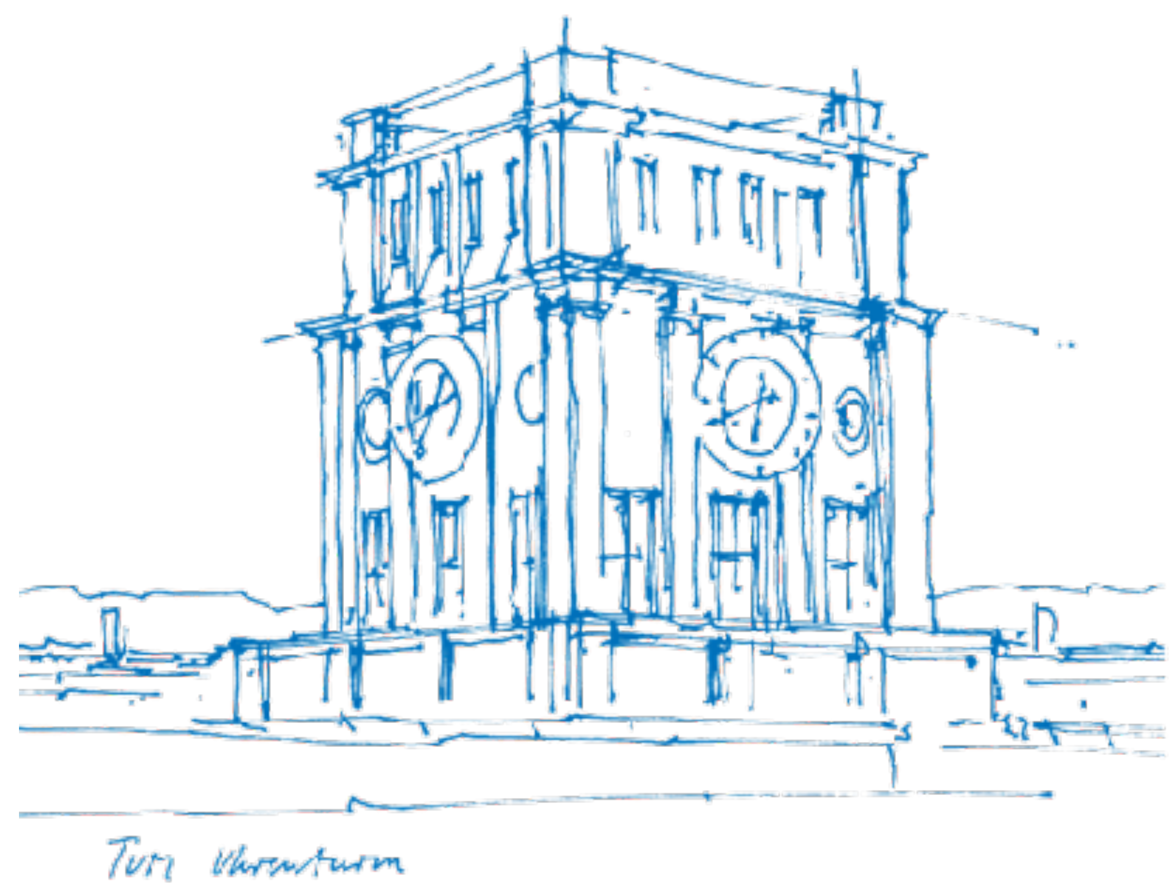
Reinforcement Learning for Adaptive Locomotion of a snake-like Robot using Tensorflow

Jonathan Rösner
Technische Universität München
Department of Informatics
Robotics and Embedded Systems
Garching, 24. November 2017



Outline

What is Reinforcement Learning?

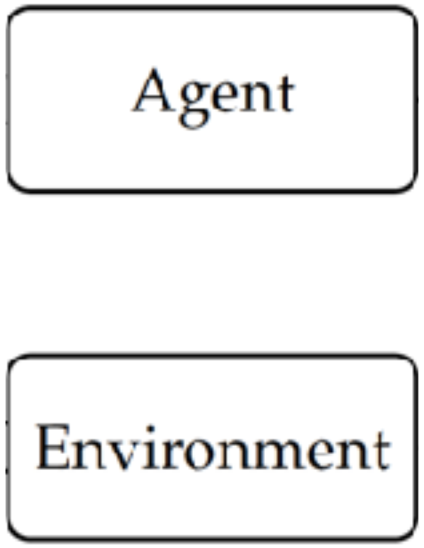
Agents and environments

What is Reinforcement Learning?

Agents and environments

Three Components:

- Environment
- Agent
- Time



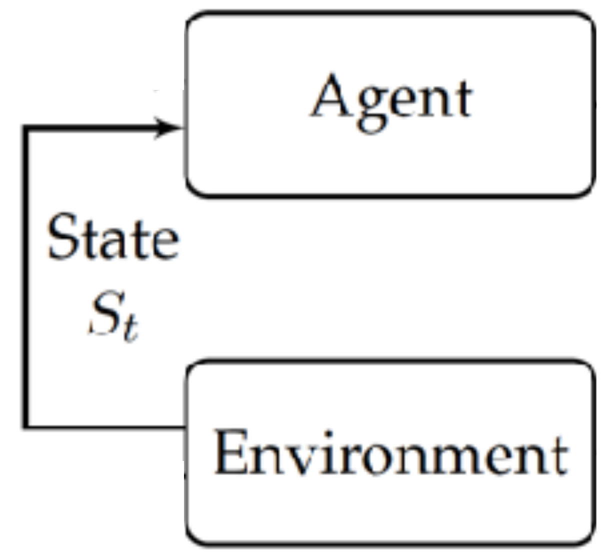
What is Reinforcement Learning?

Agents and environments

Three Components:

- Environment
- Agent
- Time

Agent can observe environment



What is Reinforcement Learning?

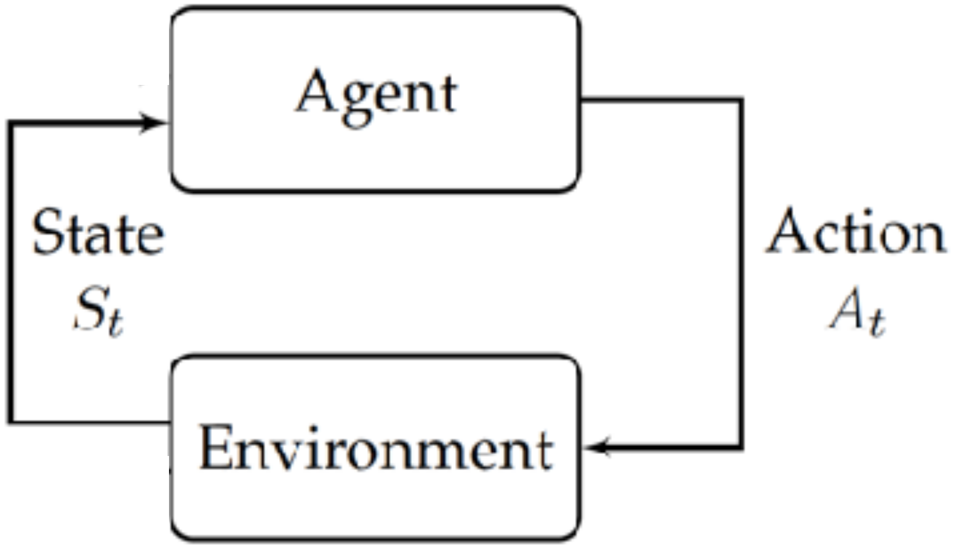
Agents and environments

Three Components:

- Environment
- Agent
- Time

Agent can observe environment

Agent can act upon environment



What is Reinforcement Learning?

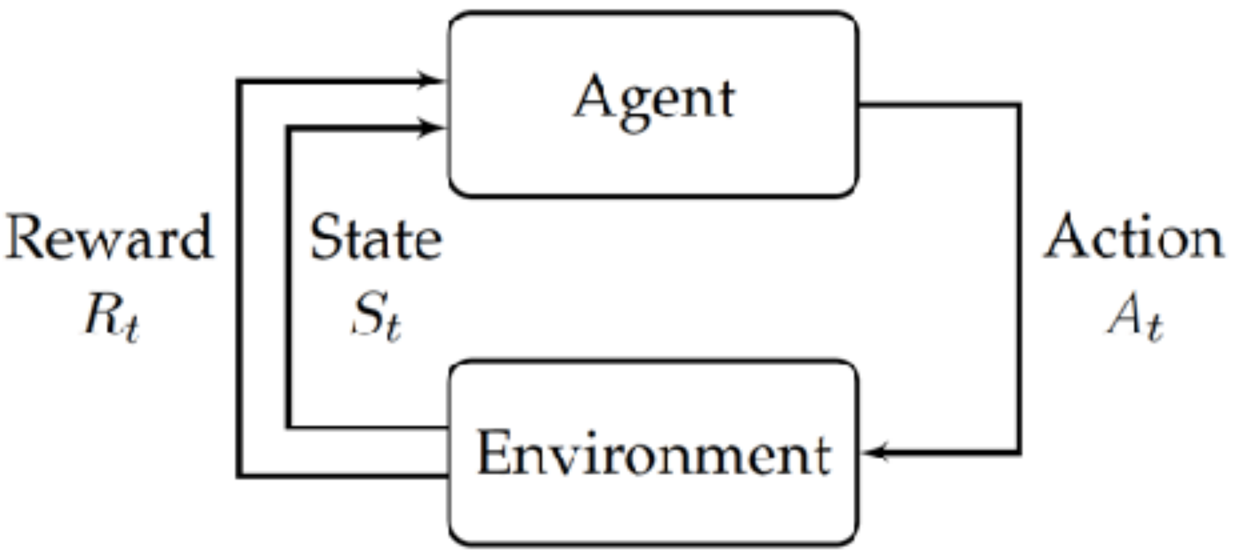
Agents and environments

Three Components:
- Environment
- Agent
- Time

Agent can observe environment

Agent can act upon environment

Environment emits reward



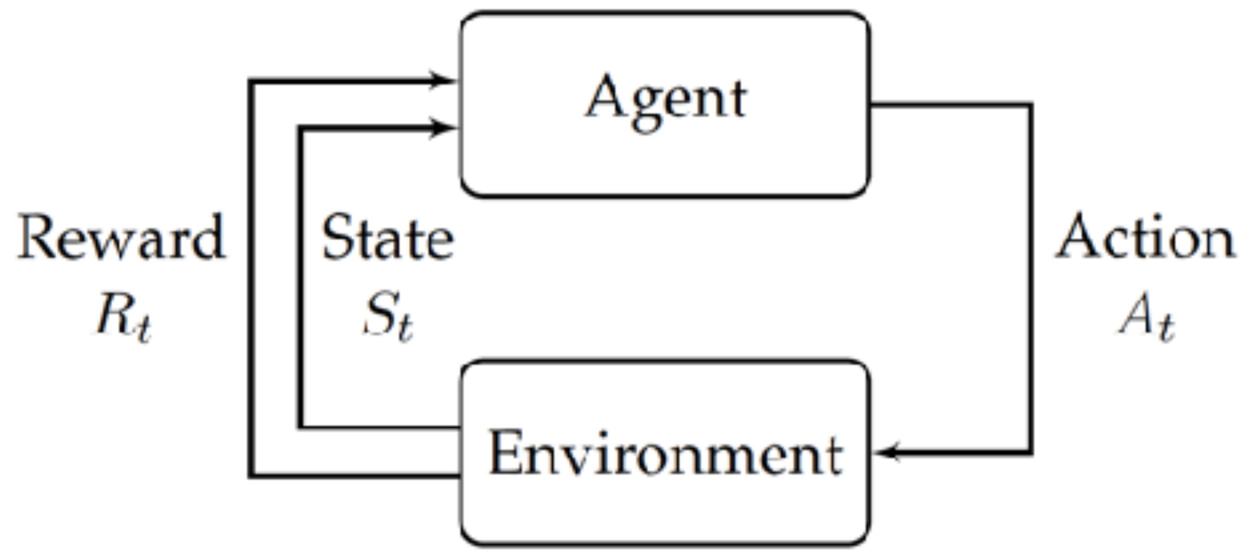
What is Reinforcement Learning?

Agents and environments

Agent can then create a policy that maps from states to actions

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

In the beginning it is random but by learning it approximates the optimal policy



What is Reinforcement Learning?

How does it learn?

What is Reinforcement Learning?

How does it learn?

Sample transitions from the environment

What is Reinforcement Learning?

How does it learn?

Sample transitions from the environment

Pick actions that return high reward

What is Reinforcement Learning?

How does it learn?

Sample transitions from the environment

Pick actions that return high reward

Goal is to always maximise sum of rewards

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Exploration vs Exploitation

What is the difference?

Exploration vs Exploitation

What is the difference?

Exploitation:

- taking the action that is thought to yield the highest reward

Exploration vs Exploitation

What is the difference?

Exploitation:

- taking the action that is thought to yield the highest reward

Exploration:

- taking a new (often random) action

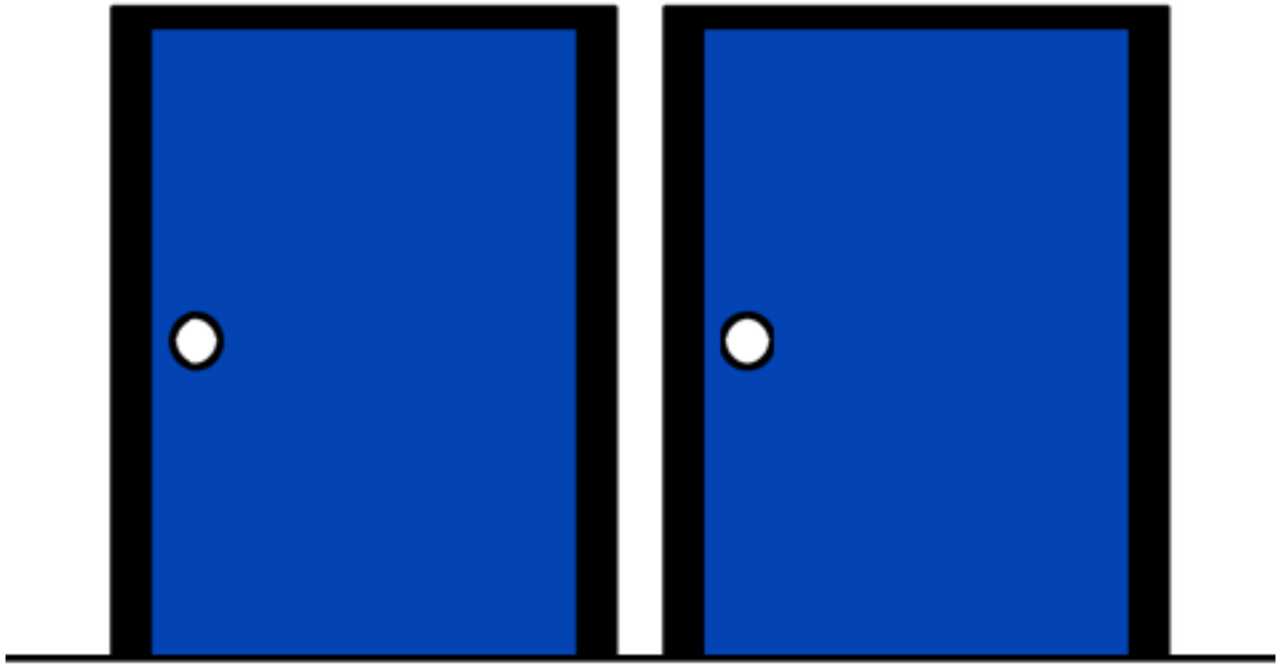
Exploration vs Exploitation

Why not always exploit?

Exploration vs Exploitation

Why not always exploit?

An agent has the choice to open the left or the right door

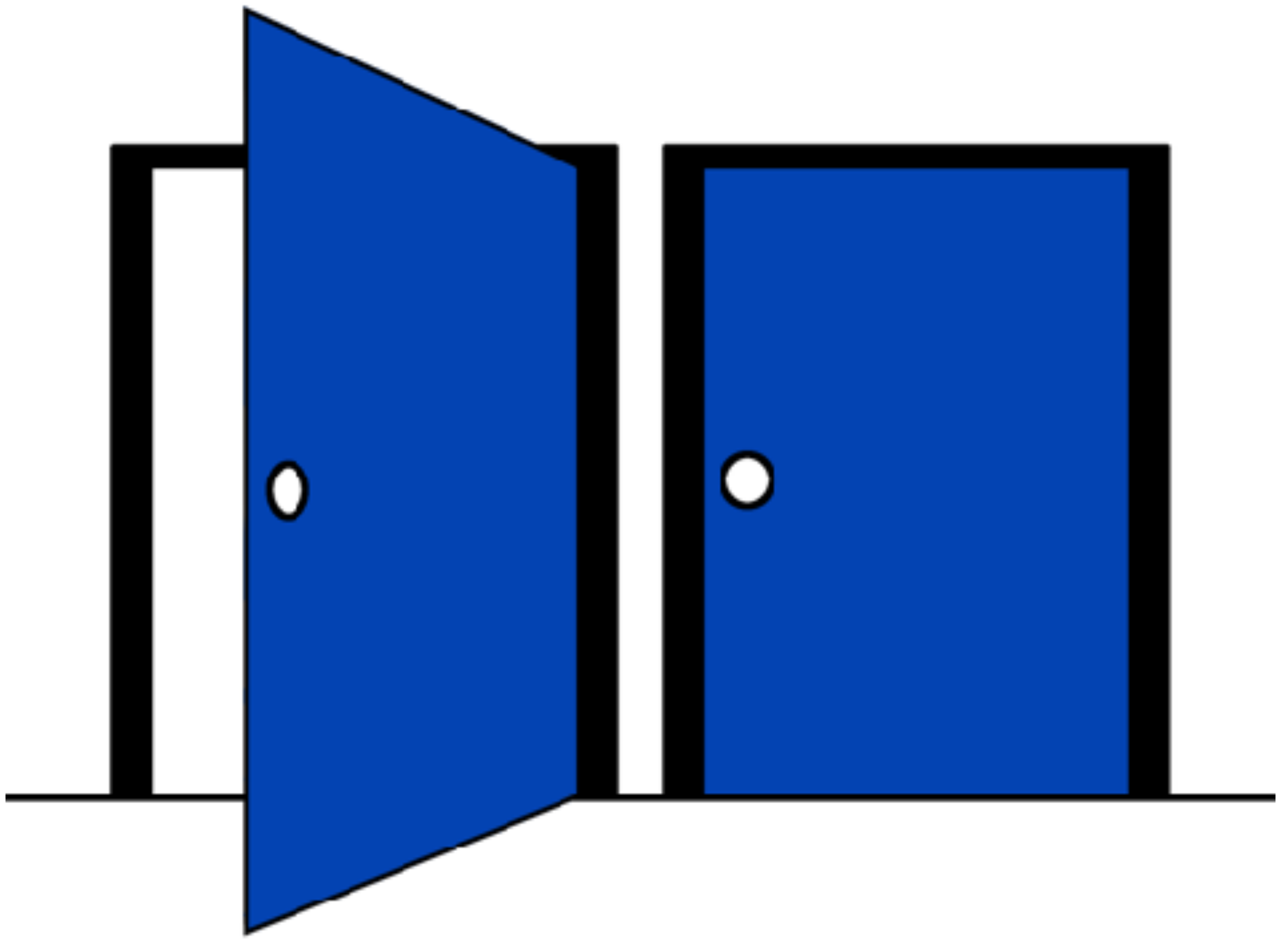


Exploration vs Exploitation

Why not always exploit?

An agent has the choice to open the left or the right door

When opening the left door he receives -5 reward



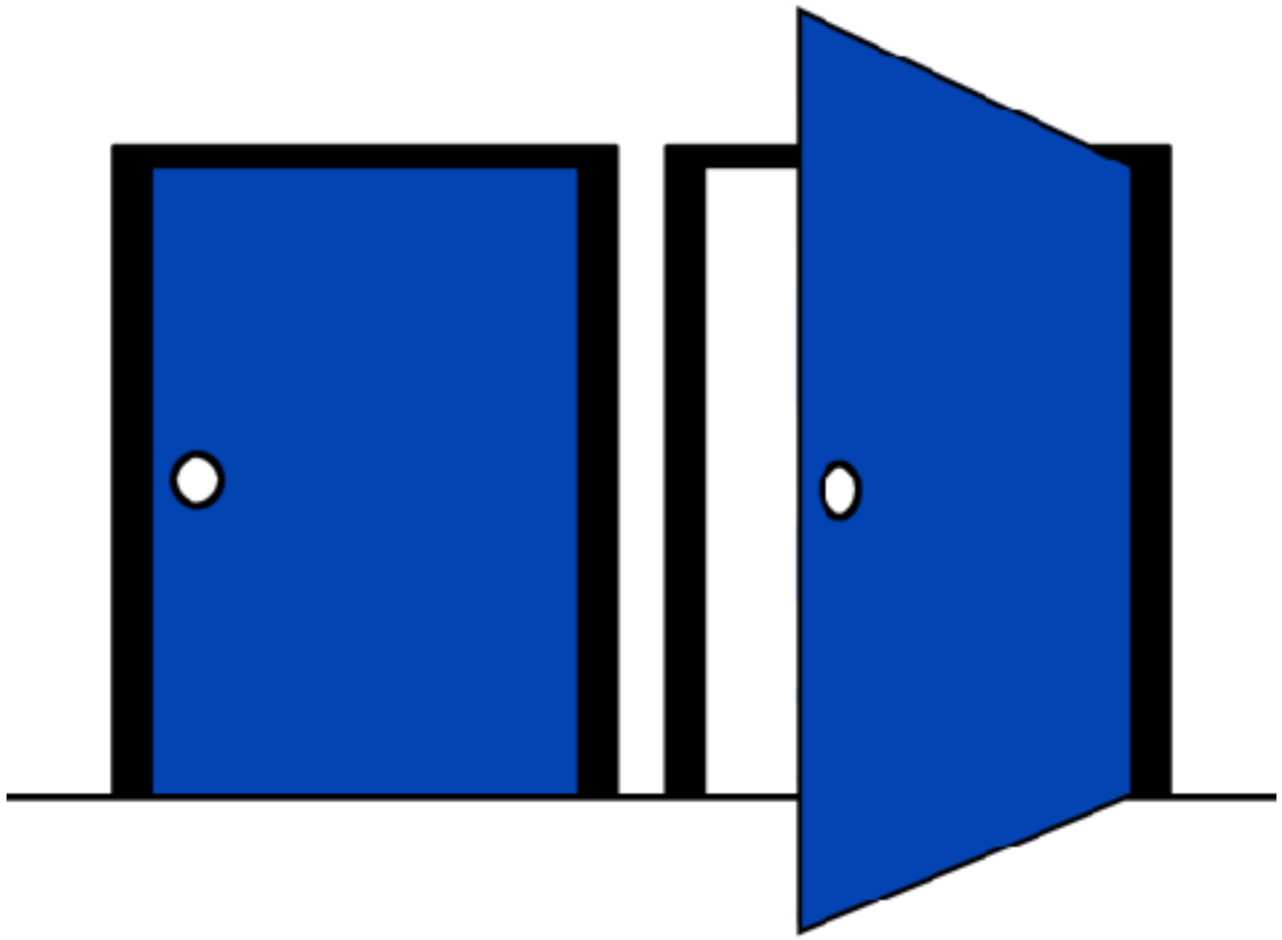
Exploration vs Exploitation

Why not always exploit?

An agent has the choice to open the left or the right door

When opening the left door he receives -5 reward

When opening the right door he receives +5 reward



Exploration vs Exploitation

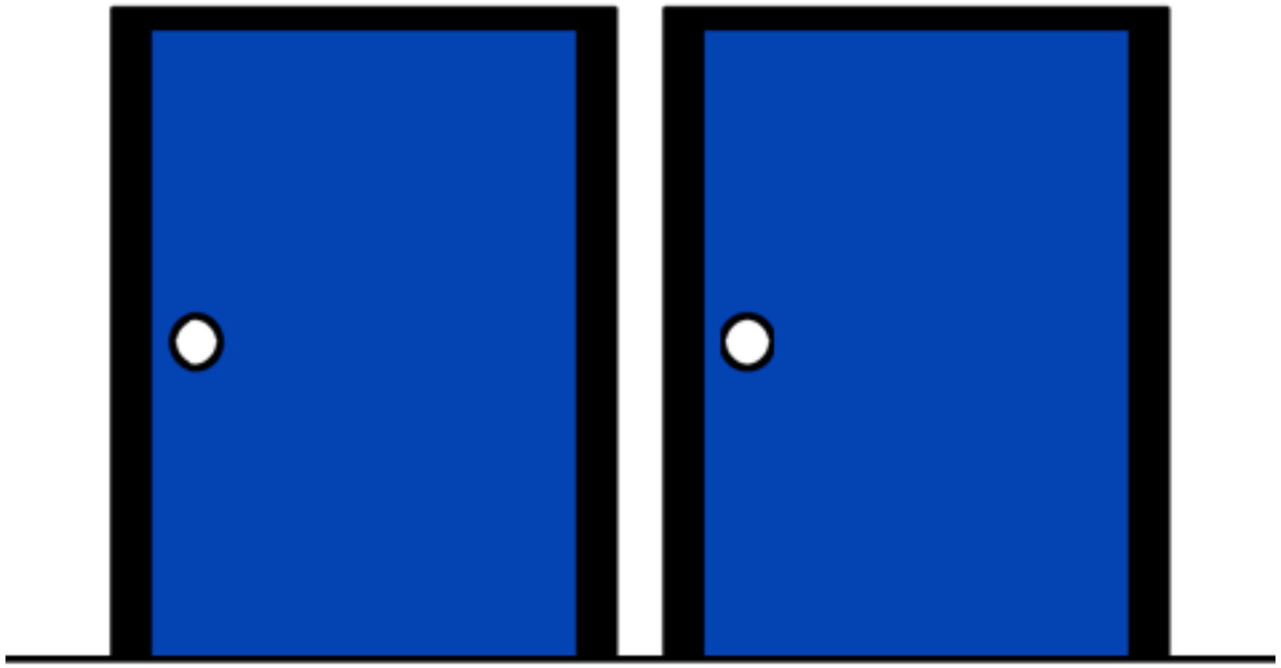
Why not always exploit?

An agent has the choice to open the left or the right door

When opening the left door he receives -5 reward

When opening the right door he receives +5 reward

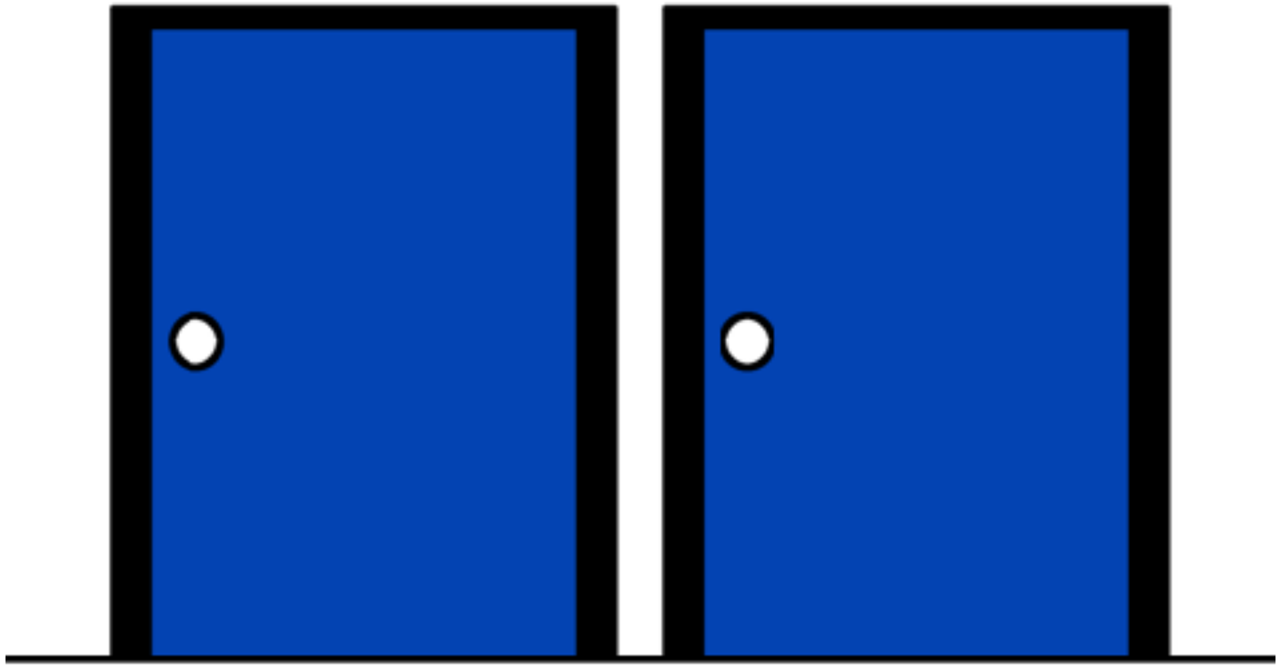
Should the agent always open the right door?



Exploration vs Exploitation

Why not always exploit?

Answer: No!

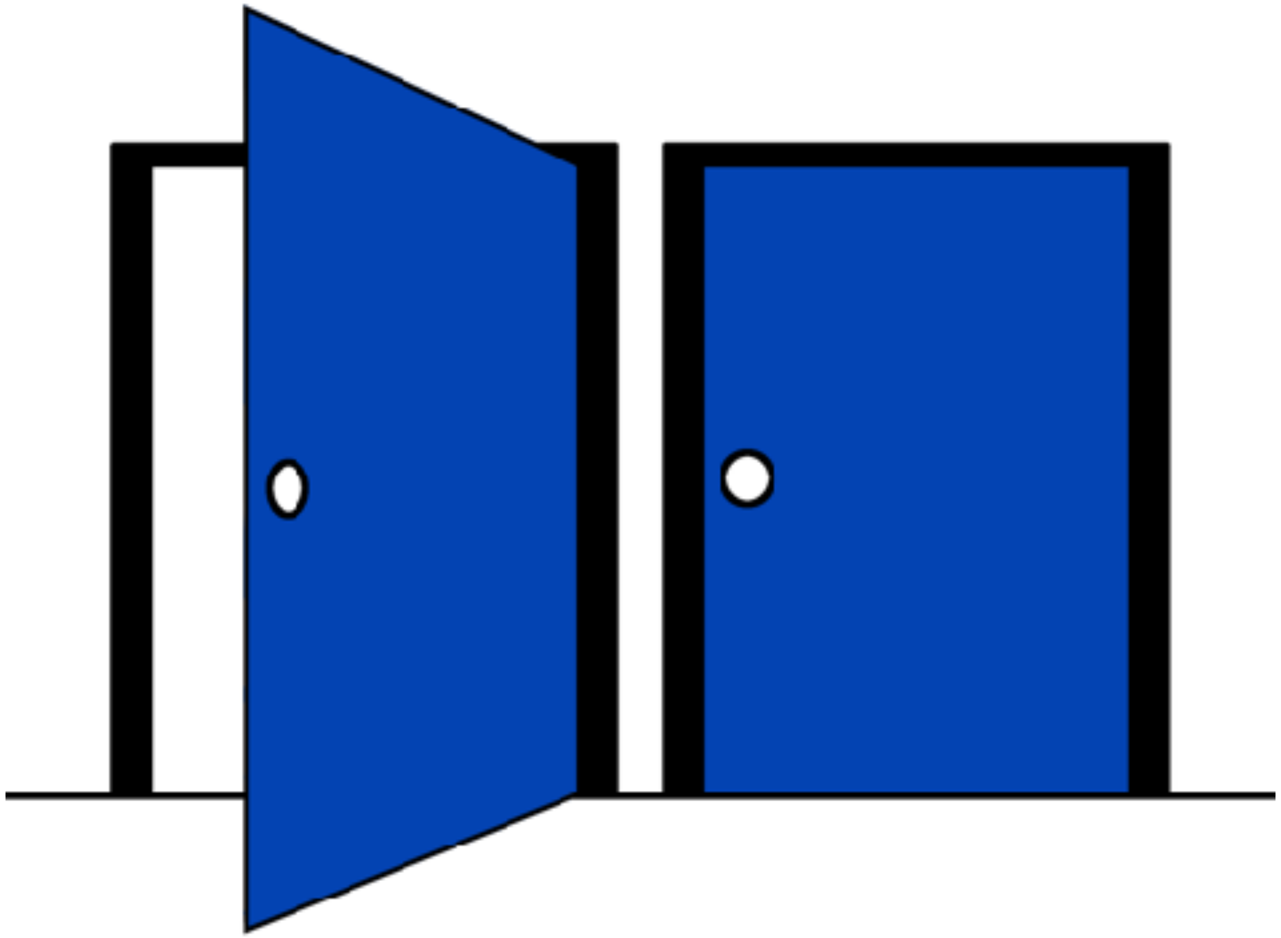


Exploration vs Exploitation

Why not always exploit?

Answer: No!

Maybe next time he opens the left he receives +10 reward



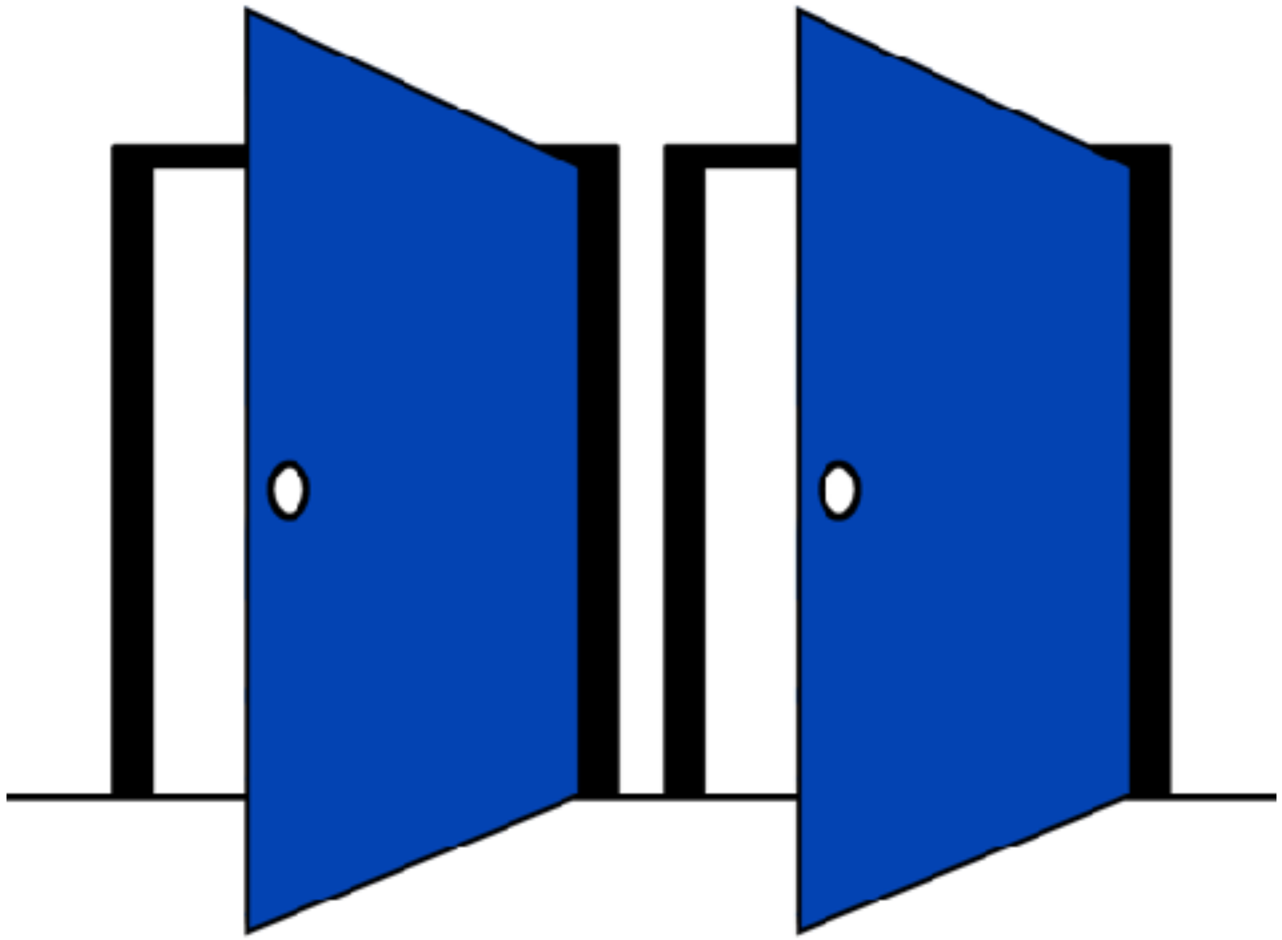
Exploration vs Exploitation

Why not always exploit?

Answer: No!

Maybe next time he opens the left he receives +10 reward

Solution is to sometimes explore and sometimes exploit



Different ways to explore

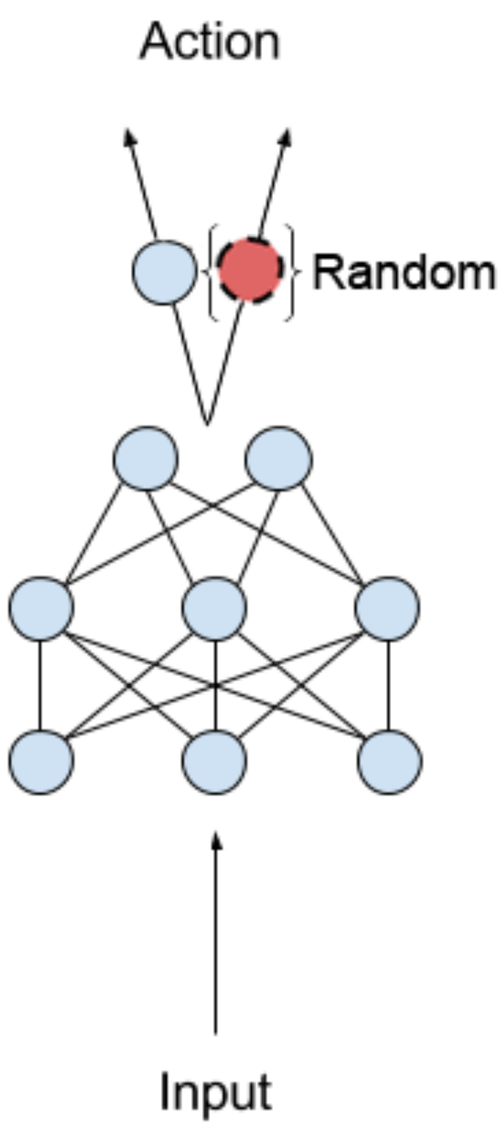
Many different ways to achieve the same goal

Different ways to explore

ε-Greedy Policy

With probability of ε choose a random action, otherwise select the best action

Decay ε after a while to exploit more



Different ways to explore

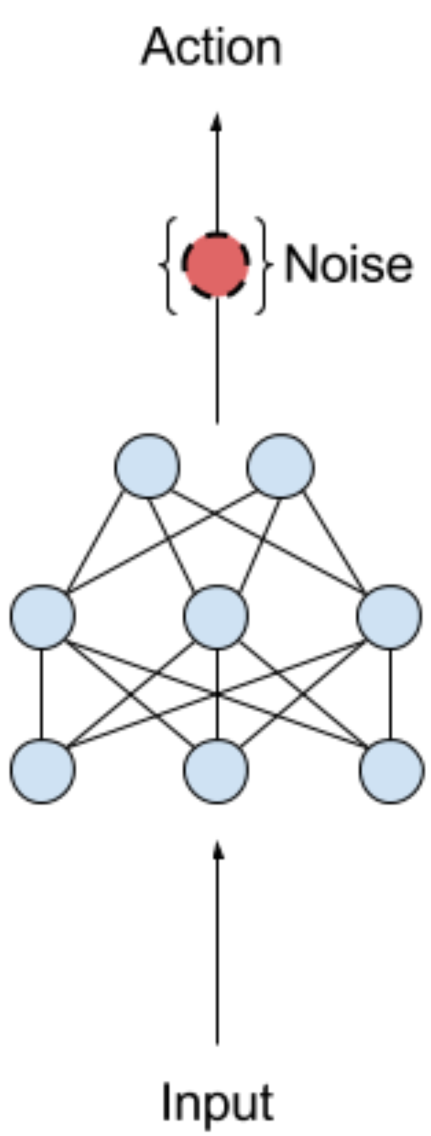
Action noise

In every timestep add a noise factor to the action

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$$

Noise terms can be correlated

$$x_{t+1} = x_t + dx_t$$



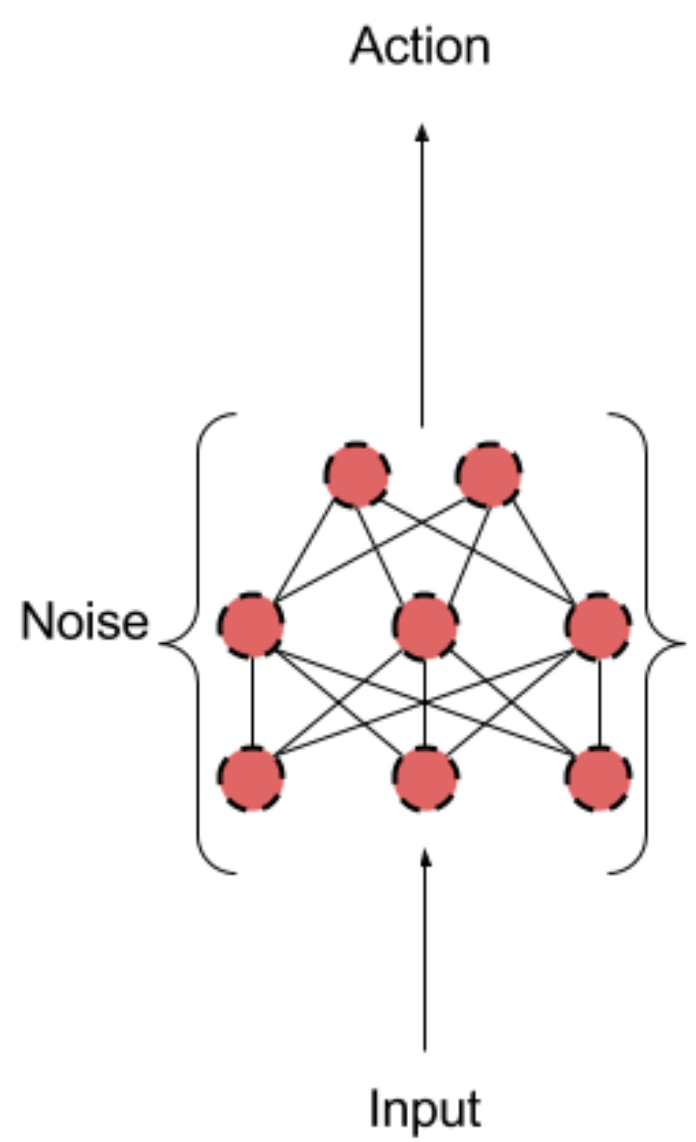
Different ways to explore

Parameter Noise

Instead of adding noise to the action add it directly to the network's parameters

$$\widetilde{\theta} = \theta + \mathcal{N}(0, \sigma^2 I)$$

Computationally Expensive, therefore compute noise only once per episode

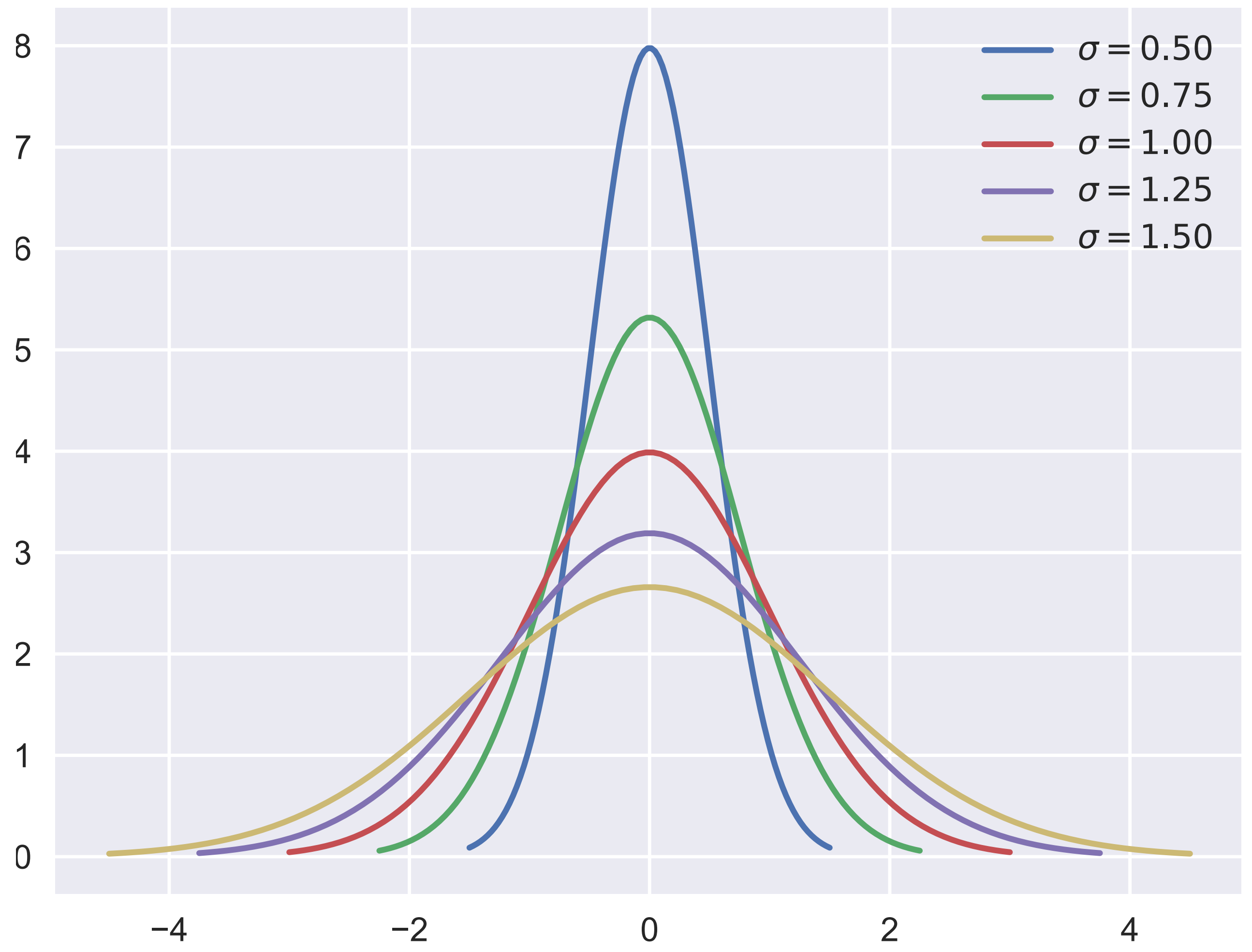


Different ways to explore

Standard deviation noise

Change the distribution of the policy to increase or decrease exploration

Higher standard deviation means more exploration



Implementation

Three very different algorithms:

- Deep-Q-Network
- Deep Deterministic Policy Gradient
- Proximal Policy Optimization

Tensorflow for deep neural networks

OpenAI for simulations (especially MuJoCo)

Results

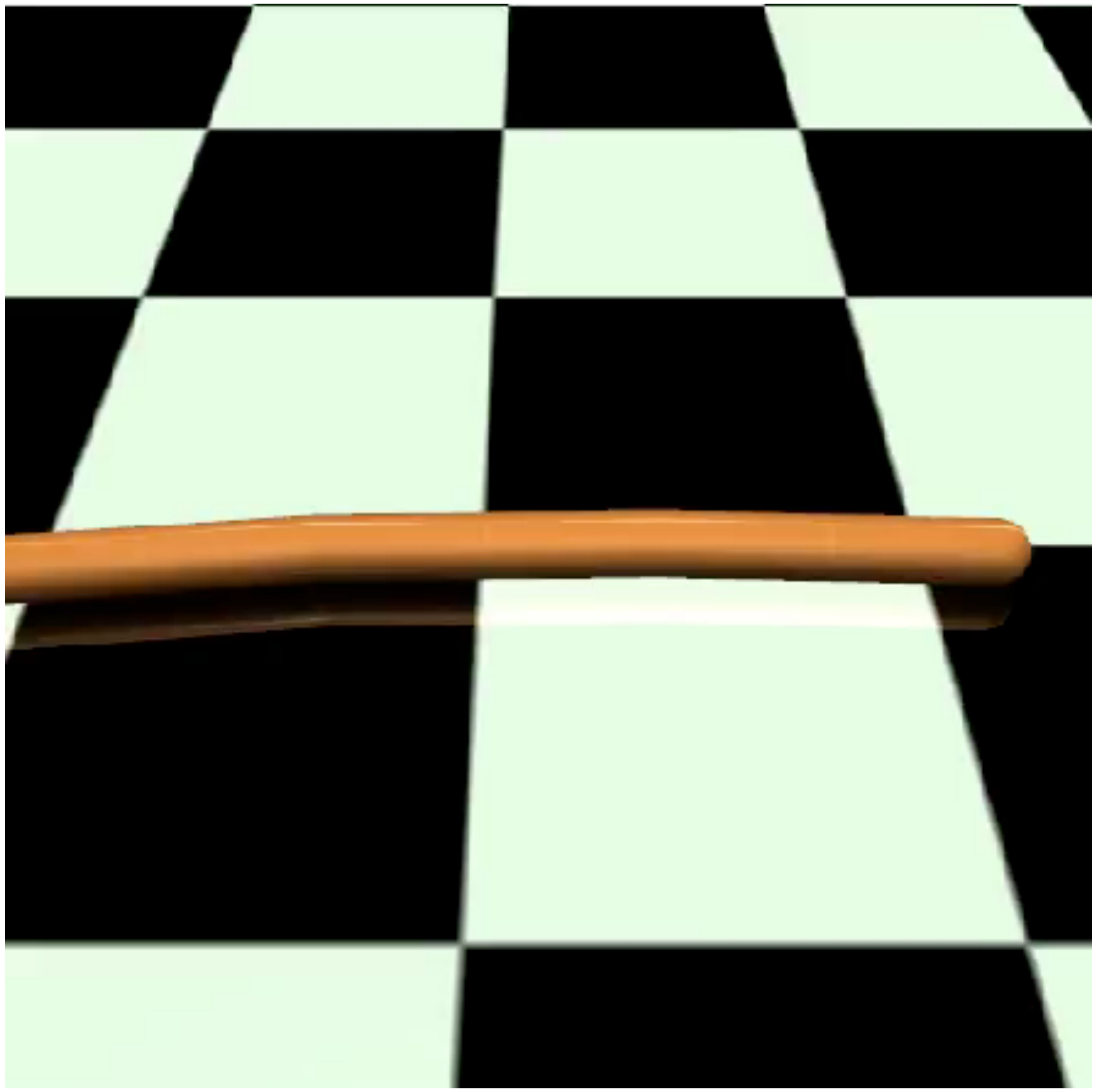
How did the algorithms perform?

Results

Deep Deterministic Policy Gradient with action noise

Did not learn what we would consider snake-like movement

Got stuck in a local maximum

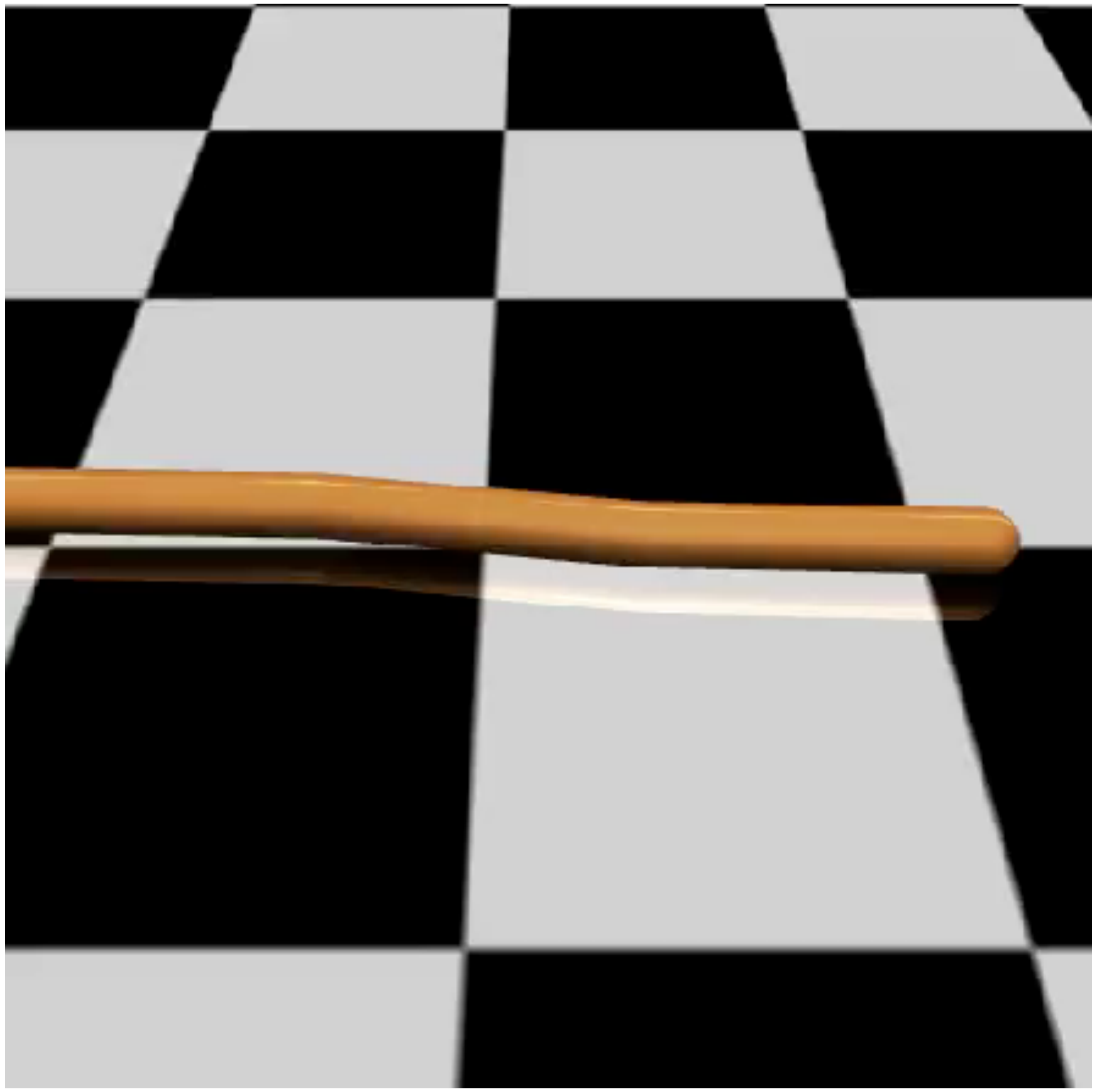


Results

Deep Deterministic Policy Gradient with parameter noise

Could not reproduce the results from the original paper

Did not learn a useful policy at all



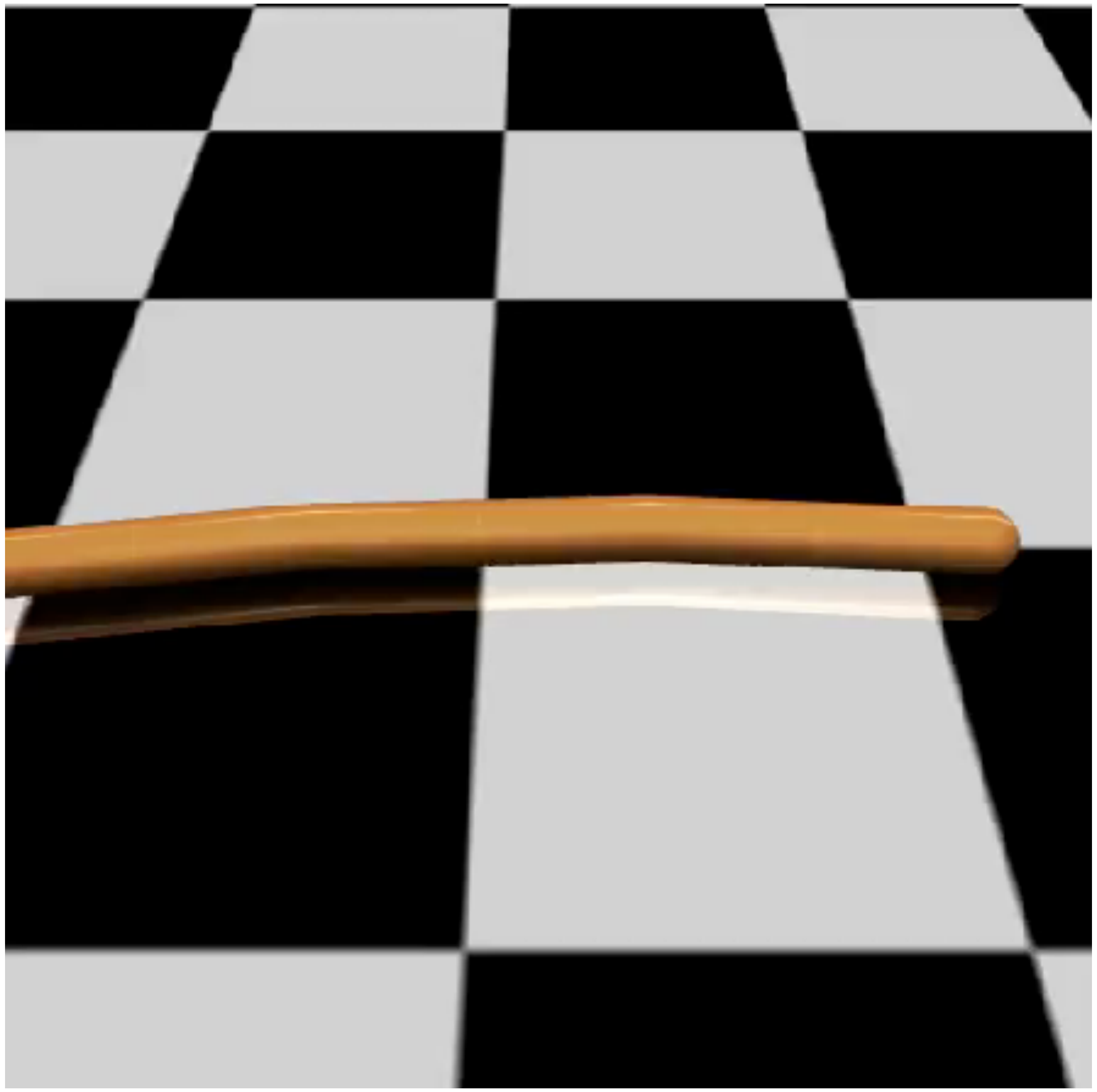
Results

Proximal Policy Optimization

Good results

Least computationally expensive

But high variance between different runs

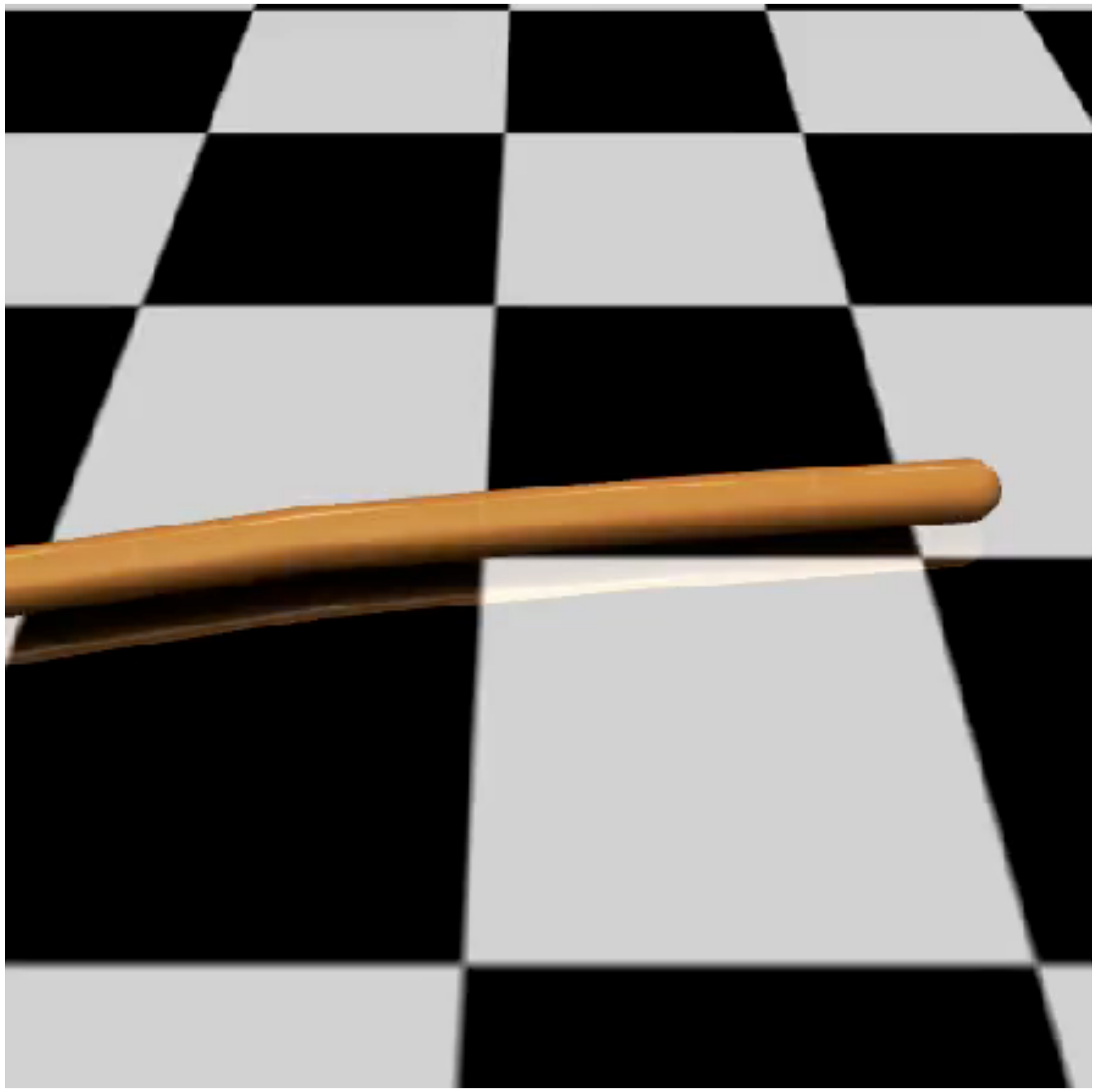


Results

Proximal Policy Optimization with new method for exploration

Even better results

Still high variance



Results

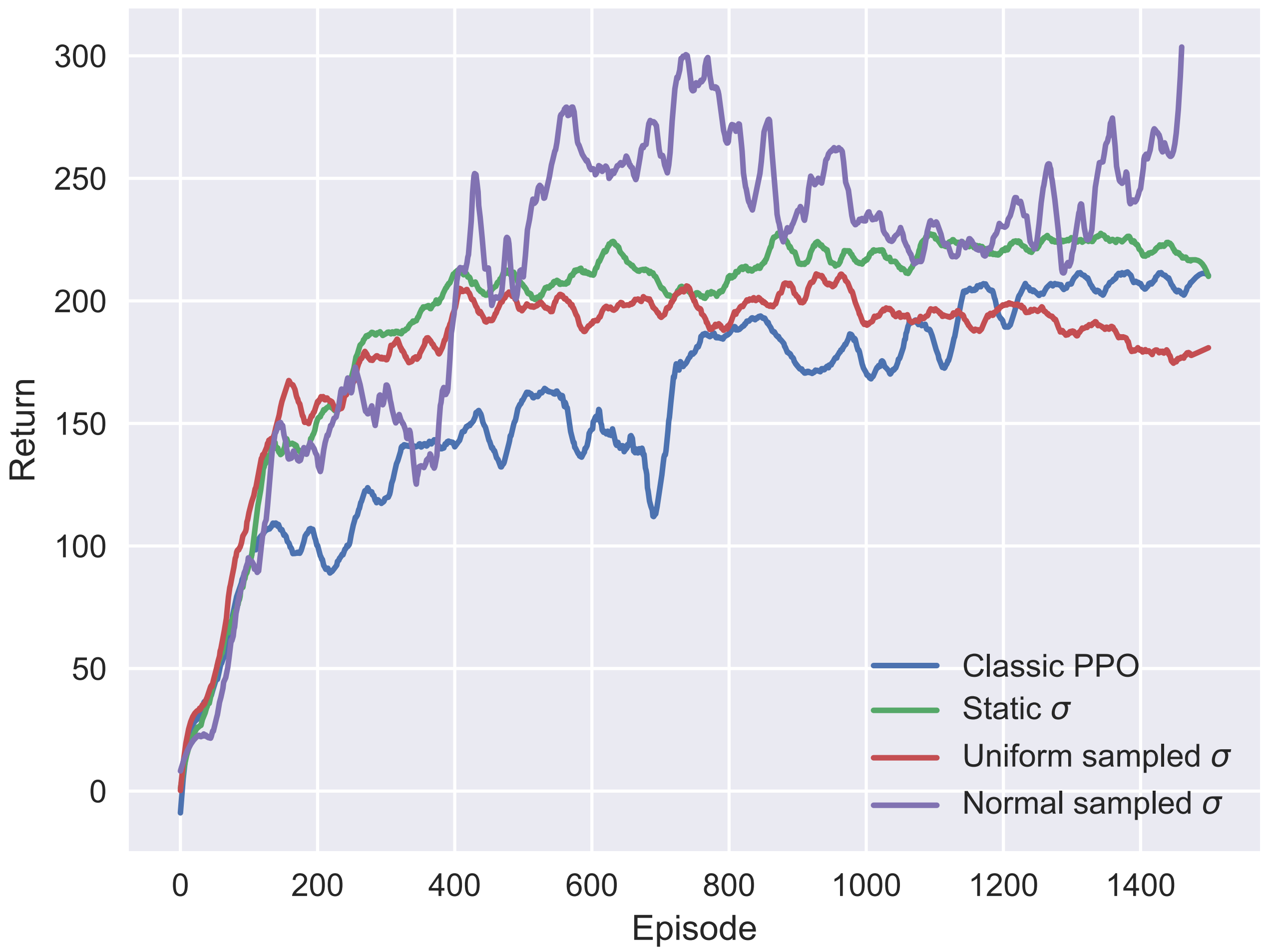
Comparing the different algorithms and noise factors

PPO outperforms DDPG every run

Results

Comparing the different algorithms and noise factors

Sampling the standard deviation from a normal distribution often made significant difference to the early learning curve



Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Algorithms still have flaws:

Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Algorithms still have flaws:

- only act in discrete space

Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Algorithms still have flaws:

- only act in discrete space
- learning a local maximum rather than a global one

Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Algorithms still have flaws:

- only act in discrete space
- learning a local maximum rather than a global one
- high variance between different runs

Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Algorithms still have flaws:

- only act in discrete space
- learning a local maximum rather than a global one
- high variance between different runs

The high variance of PPO made comparing new features difficult

Conclusion & Future Work

RL can learn difficult tasks even in high dimensional space

Algorithms still have flaws:

- only act in discrete space
- learning a local maximum rather than a global one
- high variance between different runs

The high variance of PPO made comparing new features difficult

Decreasing it should be of high priority for future work

Thank you!

Questions?