
- learning a model can be one of these task
    - by sharing the state representation between the model and the values
    - the state representation gets a dense high quality signal from the model learning even when the behaviour is poor and few rewards are seen

Google DeepMind

03 - *Local* Planning

Matteo Hessel
Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L
www.decoding-intelligence.com

Planning for action selection

So far the model was used to improve a global value function via planning

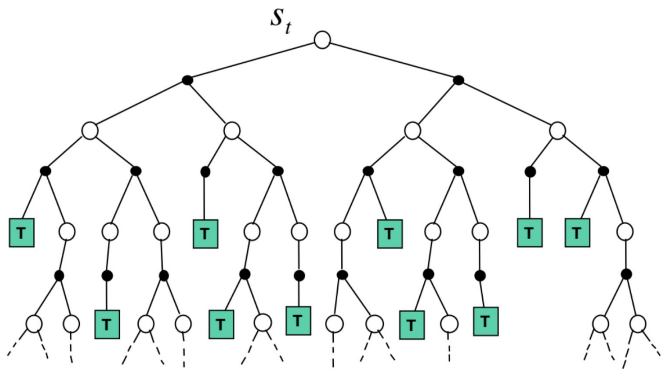
- If a good model is available it can also be used directly for action selection
- By dynamically replanning in each state to select the best next action

The local non parametric value function constructed by planning might be easier to construct than distilling all real and fictional experience in a single global value function

Forward search

A forward search algorithm can do so on each step by

- Taking the current state as root
- Constructing a complete search tree from that root
  by using the model to unroll possible next states for different sequences of actions
- You can then pick the action that results in the highest return

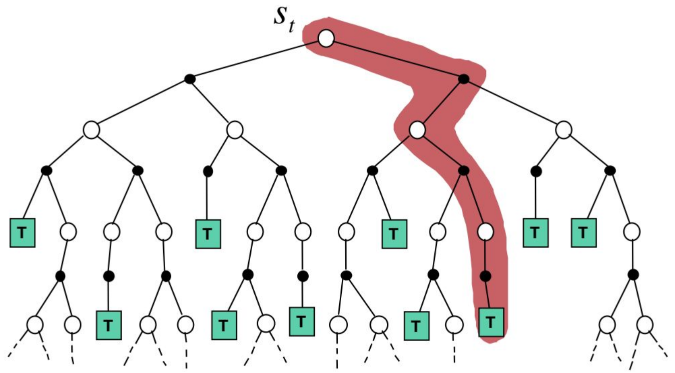


Google DeepMind

Simulation-based search

Forward search paradigm uses sample-based planning instead of constructing the full tree
On each step:

- The agent uses the model to simulate episodes of experience from the current state
- After each episode, it applies model free RL to update the values in the partial tree
- Uses the updated value estimates to guide which actions to expand in the next episode
- When the simulation budget is exhausted you pick the action with the highest value estimate in the root



Using global value estimates to guide search

At each episode simulation based search

- extends the partial lookahead tree
- updates the values for inner nodes in the tree

But the search algorithm itself doesn’t know which actions to pick once an episode reaches a leaf node

- Picking randomly at the leaf nodes is very inefficient
- If you simultaneously train a global value function
- You can then pick the action with the highest value according to the global estimator

Using global value estimates to truncate search

On each episode simulation based search unrolls the model all the way to a terminal state

- It’s computationally very expensive if each “episode” consists of thousands of agent-env interactions
- If the model is inaccurate, the simulation accuracy degrades as you get father aways from the root

Simultaneously learning a global value function (e.g. via model-free RL) can then help

- Allowing to truncate simulations at a fixed depth
- And then using the estimated values at the leaf states
  to guess how much more return could have been collected from there onwards

Closing the loop

We have seen that you can use a model to

- Plan a global value function by generating fictitious experience (e.g. via Dyna)
- Construct a local value function on each step to pick the next action (e.g. via search)

And we have seen how you can use the global value function to enhance the search process

You can also do the opposite

- Use the very accurate local value function to construct high quality targets for you global value function
- For instance in your Q-learning algorithm you could maximize over the Q-values in the root (as computed by simulation based search)

$$Q(S, A) \rightarrow R + \max_{a'} Q^{search}(S', a')$$

Google DeepMind

Google DeepMind

04 - Value equivalent models

Matteo Hessel

Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L

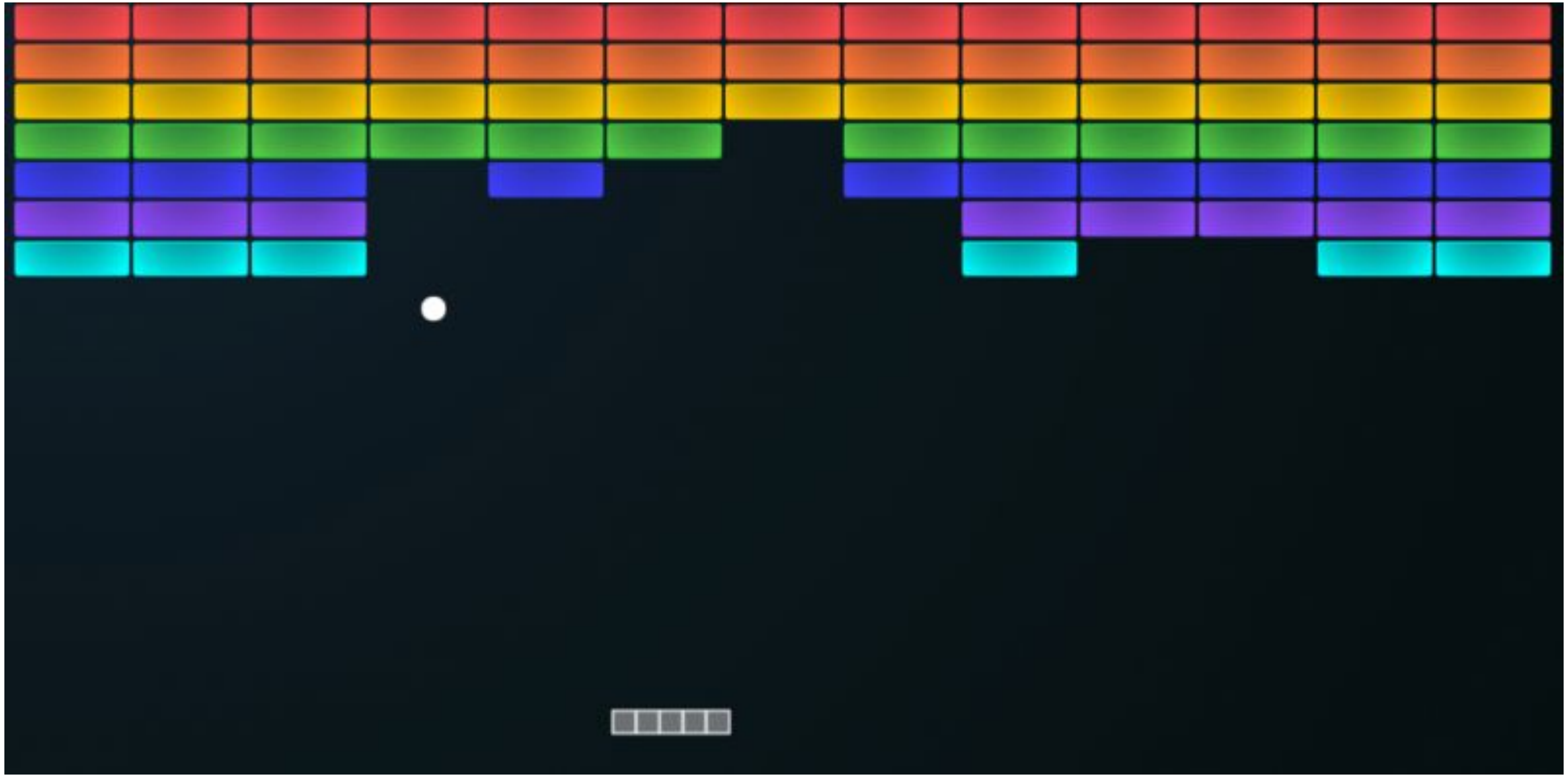
www.decoding-intelligence.com

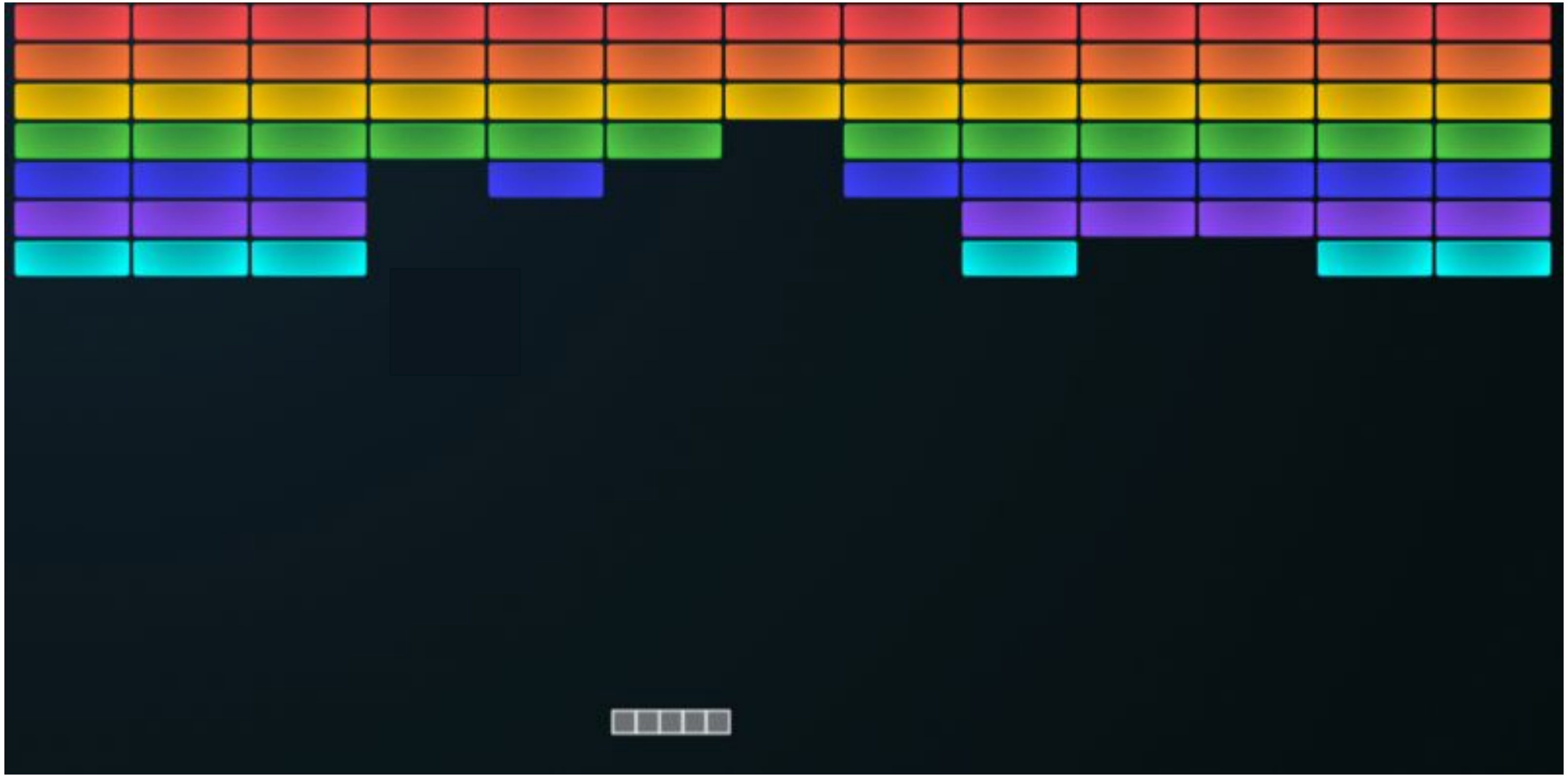
Challenge 1: learning a “good” model

Learning models from raw observations in complex environments is **hard** because:

- Observations are high dimensional
- Often most of the pixels in an observation are irrelevant

How do we make the model **focus** on what we care about for planning?





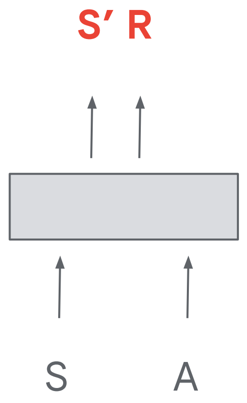
What do we actually want from a model?

We defined a model as a component that given a state and action returns the next state **S'** and immediate reward **R**.

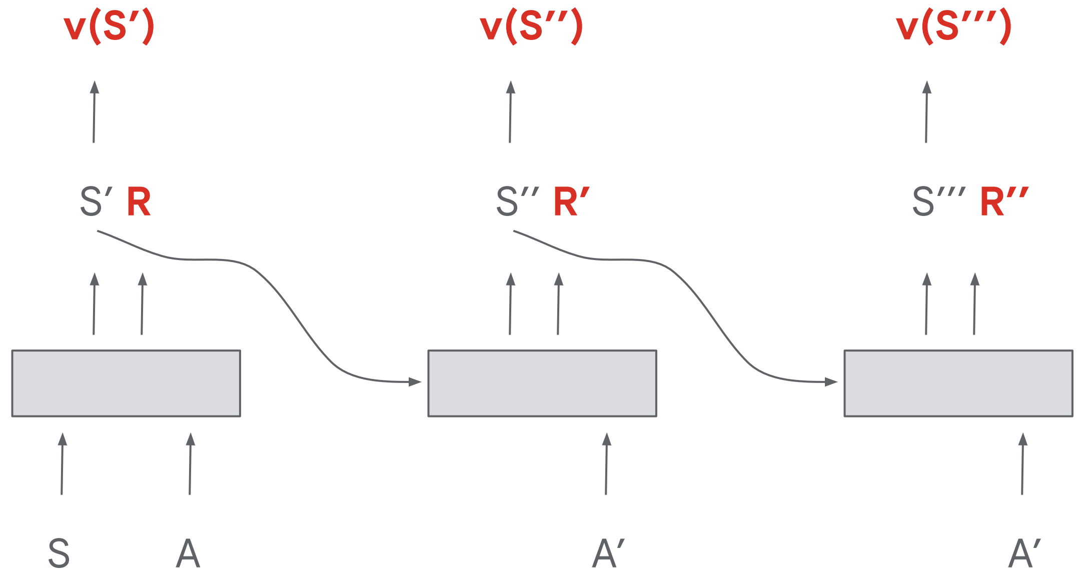
But we actually rarely use the next state directly!

- **Dyna**: the next state will is fed into a value function to construct the bootstrap target
- **Search**: the next state is fed into a the model again and again until we truncate again by bootstrapping

If we only use the predicted states to then make value predictions we should train the model to be good at that!



Value equivalent models



MuZero

MuZero puts together most of the ideas we discussed today in a single agent

- Train a value equivalent model
- Use search over the value equivalent model to pick actions
- Distill the local values and action probabilities into a global policy and value functions
- Use the policy and value functions to guide search

This results in one of the strongest RL agents in the literature on a wide range of domains

- Board games (Go, Chess, ...)
- Video games (Atari, ...)
- Continuous control (Mujoco, ...)
- Real world applications (Video compression, ...)

Thank you.

Google DeepMind

Google DeepMind