Deep Reinforcement Learning

Olof Harrysson 2018

Presentation with videos

Reinforcement Learning (RL)

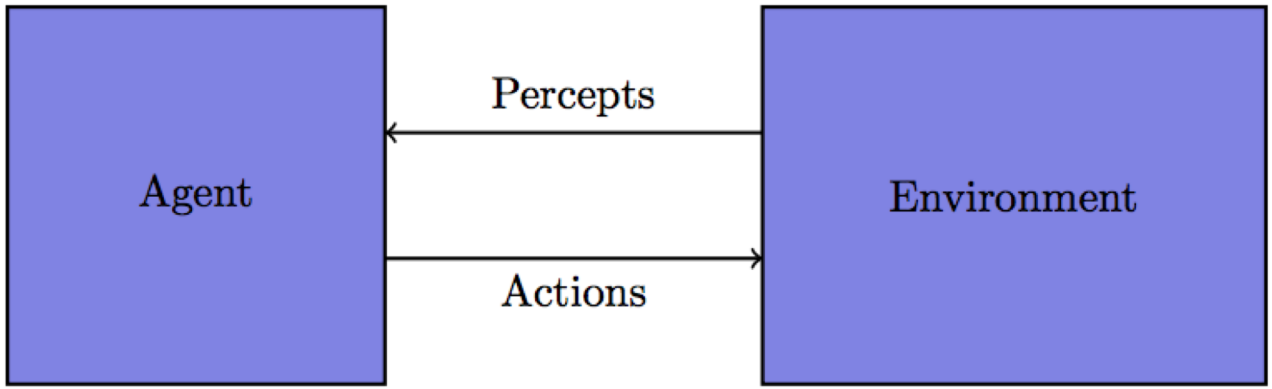
Reinforcement learning is a branch of machine learning where an agent learns by trial and error, much like humans do. Good actions are rewarded positively whilst bad actions are punished

Agents are most often trained in a simulated reality called an environment. This makes it easy to reset the simulation compared to resetting a physical object

However, once a trained agent is uploaded to a physical robot it generally doesn’t perform well due to differences in the simulated world and the real one

Agent ⟷ Env

1. Environment sends agent a state that describes the world
2. Agent use the state to decide an action and performs this action in the environment
3. Environment update its state and gives the agent a reward corresponding to the action taken



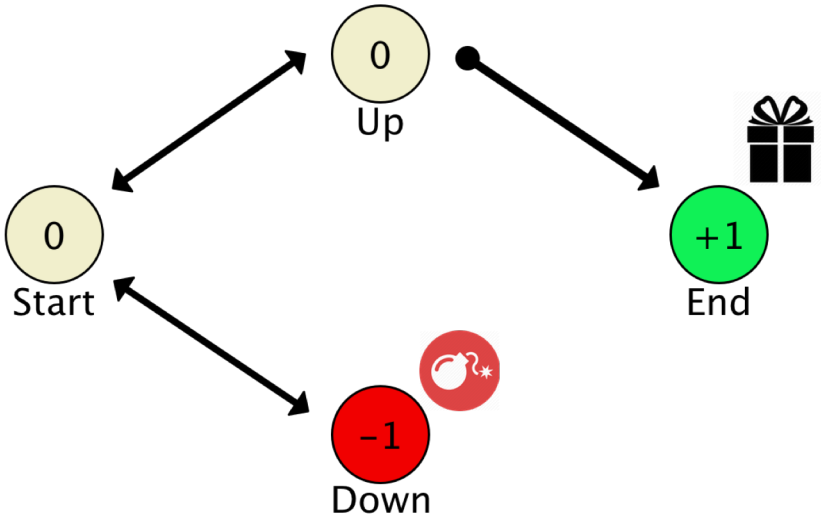
Q-learning

Imagine an **agent** whose goal is to walk to a green circle. Once there, it gets a **reward** of +1. The position of the agent determines the **state** of the game. To change the state, the agent can take an **action**. Walk up, down, left, right

On its way to the green circle the agent wants to avoid the red circle since going there will give it a negative reward. It also wants to avoid going back and forth between the two white circles

Q learning is an algorithm that helps the agent evaluate state and action pairs to make it possible to choose the optimal action in whichever state. “If I’m in start, I should go up”

All the state and action pairs are saved to remember how good they were



Function Approximation

Q-learning doesn’t work for large state spaces. The algorithm simply needs to much computational time to converge

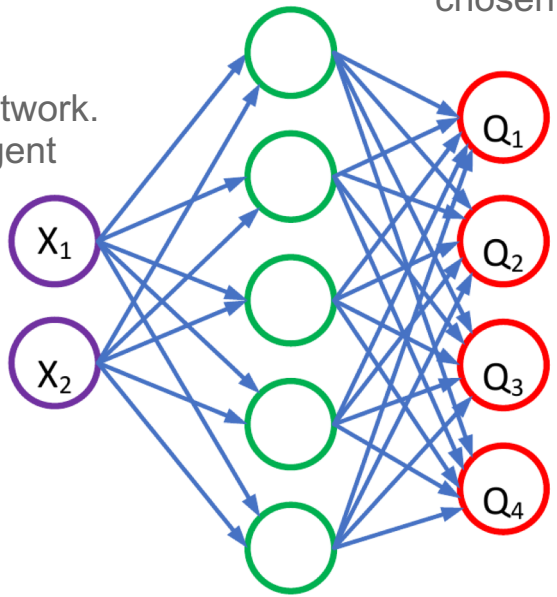
Continuous state spaces would have an infinitely large state space and is therefore not solvable with traditional Q-learning. One could approximate the states into a finite amount of bins and from there do Q-learning. Unfortunately this also often results in a large state space

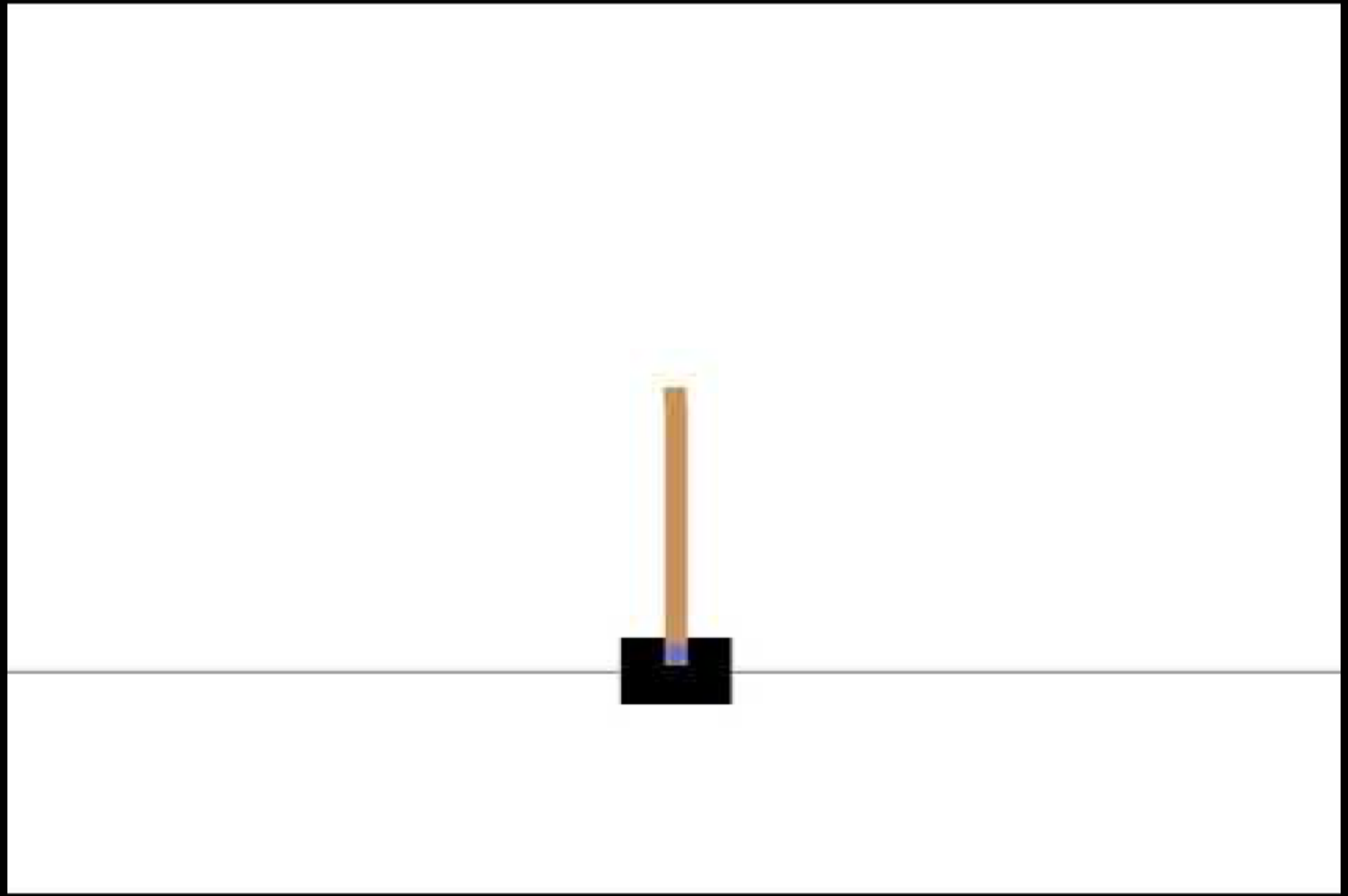
In deep Q-learning, neural networks are used as function approximators to get around this problem

Deep Q-network (DQN)

The output is corresponding Q values for available actions. The highest Q-value is chosen as action taken

The state is the input to the network. E.g. position, velocity of the agent





Training a Deep Q-Network (DQN)

- Similarly to traditional Q-learning, the Q(s,a) values are calculated with the function $Q(s_t, a_t) = reward + \gamma \times max_a\, Q(s_{t+1}, a_{t+1})$

- The network is then updated with s_t as input and $Q(s_t,a_t)$ as label

Training a Deep Q-Network (DQN)

- Problem - DQN suffers from is that states are highly correlated due to the episodic nature of the environments. The current state will influence what the next state will be. This introduces bias into the model and easily leads to the DQN learning a sub-optimal behaviour or overfitting to current episodes

- Solution - Experience Replay
  - We give the agent a memory and save the last M seen transition data e.i. state, action, reward, next state
  - Instead of updating the network on the current transition, we sample N transitions from our memory and update the DQN in a batch setting
  - This reduces overfitting on single transitions and a correlated episode

Training a Deep Q-Network (DQN)

- Problem - Using the network to calculate states' values and also update the network can create feedback loops. This often leads to Q-values exploding in size which decreases performance

- Solution - Target Network
  - Create a second network identical in structure to the DQN but with different weights
  - Use the second network to calculate the label used for the primary network update
  - Update the second networks weights in a small direction towards the primary net's weights
  - The use of a target network reduces oscillation for the DQNs predicted Q values

TensorBoard SCALARS INACTIVE

Show data download links

Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing

0,951

Horizontal Axis

STEP RELATIVE WALL

Runs

1d923

1d923-DQN-agent-2018-05-20 11:58:39-avg

1d923-DQN-agent-2018-05-20 11:58:39-max

1d923-DQN-agent-2018-05-20 11:58:39-min

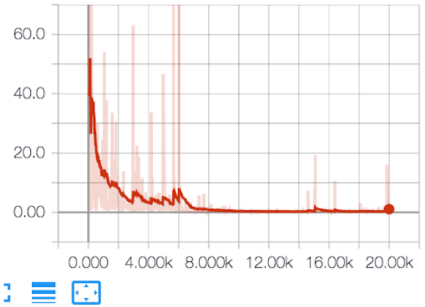
1d923-DQN-agent-2018-05-20 11:58:39-value

TOGGLE ALL RUNS

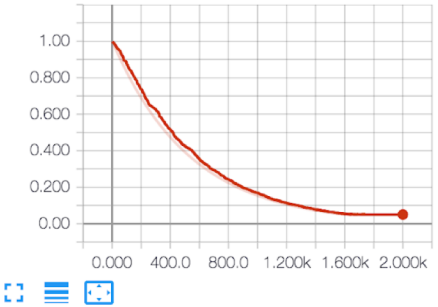
Loss|Q-val|Score|Noise|Action|Test

Tags matching /Loss|Q-val|Score|Noise|Action|Test/

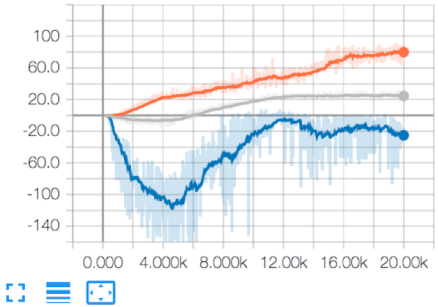
Loss



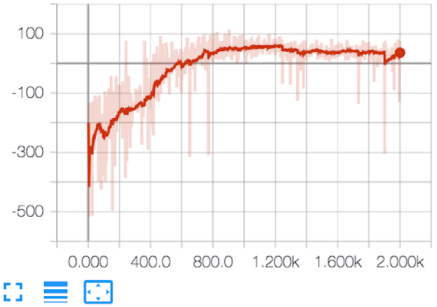
Noise



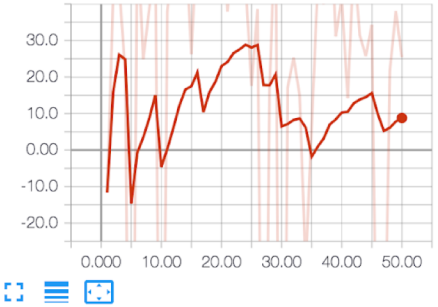
Q-val



Score



Test_score



Loss

Loss

Exploration vs Exploitation

- When should the agent take actions that it considers suboptimal to potentially uncover an even better strategy? This is an unsolved problem for unknown or stochastic rewards

- At what point should it stop the exploration and start doing the exploitation?

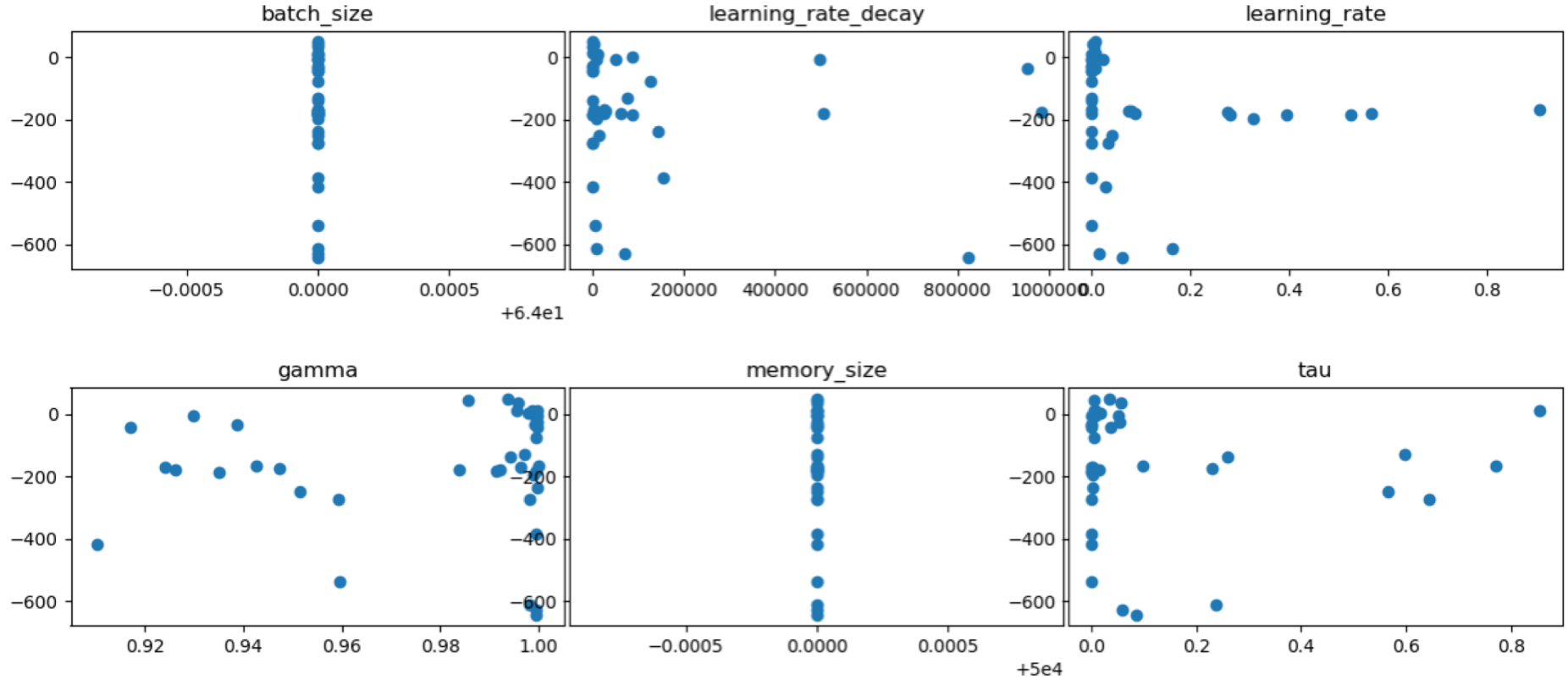
- Generally the agent starts doing a lot of exploration and gradually shifts its policy to exploitation

E-greedy Exploration

- With probability ε, take a random action. Otherwise take the action the agent considers the best.

- DQN started with ε=1 and lowered it whilst program ran until ε=0.05

Parameter Search

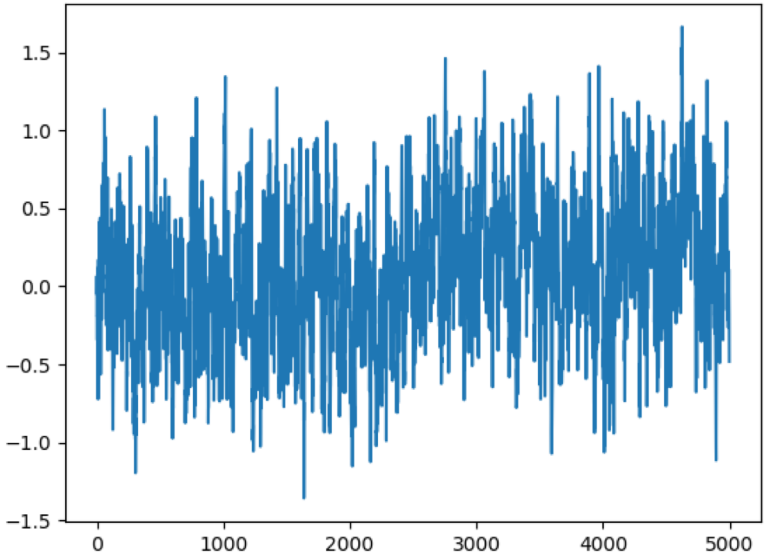


Deep Deterministic Policy Gradient (DDPG)

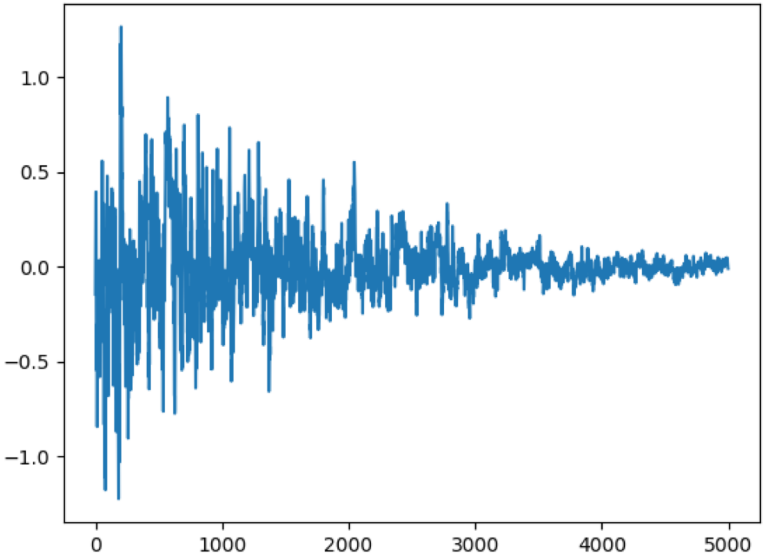
- Can take continuous actions
- Consists of an Actor and a Critic, each with its own neural network
  - Actor takes a state as input and outputs a continuous action
  - Critic takes both a state and action as input and outputs the value of taking the action in that state
- Critic is updated in the same fashion as the network in the DQN
- The actor is updated so that it gives that action that will lead to the highest value in the critic

Continuous Noise

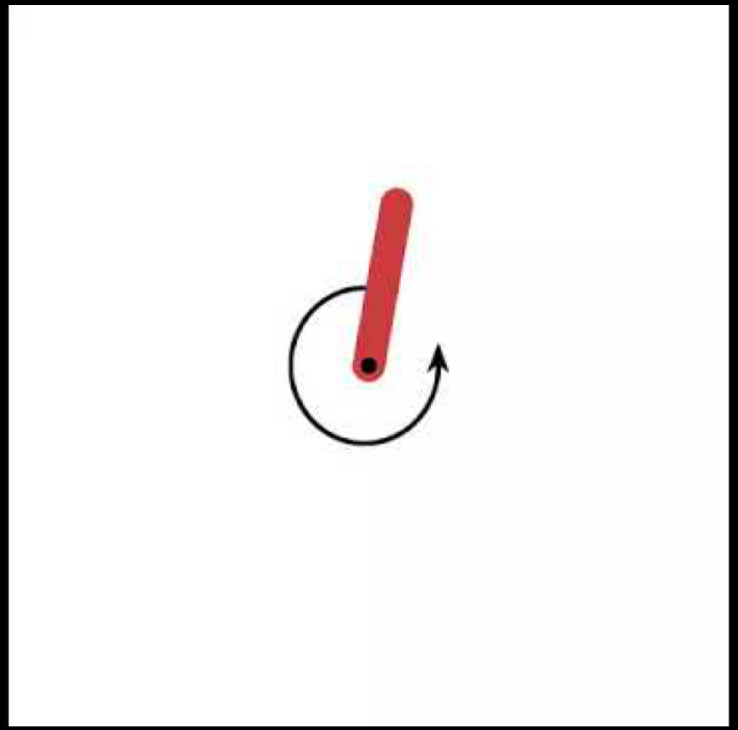
Ornstein Uhlenbeck



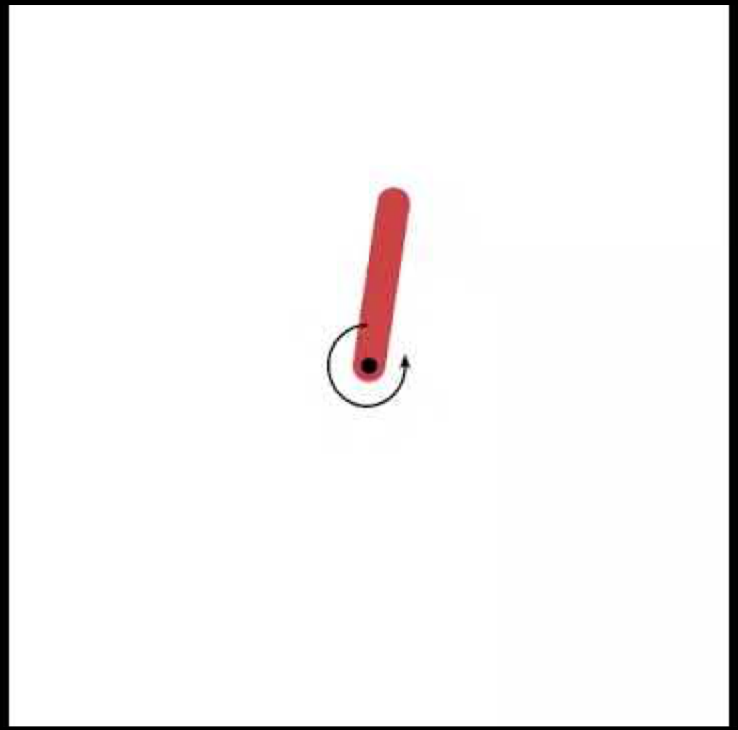
Reduced as program is run

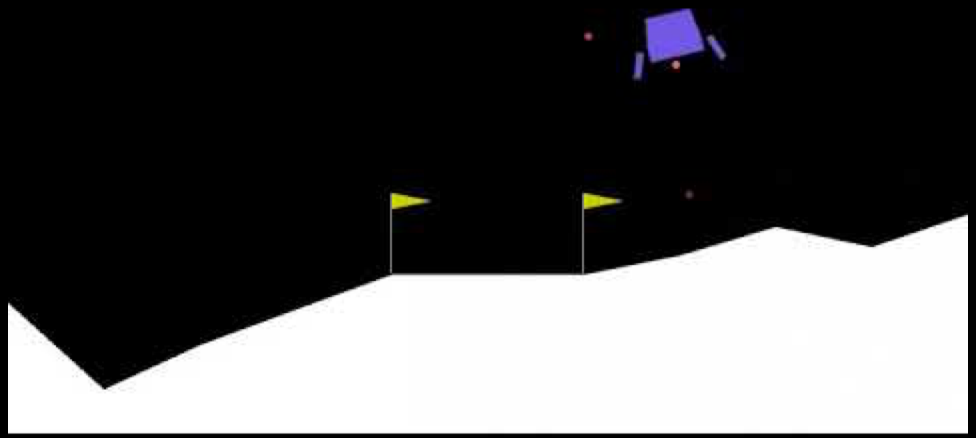


Without noise decay



With noise decay





Possible Improvements

- Specialized reward functions
- Prioritized experience replay
- Fine tune parameters for specific environments
- Different network shapes / settings

Thank you for your attention

Questions?