

CMPT 888: Computer Animation

# Comparison of Deep Reinforcement Learning Algorithms for Learning Control Policies for Physics-based Locomotion Tasks

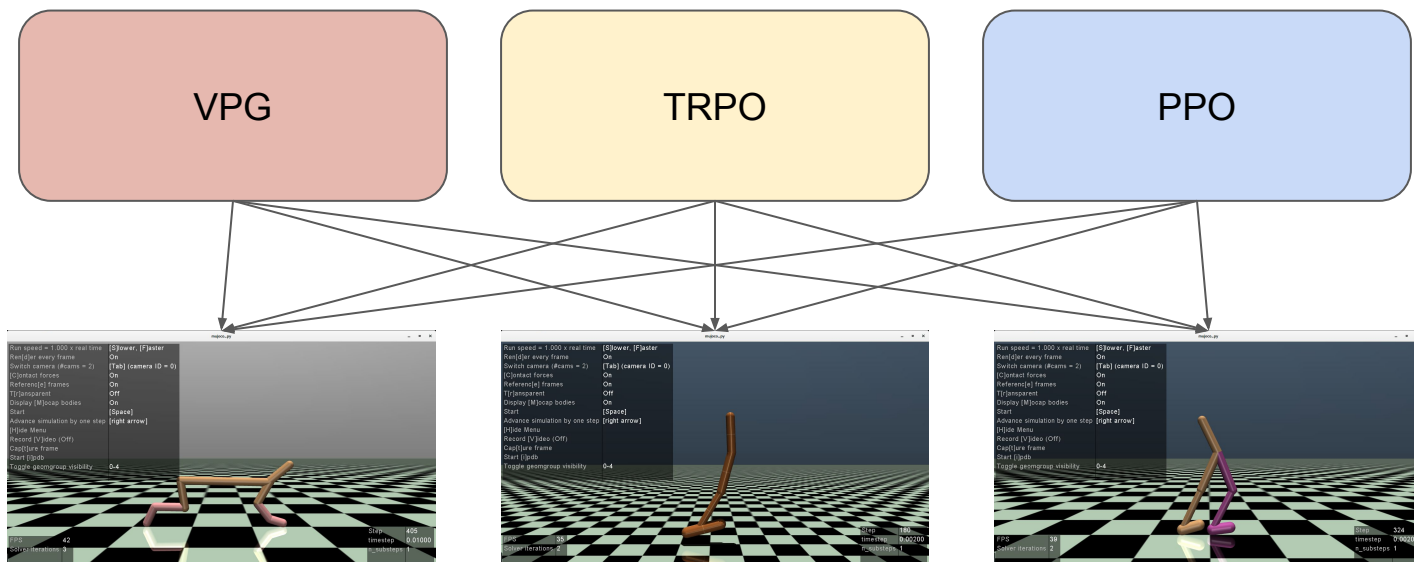
Project By:

**Anmol Sharma**

Medical Image Analysis Lab  
School of Computing Science  
Simon Fraser University

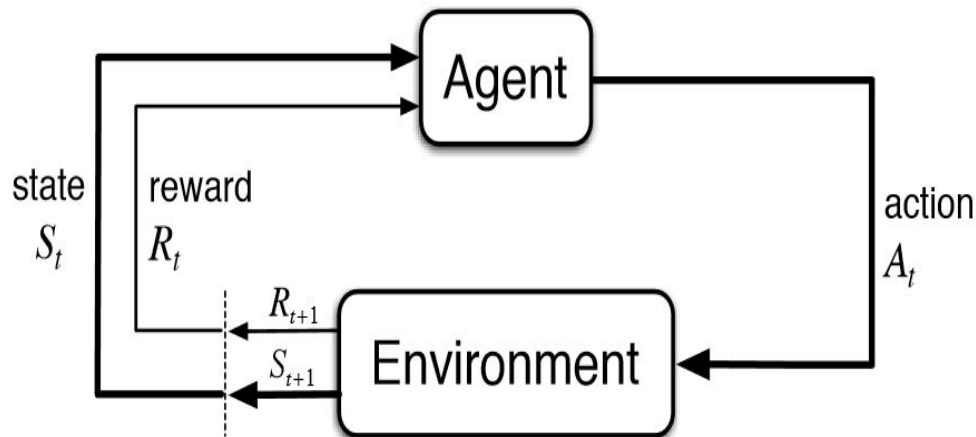
# Problem Statement

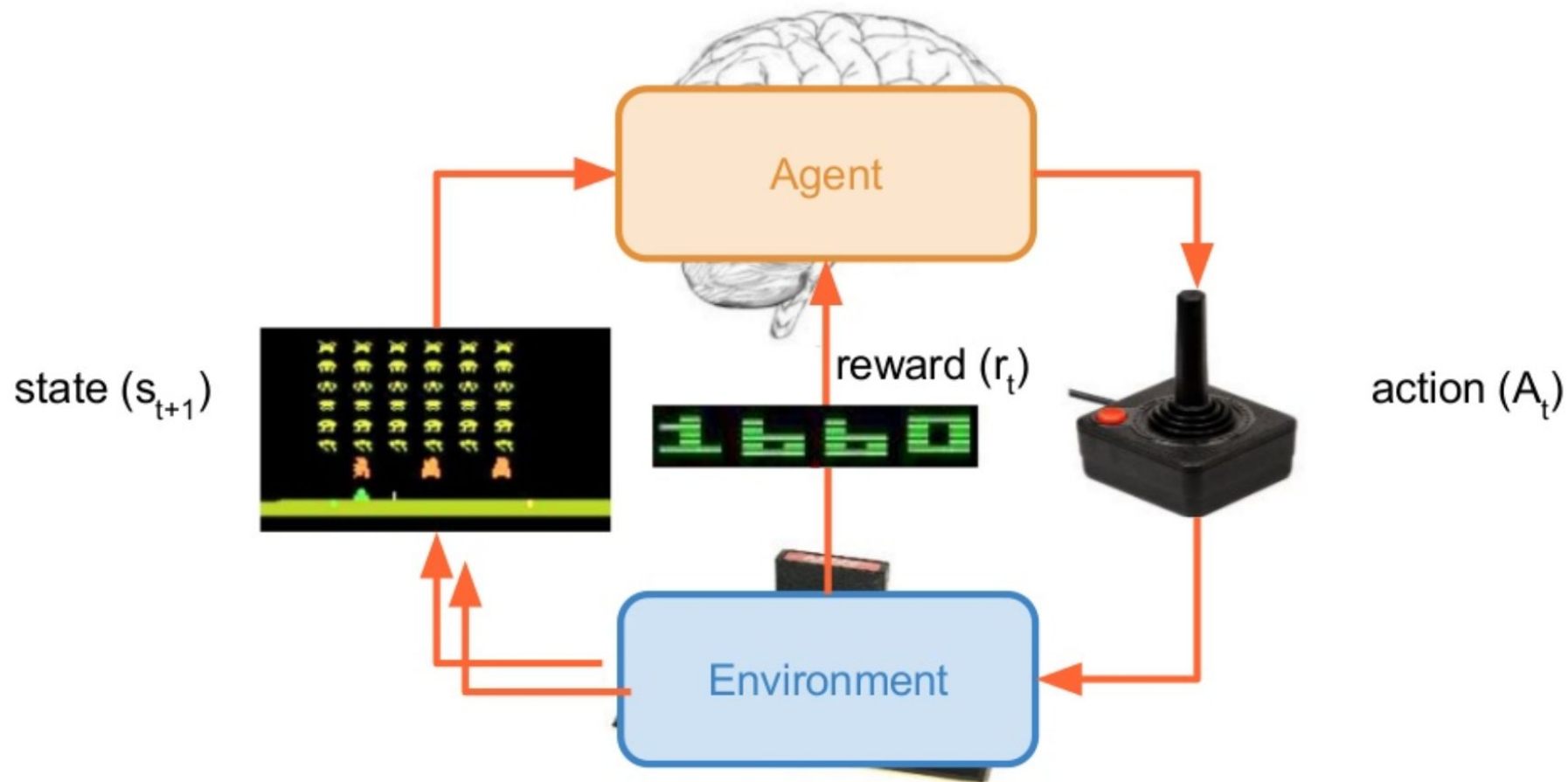
## Compare Deep Reinforcement Learning Algorithms



# Reinforcement Learning

- A class of machine learning algorithms that learn by trial and error.
- An “agent” interacts with the environment by performing an action, and in turn gets rewards.
- The agent learns from positive/negative rewards to perform better.

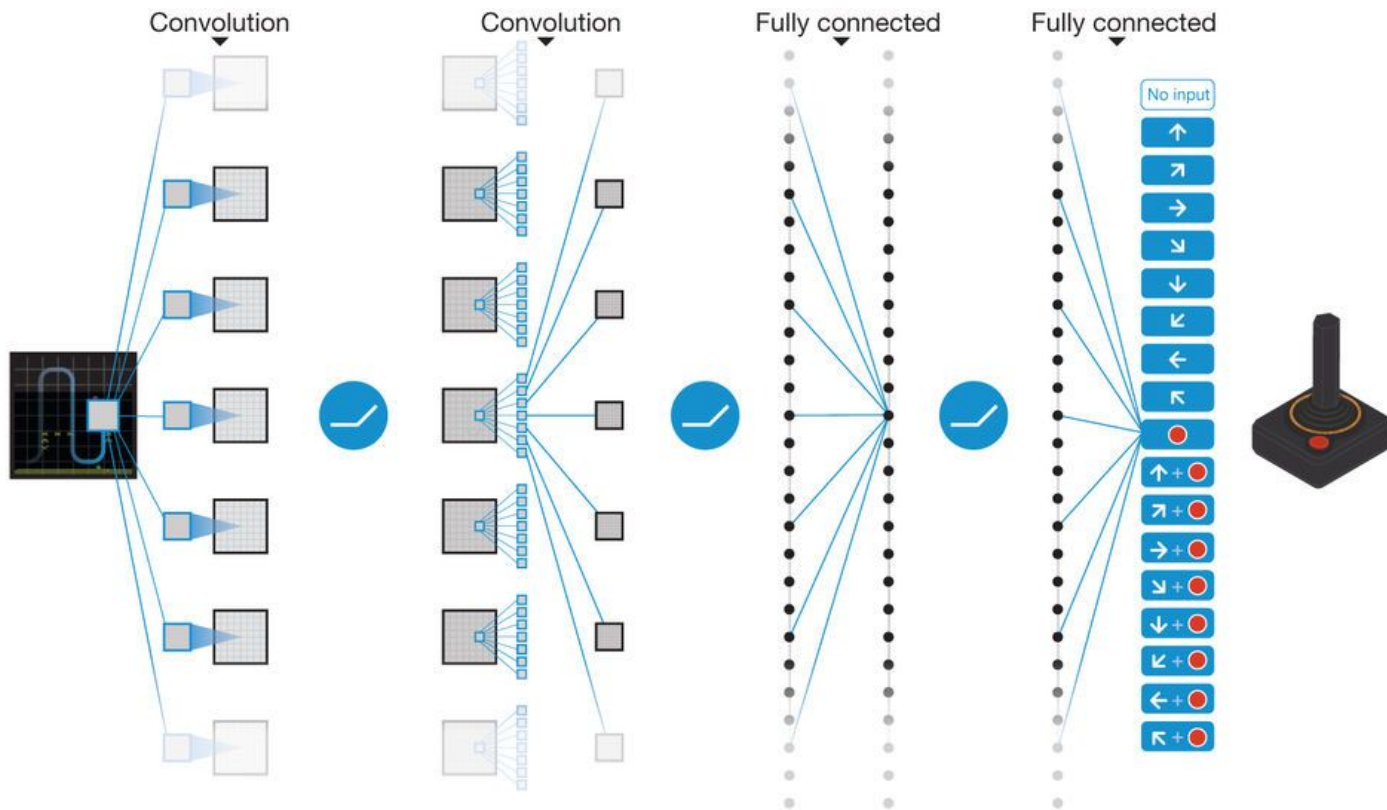




# Deep Q-Learning

- Recently there has been a surge in Deep Learning research.
  - Deep Learning showed groundbreaking results in many different fields.
- Google DeepMind in 2013 published a paper where they proposed a new variant of Q-Learning algorithm.
- It was called the Deep Q-Learning algorithm.
- Instead of having a Q-Matrix representation where states and actions were discrete and small.
  - Have a convolutional neural network represent the Q-function.
  - Takes input as state of the game, and action taken by agent.
  - Outputs the expected Q-value or reward by the end of episode that the agent will get.
  - Action with highest of this reward was chosen as agent's action. .
- Used the classic Bellman Equation to train the CNN using mean squared error.

# Deep Q-Learning



# Deep Q-Learning

- Deep Q-Learning showed great promise, and performed with almost superhuman capabilities, beating many human professionals in some of the Atari games.
- However this approach was still not applicable to domains where the action space is continuous.
  - Recall Atari games have discrete actions (up, down, left, right, jump, run, etc.).
  - However some tasks like robotic control continuous space.
- Hence there was a need for new algorithms which can generalize to continuous domains.

# Vanilla Policy Gradients (VPG) / REINFORCE

- One such algorithm that works on continuous action-spaces was the REINFORCE algorithm.
- This was first published in 1992, proposed by Ronald J. Williams.
- Simple policy based method.
  - Recall policy = Function that takes state as input and outputs the required action to take.
- In current implementation, VPG is implemented with the policy represented as a Neural Network.
- The policy is updated using the gradients (typically through backpropagation).
- Very simple method, easy to implement with good convergence properties.



# Trust Region Policy Optimization (TRPO)

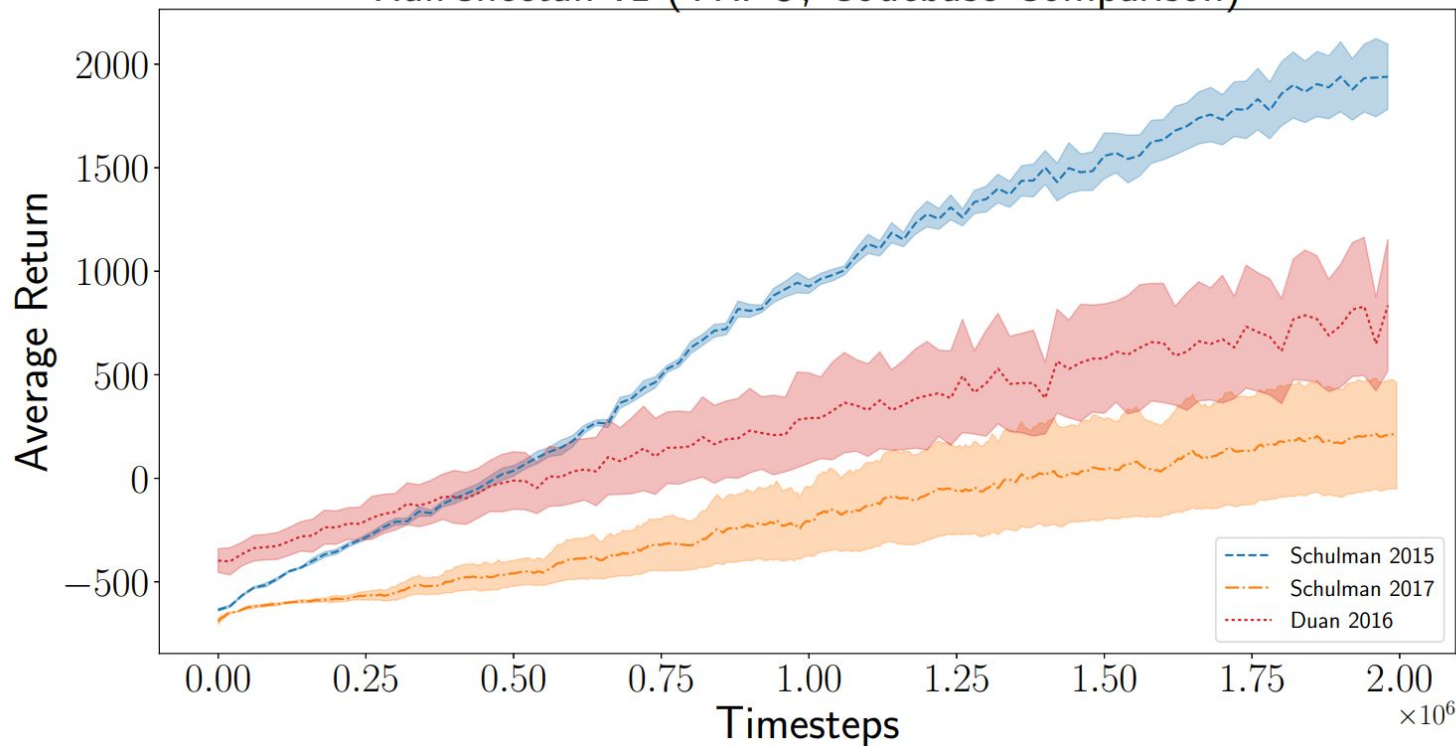
- Trust Region Policy Optimization (TRPO) is another policy-based method, where a policy function is updated according to a set of constraints.
- TRPO proposes to strongly constraint policy updates in a way that there is always an improvement.
  - That is, the agent performs in a monotonically increasing manner (in terms of reward value).
- For this, they apply a strong Kullback-Leibler divergence constraint to the optimization function:

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{\text{KL}}^{\max}(\pi, \tilde{\pi}),$$

$$\text{where } C = \frac{4\epsilon\gamma}{(1 - \gamma)^2}.$$

# Trust Region Policy Optimization (TRPO)

HalfCheetah-v1 (TRPO, Codebase Comparison)



# Proximal Policy Optimization (PPO)

- Proximal Policy Optimization (PPO) algorithm was a follow-up work by the same authors of TRPO.
- Although TRPO provides very nice guarantees of convergence, and monotonically increasing agent performance, it's incredibly hard to optimize and implement.
- PPO tries to address this problem by relaxing some of the strong constraints in TRPO, while still trying to maintain the nice convergence properties.
- They introduced a simpler constraint method, by clipping the reward function.

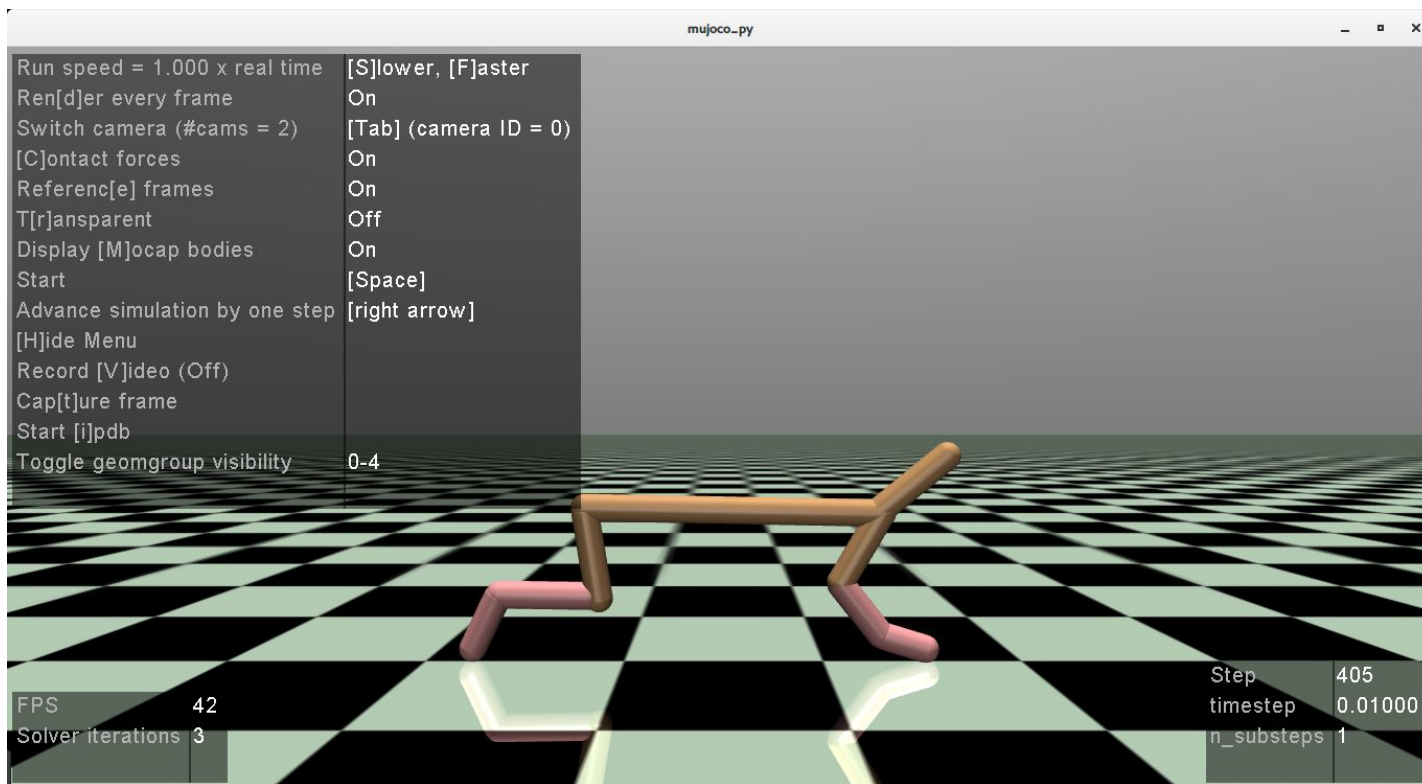
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right]$$

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[ r_t(\theta) \hat{A}_t \right].$$

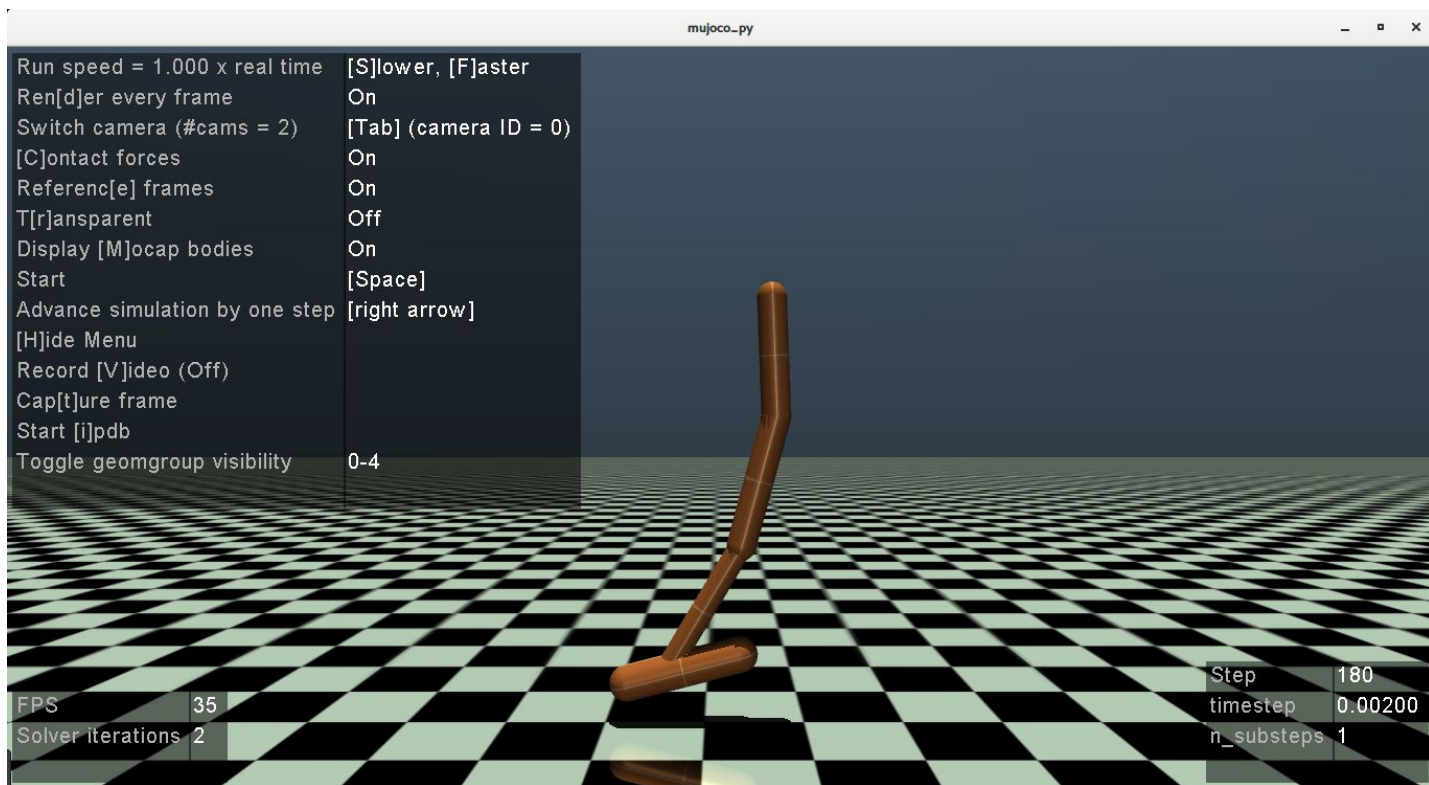
# MuJoCo: A Physics Engine

- MuJoCo is a physics-based engine which is used for simulating entities requiring physical control mechanisms.
- The software is closed source and not free, however student license is available.
- Provides many pre-defined environments of various robots with different controllable joints, and varying complexities.
- In this project, we use three environments to provide a dynamic range of complexity to our agents:
  - HalfCheetah-v2
  - Hopper-v2
  - Walker2d-v2

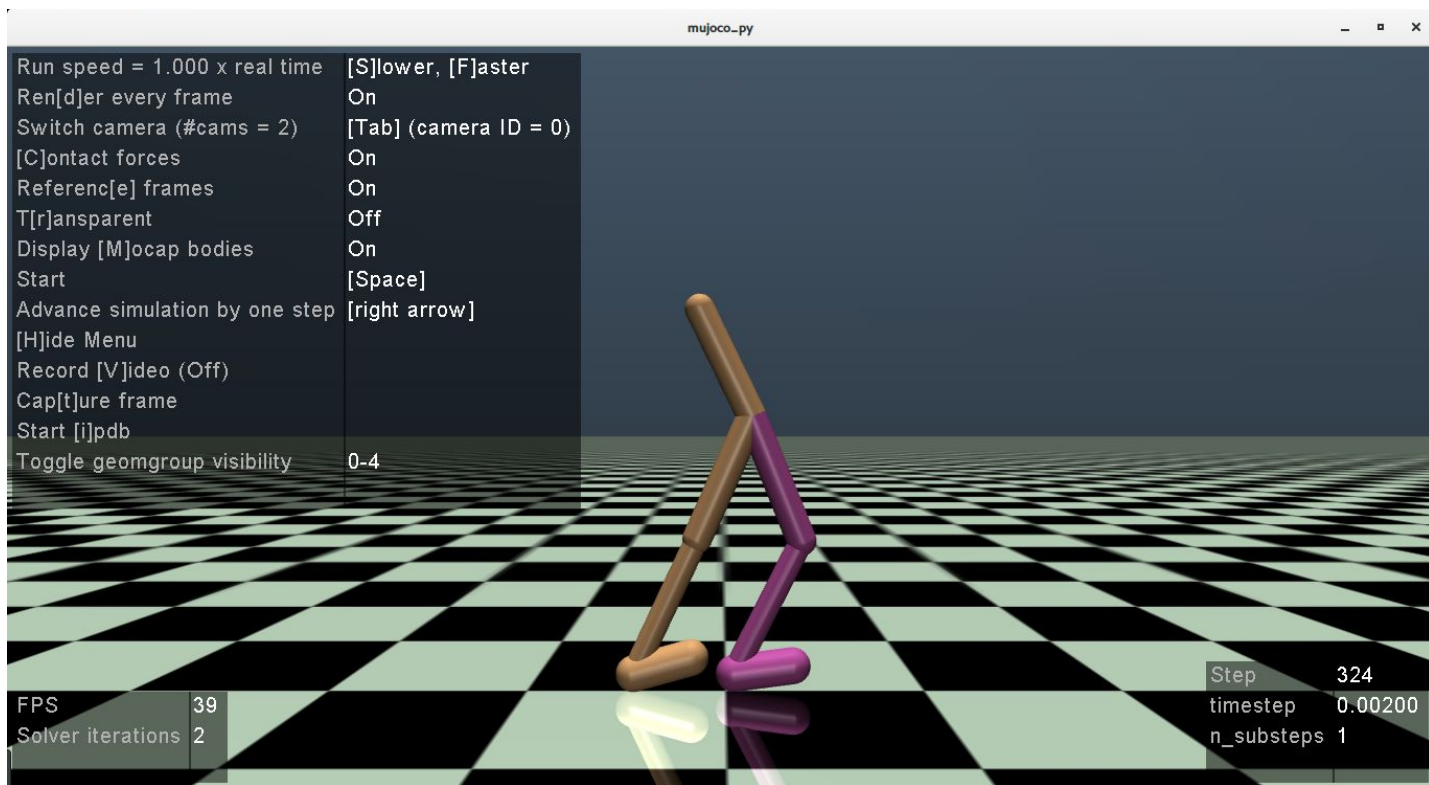
# MuJoCo: A Physics Engine



# MuJoCo: A Physics Engine



# MuJoCo: A Physics Engine



# OpenAI Gym

- OpenAI Gym is an open-source toolkit to experiment with Reinforcement Learning.
- Born out of a collaboration between Elon Musk and Sam Altman due to their reluctance against Reinforcement Learning, and AI in general.
- Tries to “democratize” AI in the sense that the best algorithms are provided as baselines.
- Standard test-bed environments are provided in the toolkit which allows fair comparison amongst different RL algorithms.
- Provides a thin wrapper over MuJoCo in the form of MuJoCo-py which allows easy manipulation of MuJoCo environments.





## Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

[View documentation >](#)

[View on GitHub >](#)

# TensorForce: An RL Toolkit

- For this project we use the TensorForce open-source toolkit, based upon TensorFlow framework.
- TensorForce allows building RL agents in the same way as TensorFlow allows building Neural Networks
  - Through stacking/defining layers on top of each other, while the backend (gradient calculation, error propagation) is handled by TensorFlow.
- The library has a JSON like interface to build agents.
- Hyperparameters for the agents were chosen to be as close to the original paper as they can be.
  - Due to the fact that some environments are harder, hence more complex networks or parameters may have to be defined.

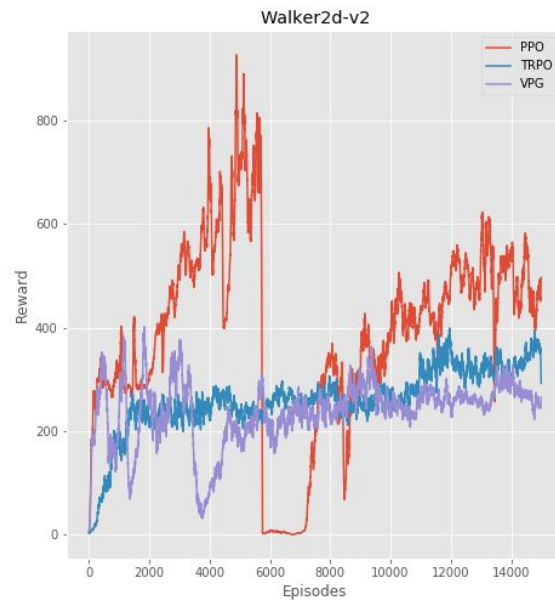
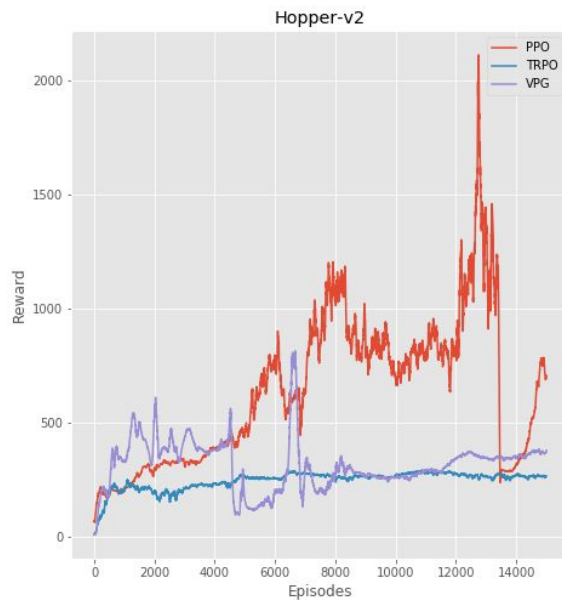
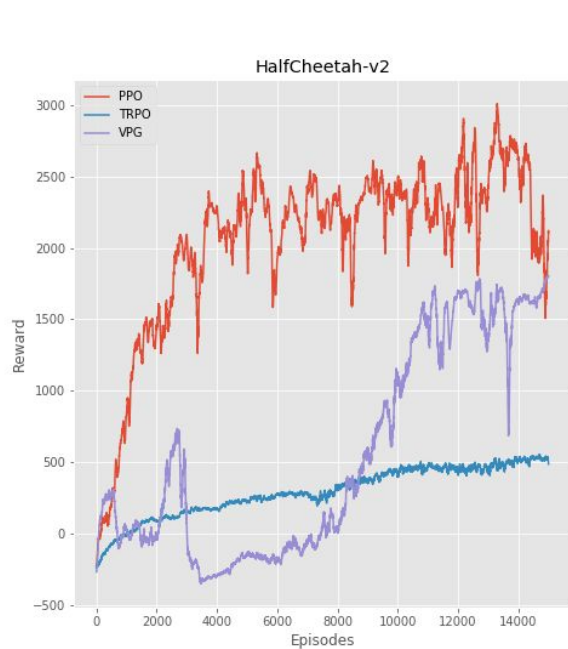
# Evaluation Metric

- For our tests, we use the absolute reward gained by each agent at the end of the episode as the basis for comparison.
- This is represented as:

$$\mathcal{R} = \sum_{t=0}^n r_t$$

- Some authors also use Q-Value as a metric, but some algorithms do not have a Q-function, hence it sometimes cannot be reported.
- Graphs which visualize the convergence behaviour of each agent, in each of the environment is provided.

# Results



# Results



Thank you!

# Reinforcement Learning: Q-Learning

- The field is decades old, and research has been going on solving many different problems through RL.
- One of the classic algorithms was Q-Learning.
  - Had a Q-Matrix as a representation to determine the expected reward when the agent takes an action  $\mathbf{a}$  in particular state  $\mathbf{s}$ .
- However the Q-Matrix representation in the algorithm couldn't scale to large state-action spaces.
  - Also, what about actions which are continuous? Like robotic control?
  - State space sometimes may come in the form of direct output from environment, which is very high dimensional.
- How to handle these complexities, and how to scale RL to larger, complex, highly dynamic systems with continuous/discrete control?


# Reinforcement Learning: Primer

- Let's quickly go through the main building blocks of an RL agent once more.
- There are a couple functions that are defined in any RL problem, which are:
  - Policy Function
  - Value Function
  - Reward Function
  - Q-Function
  - Actor/Critic
- We'll briefly describe these using illustrations.



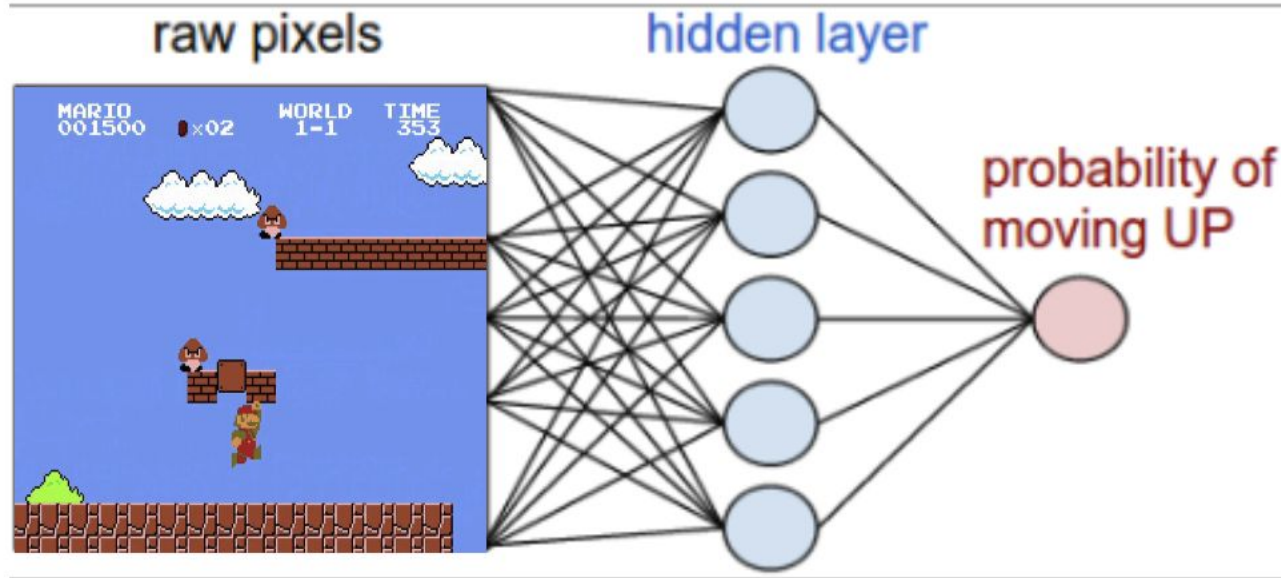
# Reinforcement Learning: Primer

## Policy Function

$$\pi \left( \text{Current State} \right) = \text{“Jump”}$$


# Reinforcement Learning: Primer

## Policy Function



# Reinforcement Learning: Primer

## Reward Function

$$\mathcal{R} \left( \begin{array}{c} \text{Current State} \\ \text{Current Action} \end{array} \right) = 100$$

The equation illustrates the reward function  $\mathcal{R}$  for a Super Mario Bros. game. The "Current State" is represented by a screenshot of the game, showing Mario at the bottom, a Goomba enemy on a brick platform, and a Koopa enemy on a higher platform. The "Current Action" is "Jump". The reward for this action is 100.

# Reinforcement Learning: Primer

## Value Function

$$V \left( \begin{array}{c} \text{Screenshot of Super Mario Bros. game state} \\ \text{Current State} \end{array}, \begin{array}{c} \text{"Jump"} \\ \text{Current Action} \end{array} \right) \approx 5000$$

Another variant of this function is called Q Function, which takes action-state pair and gives back expected return. 28