Learning model-based planning from scratch

SAPIENZA
UNIVERSITÀ DI ROMA

Learning model-based planning from scratch

- Model-based planning involves using a model of the environment to generate a plan (sequence of actions) that will achieve a desired goal.
- Learning a model of the environment can be difficult in practice, especially when starting from scratch with no prior knowledge.

Ref : (https://arxiv.org/abs/1707.06170)

SAPIENZA
UNIVERSITÀ DI ROMA

- *Imagination-based Planner (IBP)*

- Learn across episodes and learn the actual planning strategy itself

- All aspects of the planning process (construct a plan)

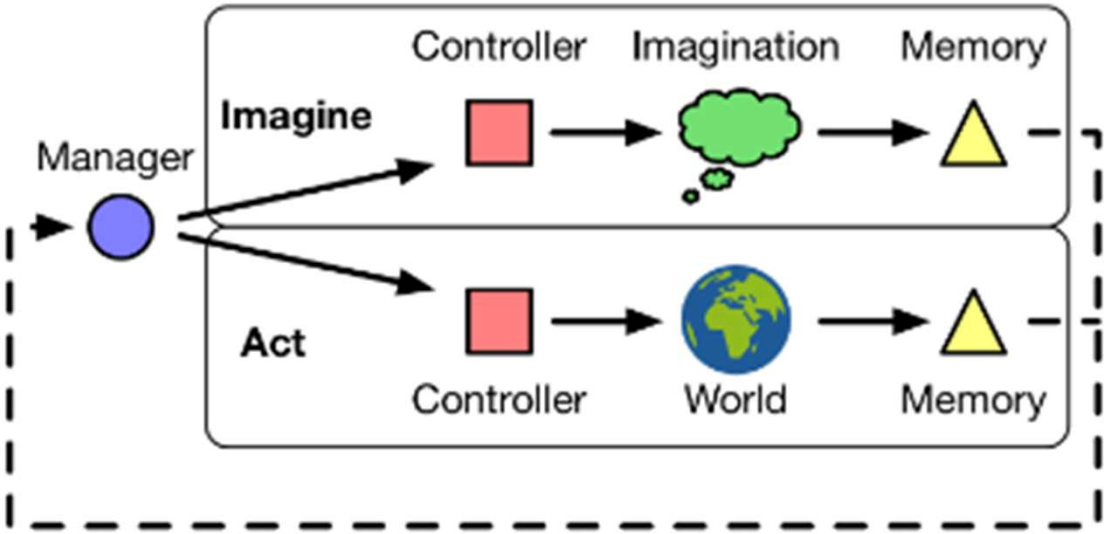
- The model of the environment is used in two ways:
  - for imagination-based
  - planning and gradient-based policy optimization (train the controller)

SAPIENZA
UNIVERSITÀ DI ROMA

- Planning with ground truth models : AlphaGo

- Learn to plan :
  - “Dyna”
  - Vezhnevets et al.

- “Imagination-based metacontroller” (IBMC) , Hamrick et al.



- IBP implements a recurrent policy capable of planning via four key components .
- At each step, the manager chooses whether to imagine or act.
- If acting, the controller produces an action which is executed in the real world.
- If imagining, the controller produces an action which is evaluated by the model-based imagination.
- Data resulting from each step are aggregated by the memory and used to influence future actions.
- The idea is to create an history h, that contains all the informations up to that step

SAPIENZA
UNIVERSITÀ DI ROMA

- The **manager** is a discrete policy which maps a history, h ∈ H, to a route, u ∈ U.
- The u determines whether the agent will execute an action in the environment, or imagine consequences of a proposed action.
- If imagining, the route can also select whether to imagine from the last real state or from the last imagined
- For us MLP trained with REINFORCE (that takes also state as input) but with modified rewards

```
class Manager(nn.Module):
  def __init__(self, state_size, history_size, n_decisions):
    super().__init__()
    self.fc1 = nn.Linear(state_size + history_size, 8)
    self.fc2 = nn.Linear(8, n_decisions)
    self.softmax = nn.Softmax(0)

  def forward(self, x):
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    return self.softmax(x)
```

SAPIENZA
UNIVERSITÀ DI ROMA

- The **controller** is an action policy which maps a state s ∈ S and a history to an action, a ∈ A.
- The state which is provided as input to the controller is determined by the manager's choice of u.
- For us MLP trained by Reinforce

```
class Controller(nn.Module):
  def __init__(self, state_size, history_size, action_size):
    super().__init__()
    self.fc1 = nn.Linear(state_size + history_size, action_size)
    self.softmax = nn.Softmax(0)

  def forward(self, x):
    x = F.relu(self.fc1(x))
    return self.softmax(x)
```

SAPIENZA
UNIVERSITÀ DI ROMA

- The **imagination** is a model of the world, which maps states, $s \in S$, and actions, $a \in A$,
  to consequent states, $s' \in S$, and scalar rewards, $r \in R$.
- For us a MLP trained with Mean Squared Error only on real data obtained from the real env

```python
class Imaginator(nn.Module):
  def __init__(self, state_size):
    super().__init__()
    self.state_imaginator = nn.Linear(state_size + 1, state_size)
    self.reward_imaginator = nn.Linear(state_size + 1, 1)

  def forward(self, state, action):
    input = torch.cat([state,action])
    s_prime = self.state_imaginator(input)
    reward = self.reward_imaginator(input)
    return F.relu(s_prime), F.relu(reward)
```

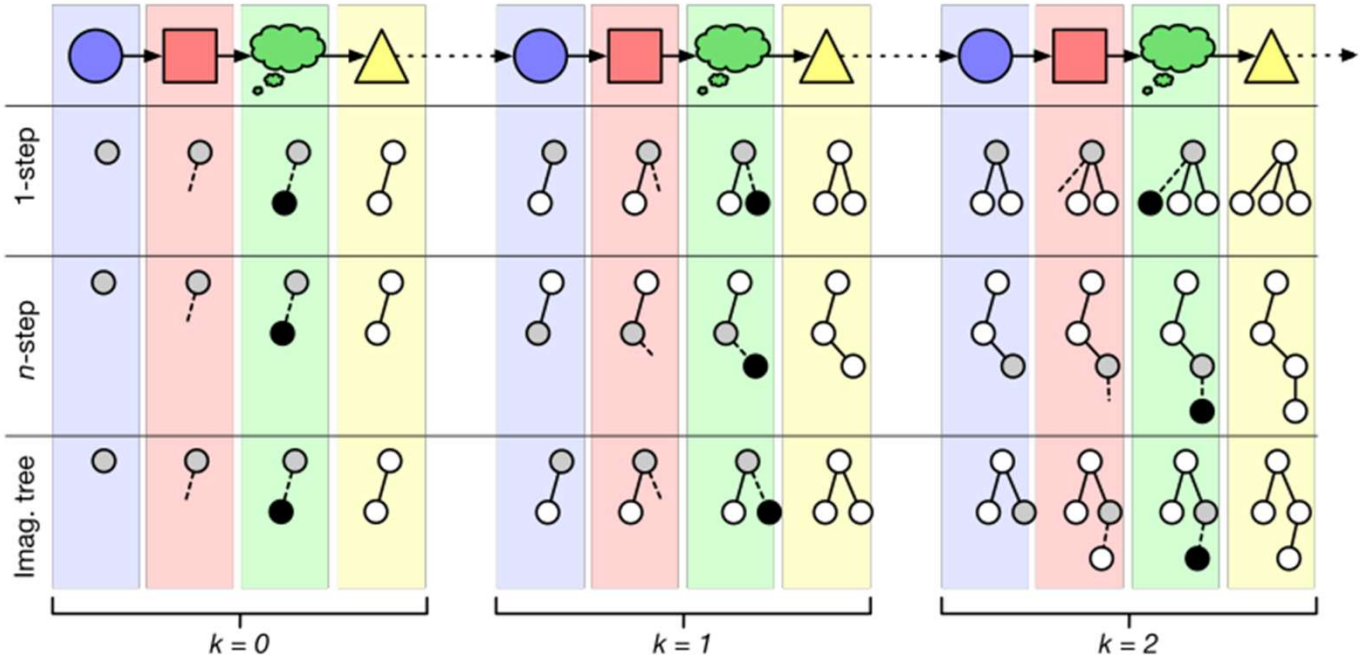
- The **memory**, $\mu : D \times H \rightarrow H$, recurrently aggregates the external and internal data generated from one iteration, $d \in D$, to update the history $h_i = \mu(d_i, h_{i-1})$
- For us an LSTM

```python
class Memory(nn.Module):
  def __init__(self, new_data_size, history_size, hidden_size=32, num_layers=4):
    super().__init__()
    self.lstm = nn.LSTM(new_data_size, hidden_size, num_layers, batch_first=False)
    self.fc = nn.Linear(hidden_size, history_size)
    self.num_layers = num_layers
    self.hidden_size = hidden_size

  def forward(self, x):
    # Set initial hidden and cell states
    h0 = torch.zeros(self.num_layers, self.hidden_size)
    c0 = torch.zeros(self.num_layers, self.hidden_size)

    # Forward propagate LSTM
    out, _ = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, seq_length, hidden_size)
```

- There are different types of **imagination strategies**:
  - 1-step imagination
  - n-step imagination
  - imagination tree
- Grey nodes indicate states chosen by the manager (blue circle) from which to imagine
- Dotted edges indicate new actions proposed by the controller (red square)
- Black nodes indicate future states predicted by the imagination (green cloud icon)

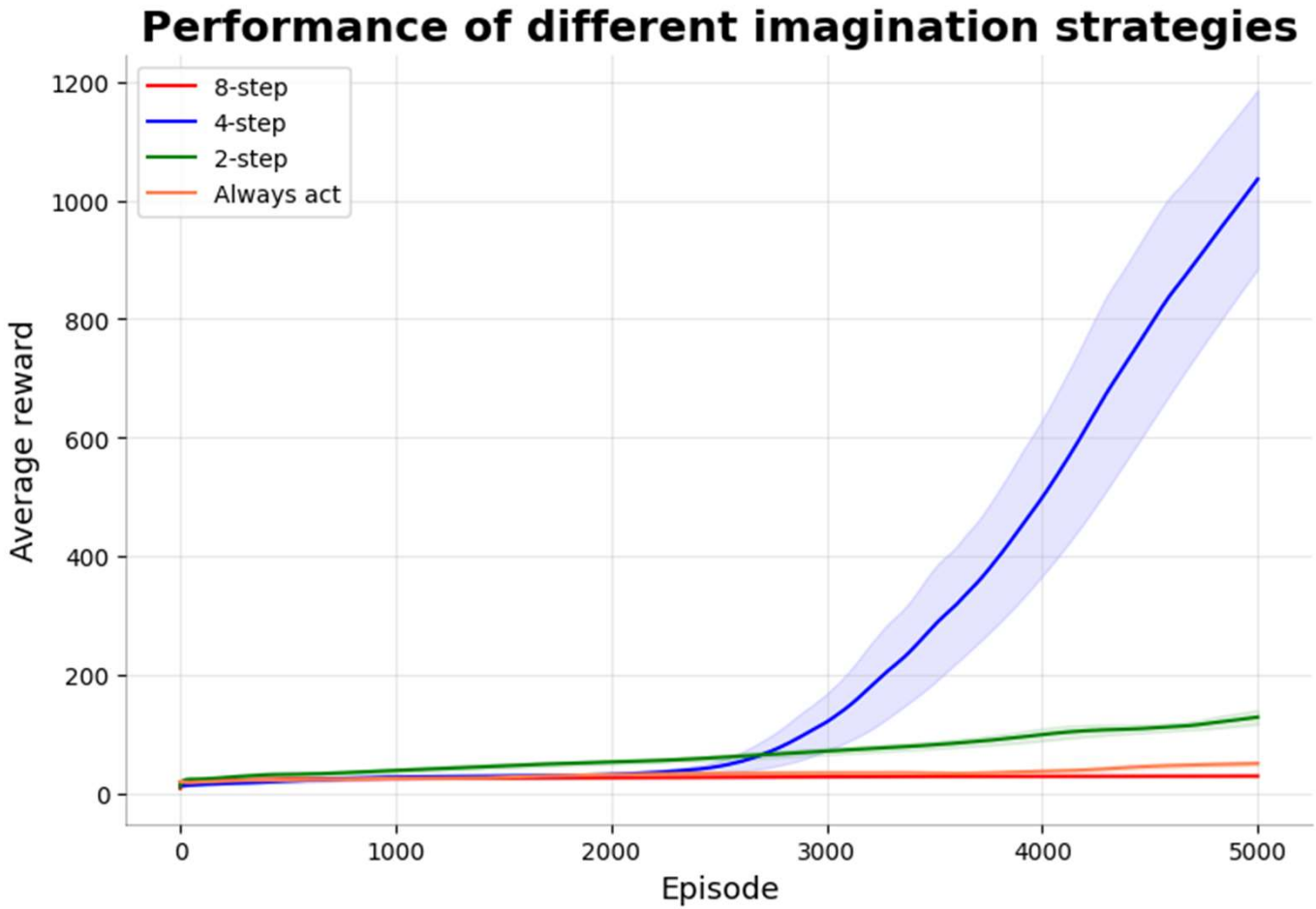


SAPIENZA
UNIVERSITÀ DI ROMA

Our Experiments:

- The first attempt was with **CartPole** environment, we know it's an "easy" one but this way you can understand if our model works.
- Max 2000 steps per episode , 5000 episodes in total
- Exponential Moving Average to reproduce the results of the paper

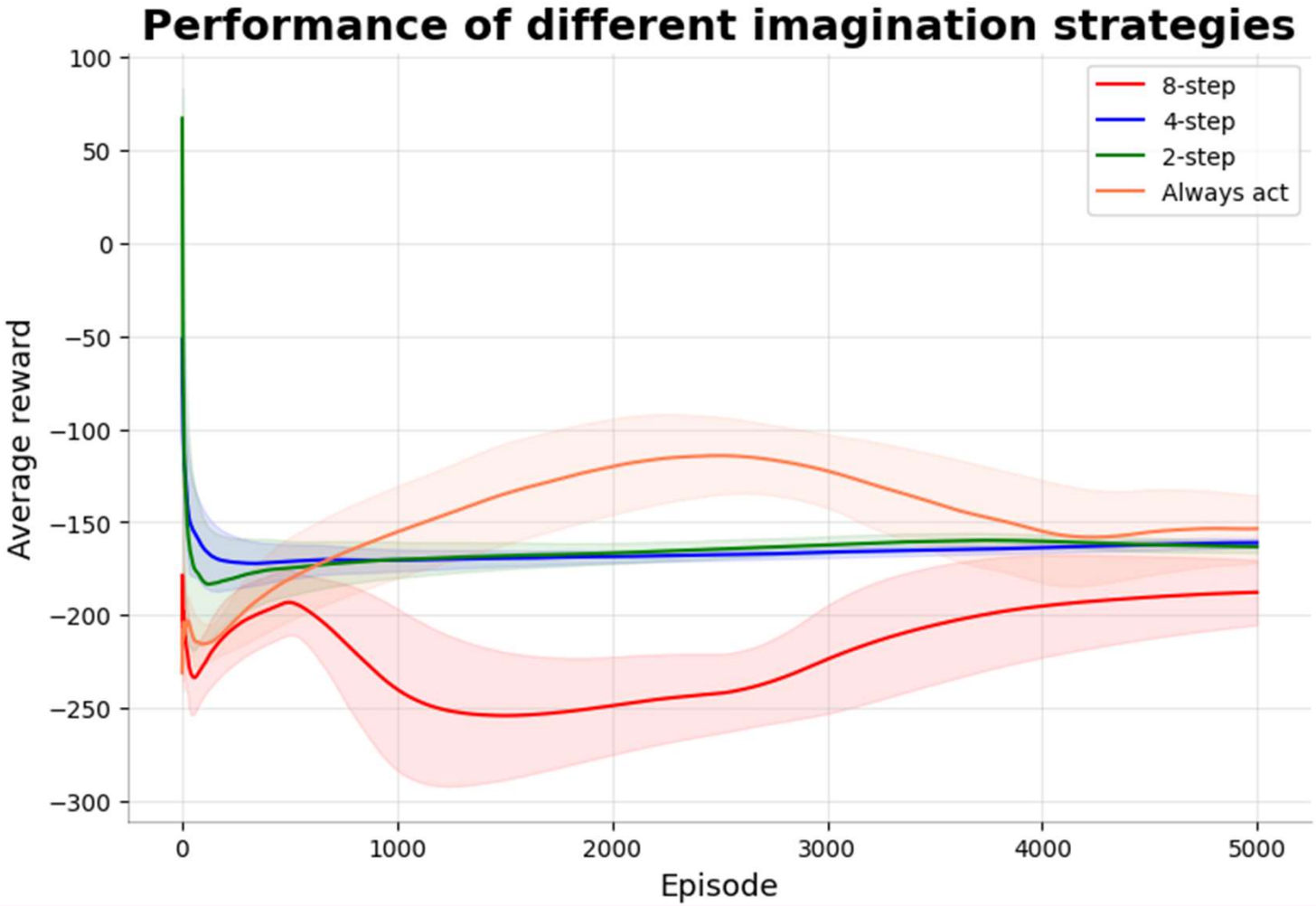
Note : Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step is allotted.



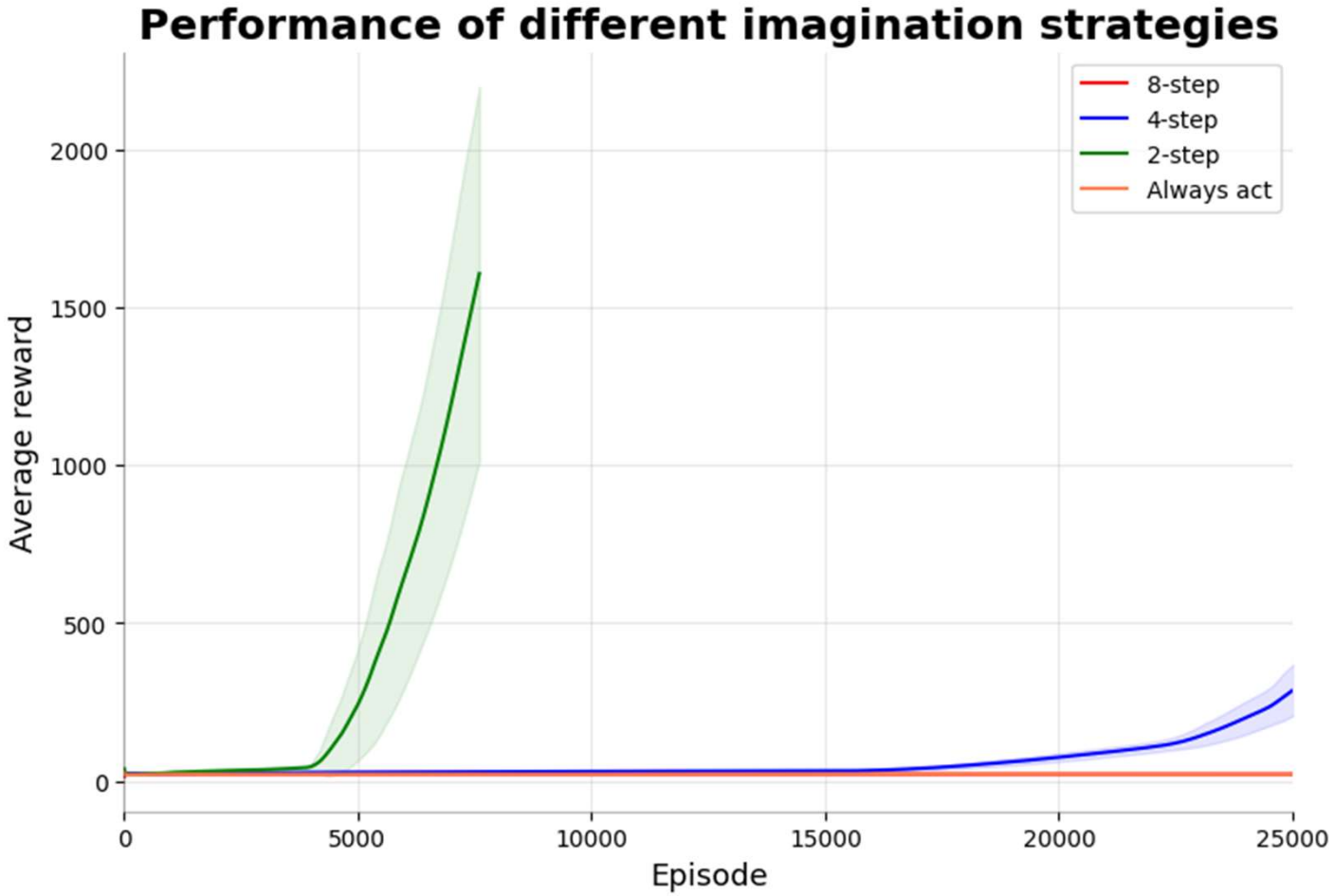
SAPIENZA
UNIVERSITÀ DI ROMA

- Second attempt was with **Lunar Lander**
- Max 2000 steps per episode , 5000 episodes in total
- Exponential Moving Average to reproduce the results of the paper

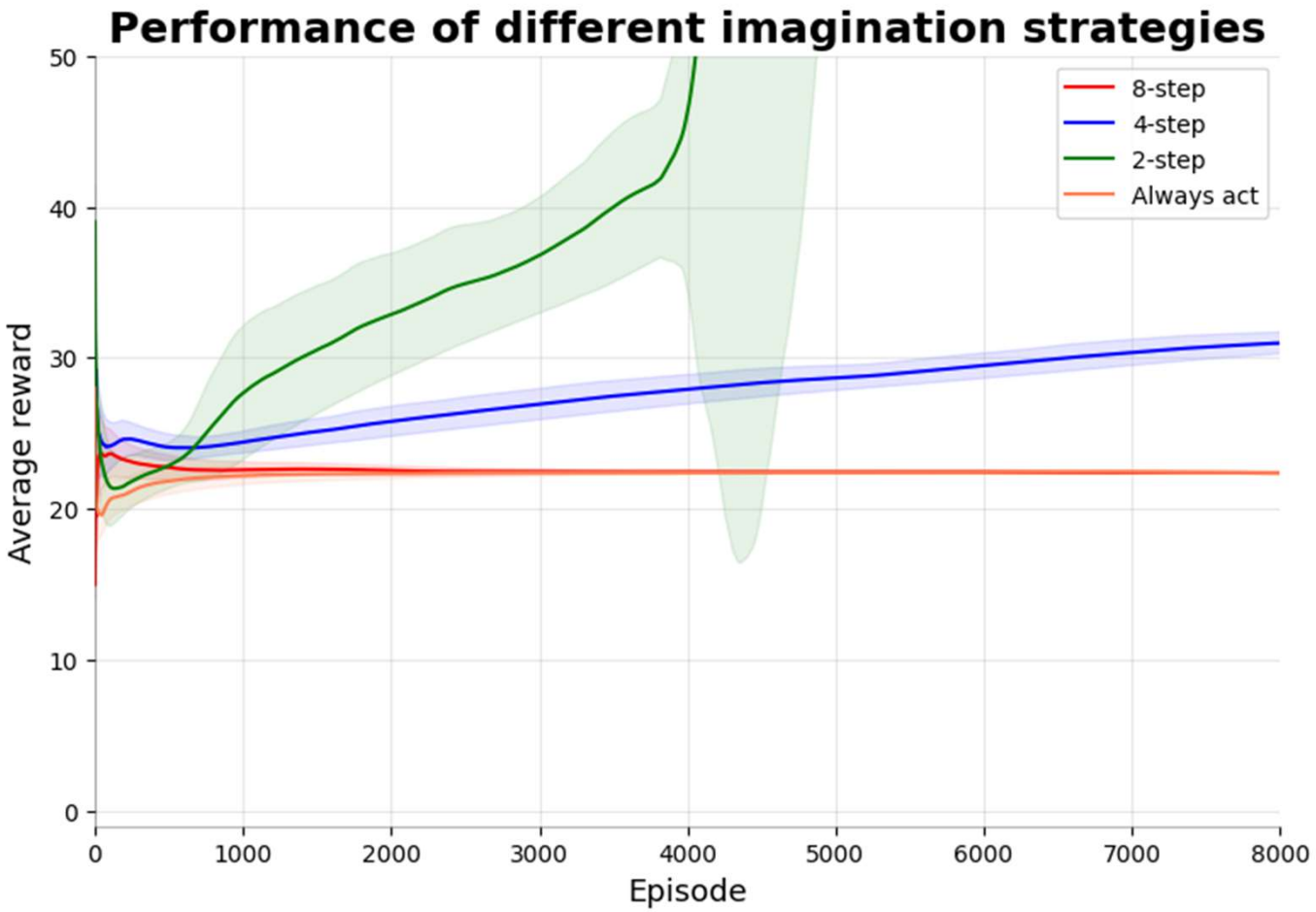
Note : Reward for moving from the top of the screen to the landing pad and coming to rest is about 100-140 points. If the lander moves away from the landing pad, it loses reward. If the lander crashes, it receives an additional -100 points. If it comes to rest, it receives an additional +100 points. Each leg with ground contact is +10 points. Solved is 200 points.



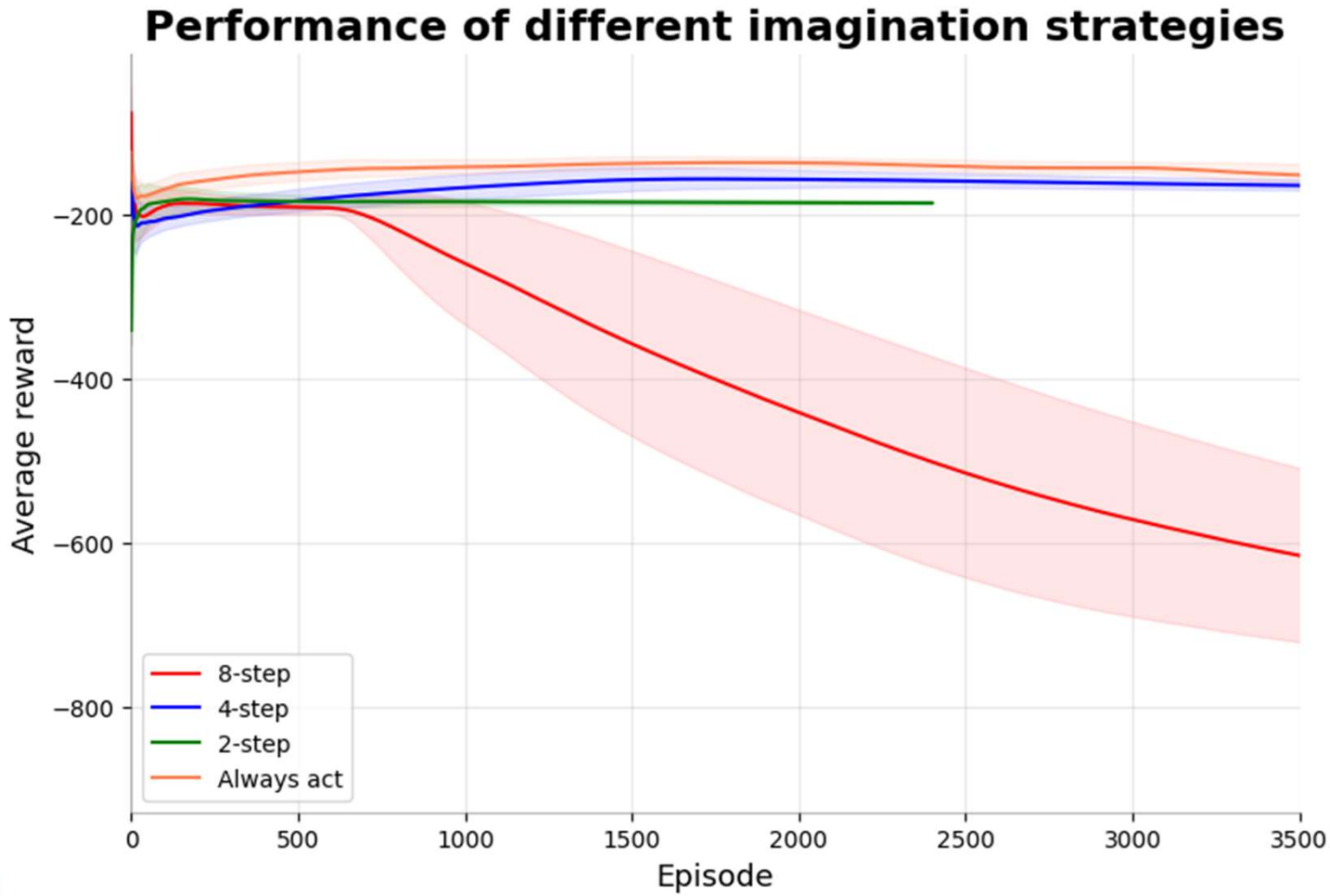
- **CartPole** training second attempt
- Max 5000 steps per episode , 50000 episodes in total
- 2-step differs substantially from the others



- We zoomed in on the plot to also show 8-steps
- Together with always act they have a much worse performance



- **Lunar Lander** second training attempt, but some need more than 12h
- Max 5000 steps per episode - 50000 episodes in total
- Slightly better results than before for 4-step

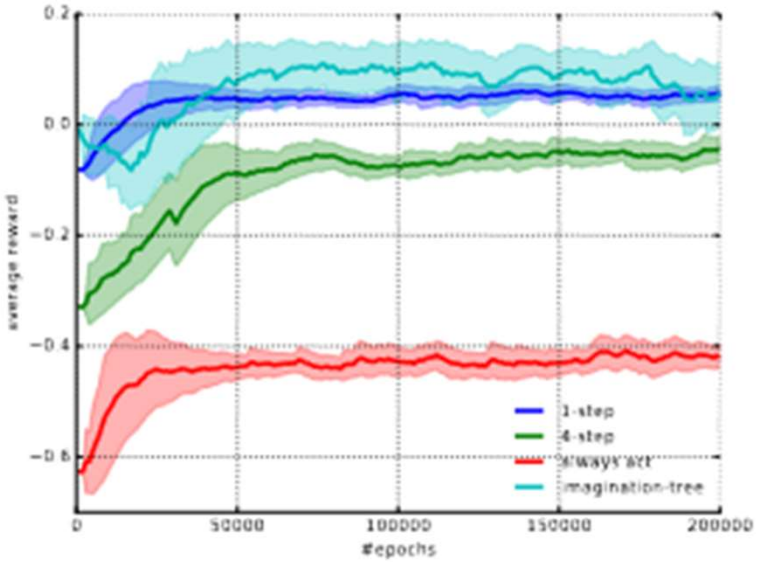


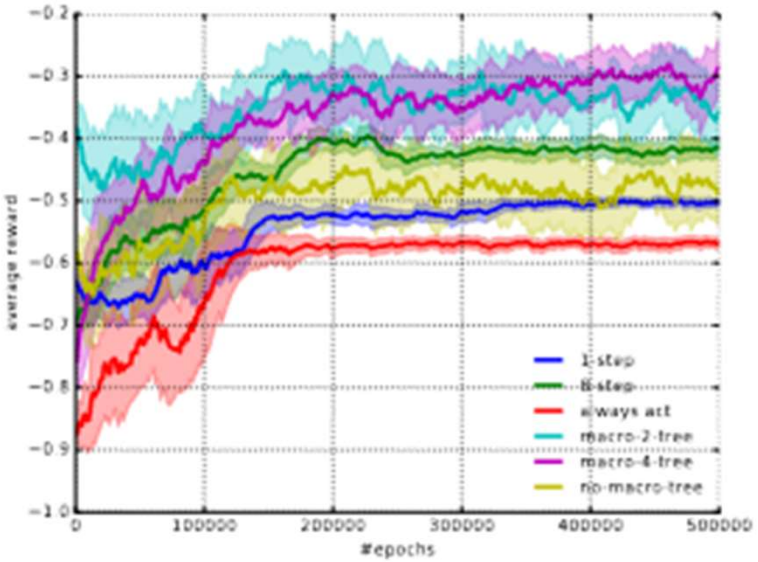
SAPIENZA
UNIVERSITÀ DI ROMA

Computational time of the algorithm

With the resources at our disposal we could not train for than 50,000 steps (about 12h for 30,000).

These are the results of the paper in maze 5x5 (left) and 7x7 (right)





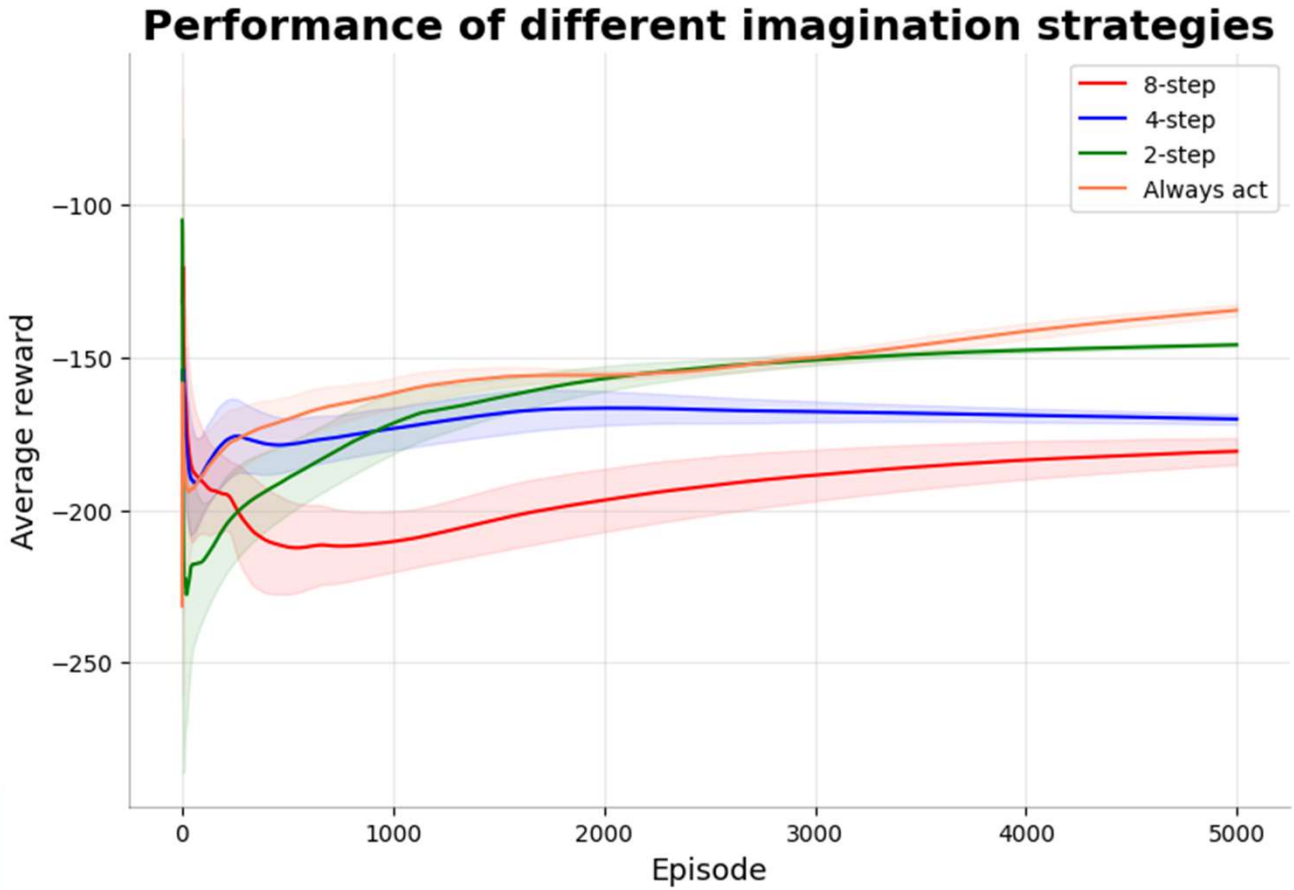
We can see that they have been driving for a lot of episodes getting very small reward improvements.

In particular in 7x7 their reward never rises above zero.

SAPIENZA
UNIVERSITÀ DI ROMA

Enviroment parameters

- By default we have : gravity = -10.0, enable_wind = False, wind_power = 15.0, turbulence_power = 1.5.
- We try to set gravity = -11.0 and all others equal to zero.
- 5000 episodes, 2000 max steps each



SAPIENZA
UNIVERSITÀ DI ROMA

Surely our IBP is not working well on Lunar Lander..

- A “good” starting point is that an **easy environment** can be solved
- Difficult to find the optimal **learning rate**. if it’s high it learns fast but perhaps jump a minimum
- **Scheduler** for the optimizer with exponential decay
- **Environment parameters** such as gravity and wind
- Not sufficient number of **episodes**
- **Cost applied** to rewards needs some engineering (constant penalty forces the manager to avoid imaging as the n-episodes increases)
- Imaginator is a Interaction Network (for us instead is an MLP)
- Initialization of weights in networks is random

SAPIENZA
UNIVERSITÀ DI ROMA

Our tentatives to solve the problem…

- We tried to apply constant,linear and logarithmic imagination reward penalization for the manager
- Play with the learning rate
- Remove the Imaginator network
- Play with environment parameters
- And the most aggressive one…. force the controller to do action that will not correspond to “do nothing” action, in order to use always the engines of the lunar spaceship

SAPIENZA
UNIVERSITÀ DI ROMA

Our best model:

IBP trained on CartPole-v1 with the following details:

- n_episodes = 5000
- n_step_per_episode = 2000
- lstm_units = 4
- n_step_strategy = 4
- lr_manager = 0.001
- lr_imaginator = 0.001
- lr_memory_and_controller = 0.003
- γ = 0.99
- Adam optimizer