AI - 02 RL GAMES

OUR TEAM

Rishi

Gandhar

Neelavo

Aditi

Yash

Dinesh

Ashish

Project Leads : Shiva Surya, Ashmit

About the Project - RL Games

This project focuses on utilizing a variety of RL algorithms to tackle challenging games like Mario, Minecraft, and others. By leveraging these algorithms, our objective is to train intelligent agents capable of autonomously excelling in these games. Furthermore, we aim to push the boundaries of RL research by experimenting with novel algorithms in simpler game environments provided by OpenAI's Gym. This project strives to contribute to the advancement of RL techniques, enhancing their applicability and effectiveness in diverse gaming contexts.

Intro to RL

Reinforcement Learning is a subfield of machine learning that teaches an agent how to choose an action from its action space, within a particular environment, in order to maximize rewards over time.

Reinforcement Learning has five essential elements:

1. **Agent.** The program you train, with the aim of doing a job you specify.
2. **Environment.** The world, real or virtual, in which the agent performs actions.
3. **Action.** A move made by the agent, which causes a status change in the environment.
4. **Rewards.** The evaluation of an action, which can be positive or negative.
5. **State.** This refers to the state the environment is in. After the agent performs an action, it gets a reward and moves on to a different state.

Algorithm 1:Q-Learning

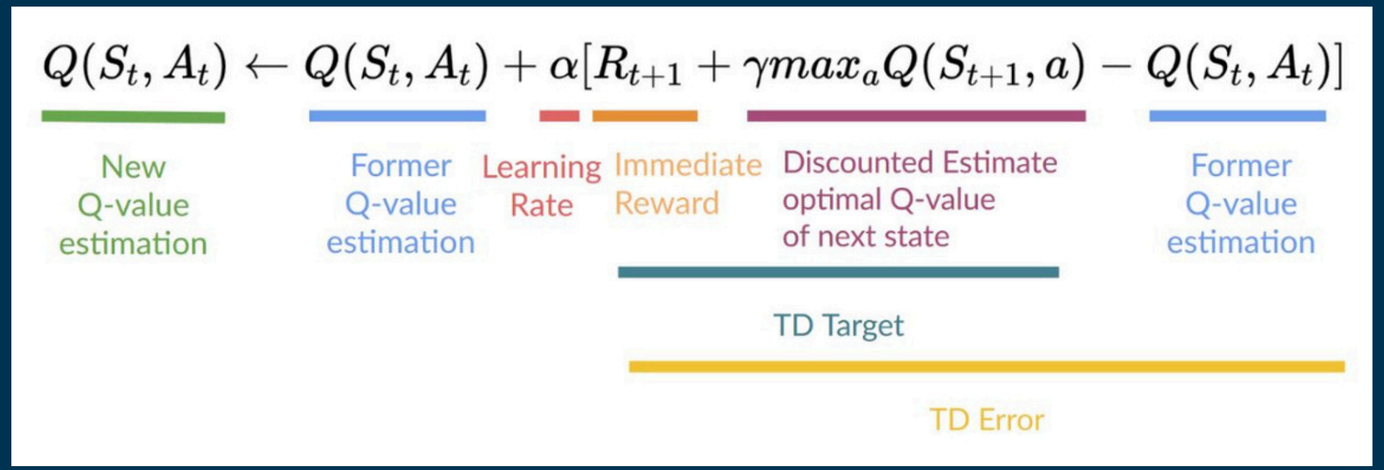
Q-Table and Q-Values:

• A Q-table stores Q-values representing the expected cumulative rewards for specific state-action pairs while following an ε-greedy policy, selecting a random action (exploration) and action with the highest Q-value (exploitation) with the probabilities ε and 1 – ε.

Computing the Q-Values:

• The Bellman equation is used to iteratively update Q-values, establishing a relationship between the current Q-value and those of subsequent state-action pairs.

• The update formula for the calculation of the Q-value is:



Advantages:

a) Simplicity.

b) Off-policy: Q-learning is an off-policy algorithm, which allows for more efficient exploration .

Disadvantages:

a) Sample inefficiency: It can require many samples to learn accurate Q-values, making it computationally expensive and time-consuming.

b) Lack of generalization in unseen states.

A Video Demo

Algorithm 2:Deep Q-Networks

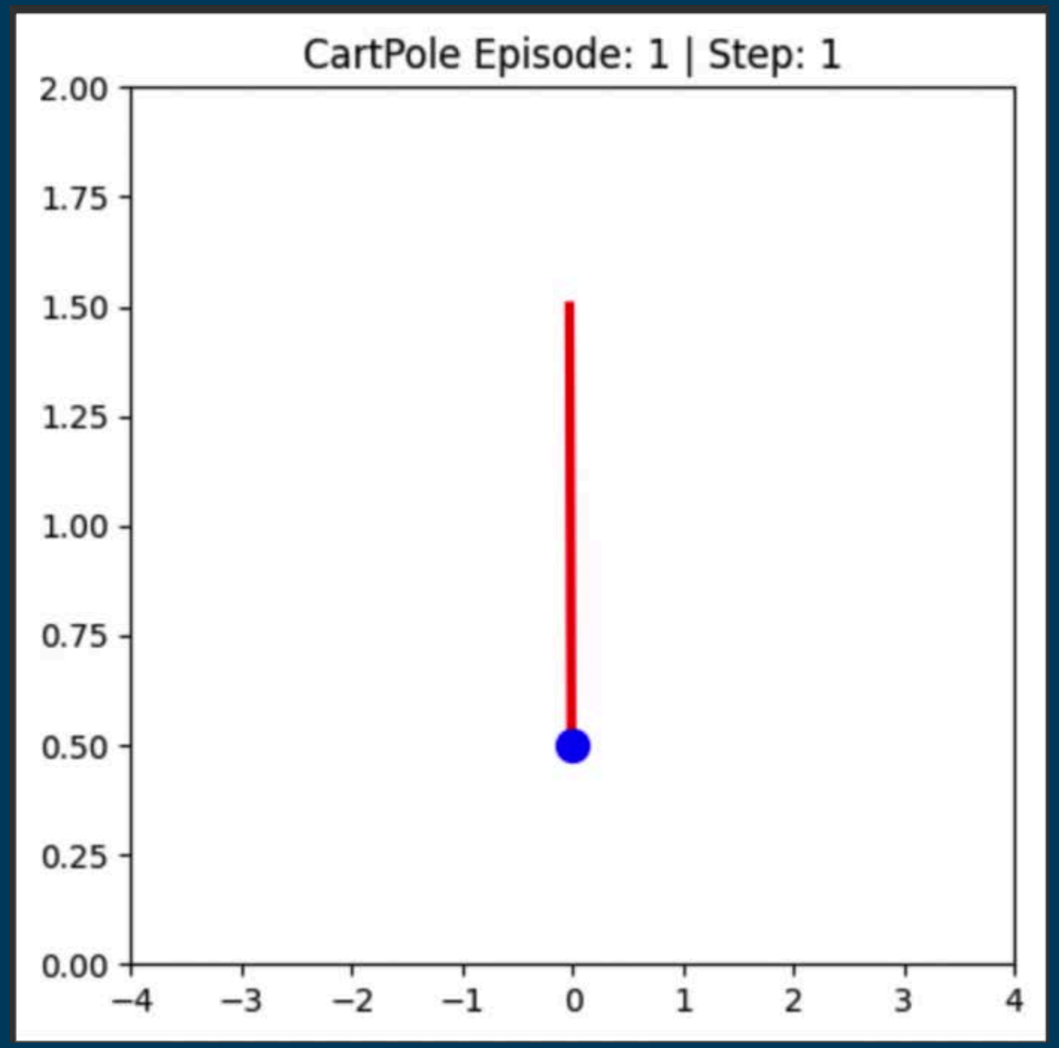
- A deep Q network is a neural network architecture that is used to approximate the Q function of an agent in an environment.
- It uses epsilon-greedy exploration to explore the environment followed by gradient-descent techniques to train the neural network.

Advantages:

- Generalization: It can learn to generalize from the training data to make decisions in unseen environments.
- End-to-End Learning: DQN learns a policy directly from raw sensory inputs (e.g. pixels), which means it doesn't require hand-crafted feature engineering.
- Sample Efficiency: It uses experience replay and target networks to stabilize learning and make efficient use of collected experiences.
- Stability: It is more reliable than q learning and prevents divergence during training better.

Disdvantages:

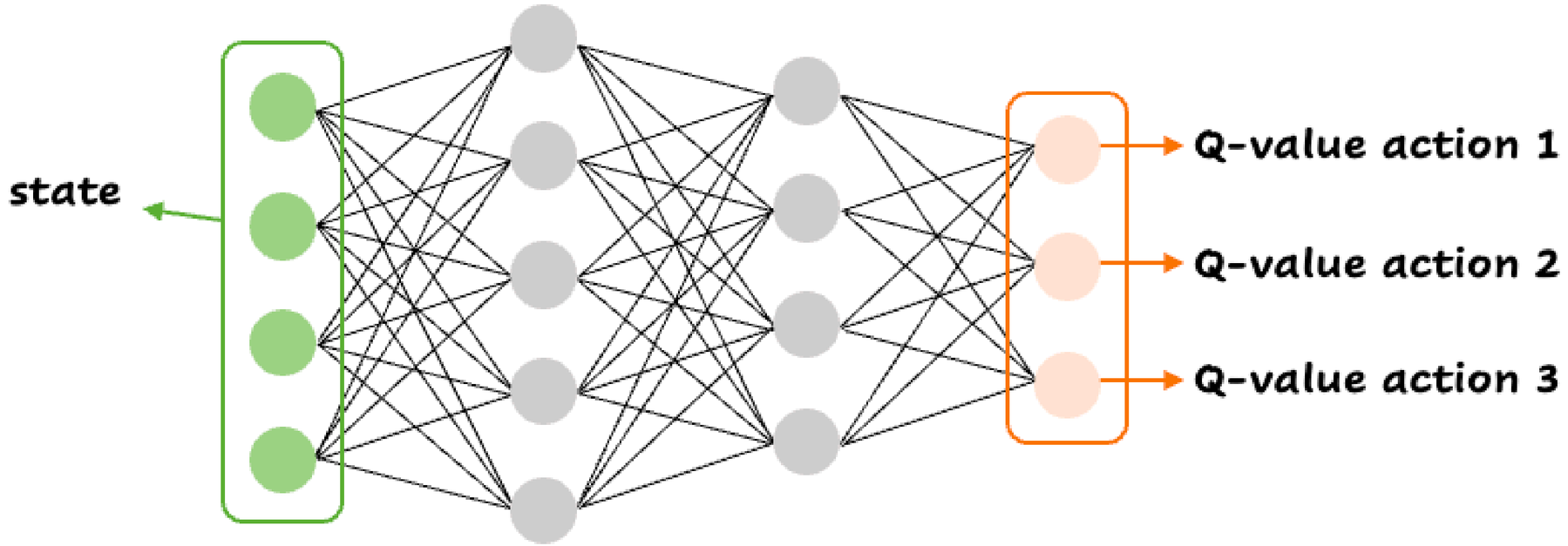
- Complexity: Implementing and fine-tuning DQN can be challenging. Hyperparameter tuning, network architecture selection, and other details require careful attention.
- Overestimation: DQN tends to overestimate the Q-values, which can lead to suboptimal policies.
- Limited Exploration: DQN's exploration strategy often relies on epsilon-greedy exploration, which can be inefficient in complex environments. More advanced exploration techniques, like using noisy networks or incorporating other exploration strategies, may be necessary.
- It can require a large amount of memory.


CartPole Episode: 1 | Step: 1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−4
−3
−2
−1
0
1
2
3
4

Tabular Q-learning

	Actions			
States	1	2	…	n
0	Q(0,1)	Q(0,2)	…	Q(0,n)
1	Q(1,1)	Q(1,2)	…	Q(1,n)
…	…	…	…	…
m	Q(m,1)	Q(m,2)	…	Q(m,n)

Deep Q-learning



Algorithm 3: Actor Critic Methods

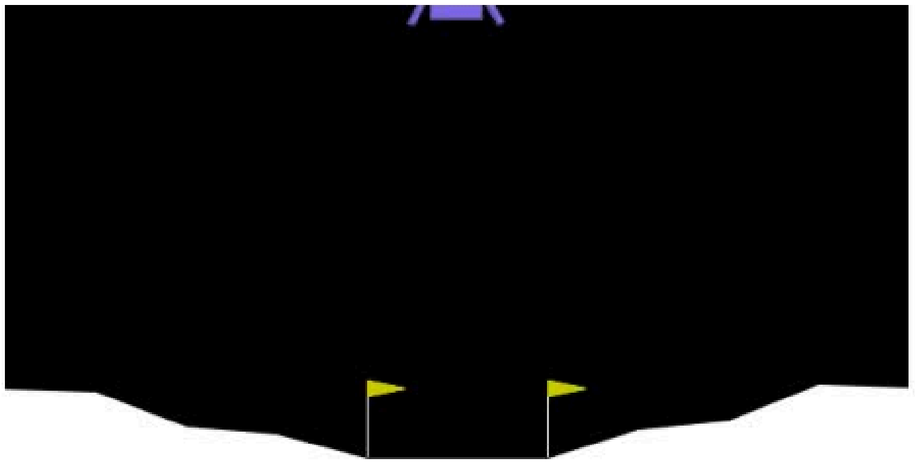
- An Actor network is responsible for learning and improving the policy. It takes the present state as an input and outputs a probability distribution over actions as output
- A critic network is used to estimate the expected-cumulative rewards i.e. Q-values. It provides a value funtion to guide the actor, and help it understand which actions are better at a given state
-

Advantages:

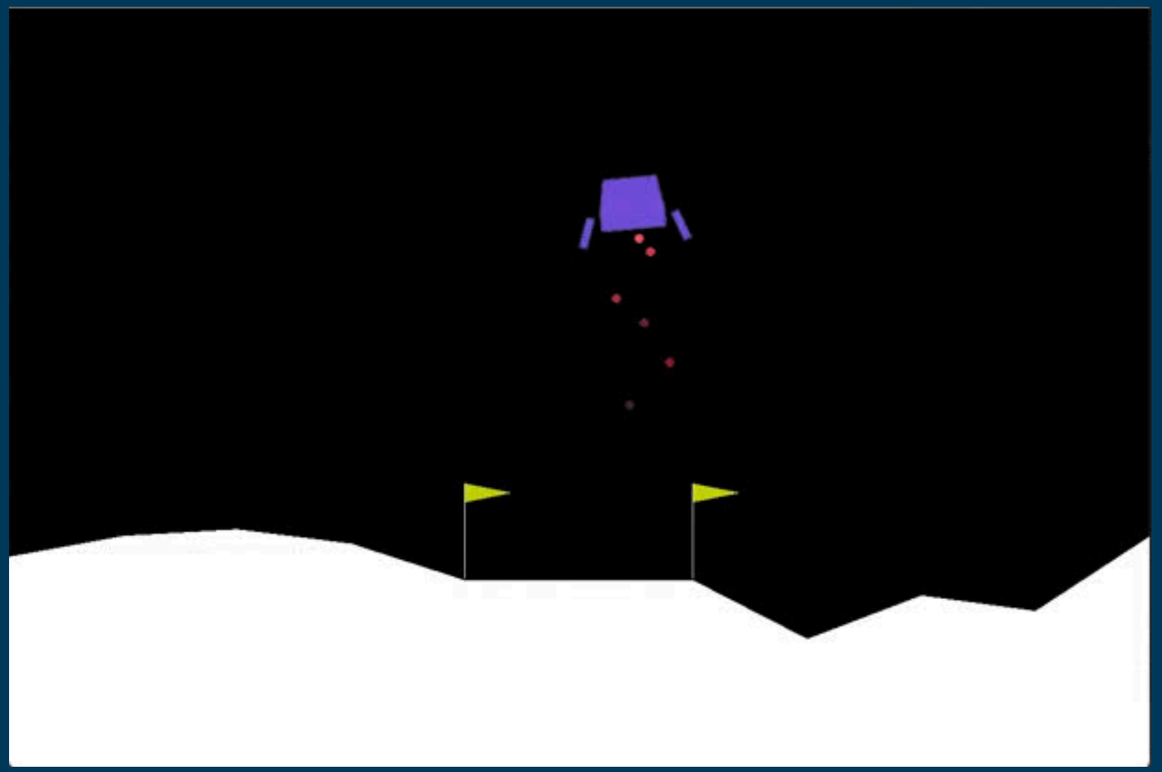
- Higher stability and convergence: The A2C provides more stability than pure-policy based methods. It uses a learned baseline, which can lead to faster convergence
- It can handle both discrete and continuous action-spaces leading to higher flexibility

Disdvantages:

- Actor-Critic methods often require careful tuning of hyperparameters. Poor choices can lead to unstable training.
- It is an on-policy algorithm, it may get stuck in suboptimal policies during learning because they are constrained to follow the current policy. Exploration can be limited, making it difficult to discover better policies when the initial policy is poor.



An Implementation- Lunar Lander



Algorithm 4: PPO

- Proximal Policy Optimization, or PPO, is a policy gradient method for reinforcement learning. The motivation was to have an algorithm with the data efficiency and reliable performance of TRPO, while using only first-order optimization.
- This utilises the concept of trajectories in policy optimization.

Advantages:

- Sample Efficiency: Very high sample efficiency
- Training Stability and Robustness: Clipped objective function prevents large Policy Updates and can handle non-deterministic environments pretty well.
- Reduced Hyperparameter Sensitivity: Handles changes in Hyperparameters well and does not require as fine tuning as other algortithms.

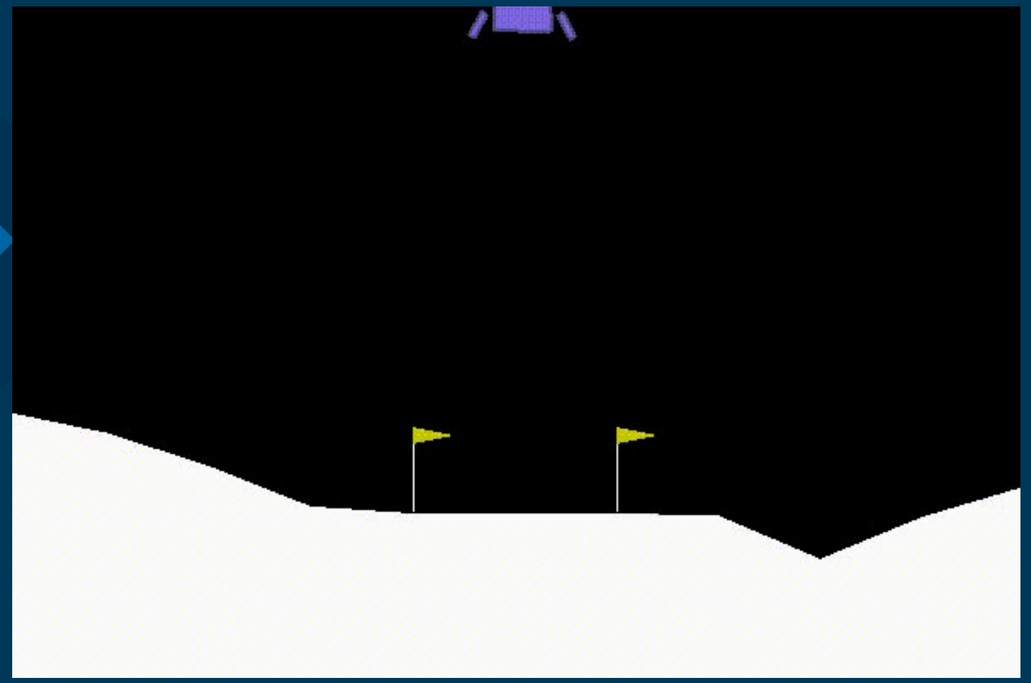
Disdvantages:

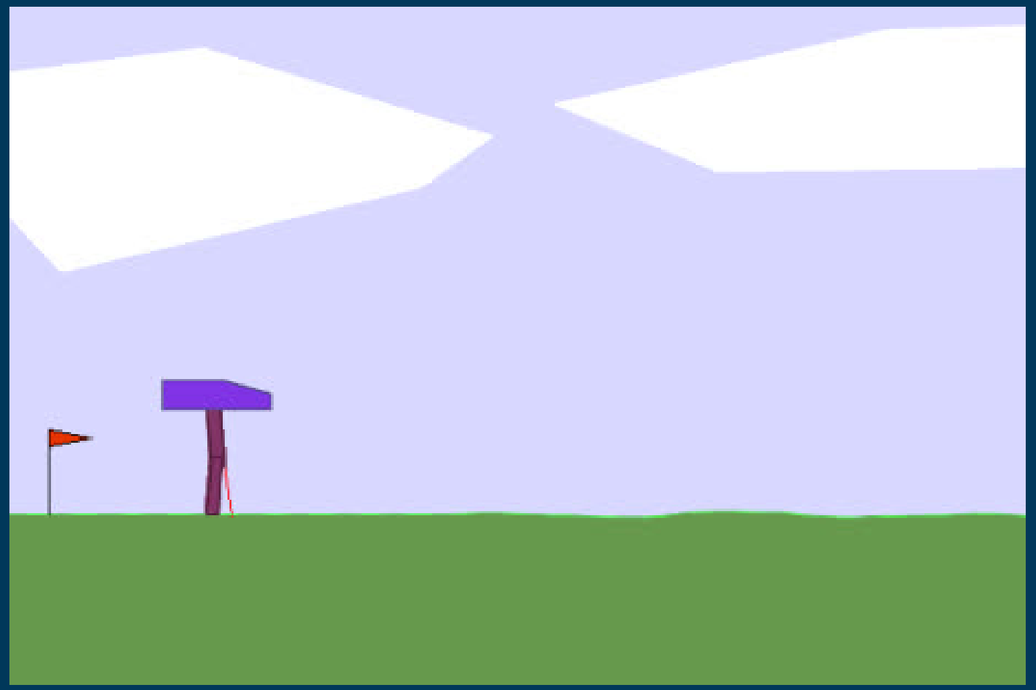
- Limited Exploration: PPO struggles in domains that require aggressive exploration due to the constrained policy updates .
- Sub-optimal Strategies: PPO may quikcly converge to a suboptimal policy due to its local optimization approach.

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

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

Implementations





CLIP:

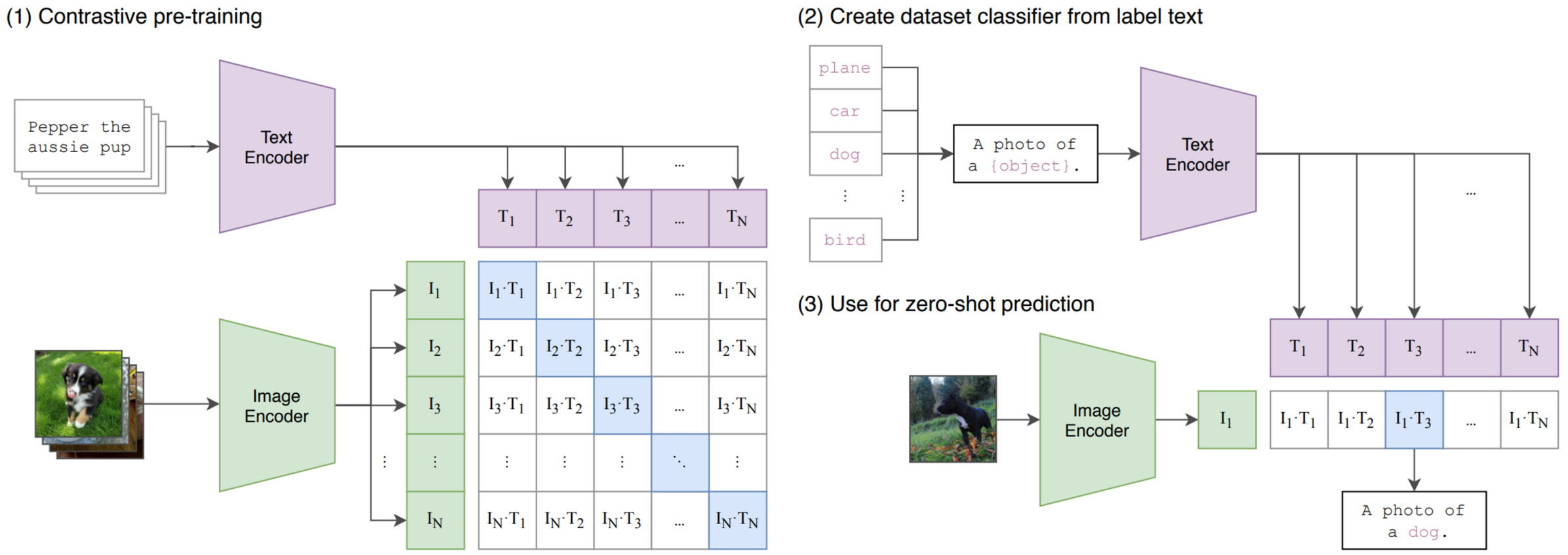


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

Advantages of CLIP

- Highly efficient as it can cover a wide varietty of objects due to the images picked being of high variance training well.
- Because they learn a wide range of visual concepts directly from natural language, CLIP models are significantly more flexible and general than existing ImageNet models. They are able to zero-shot perform many different tasks.
- One of the best algorithms to handle image and text inputs for various tasks since it implicitly handles the matching.

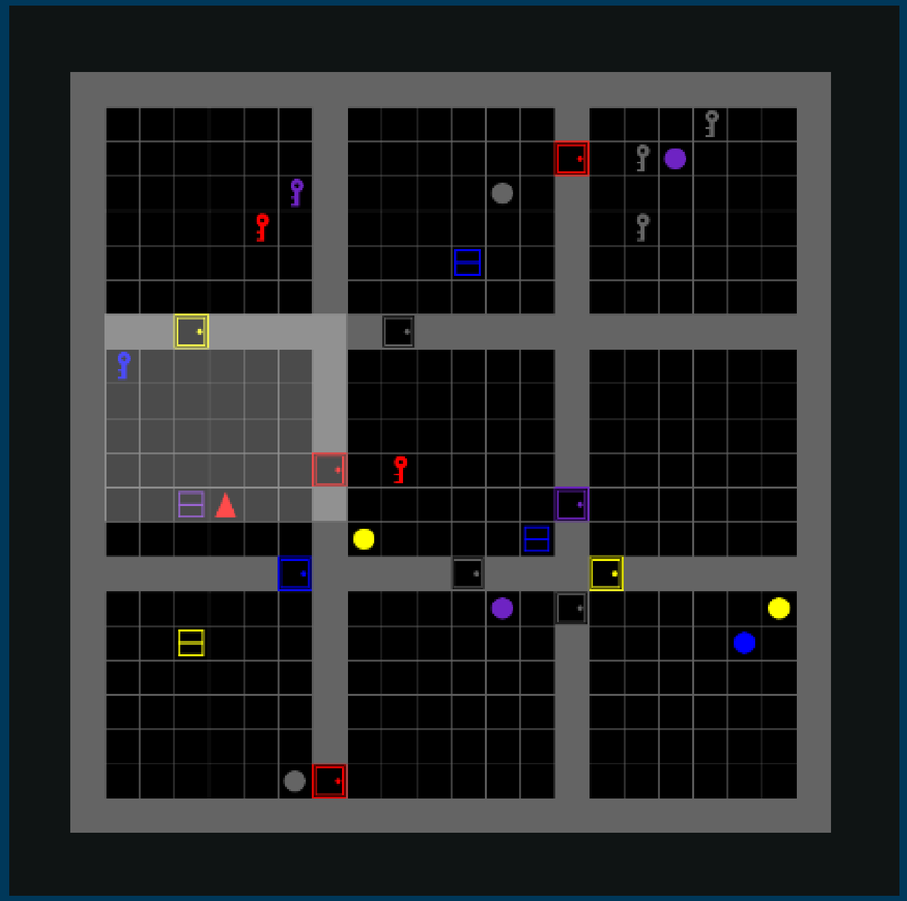
Limitations of CLIP

While CLIP usually performs well on recognizing common objects, it struggles on more abstract or systematic tasks such as counting the number of objects in an image and on more complex tasks such as predicting how close the nearest car is in a photo. On these two datasets, zero-shot CLIP is only slightly better than random guessing. Zero-shot CLIP also struggles compared to task specific models on very fine-grained classification, such as telling the difference between car models, variants of aircraft, or flower species. To tackle this we can train the CLIP architecture as per our specific needs.

BUT WHY ARE WE LOOKING INTO CLIP?

RL FOR BABY AI

Baby AI is an environment where the agent is a baby in a house and is given a “text prompt” to accomplish a task.

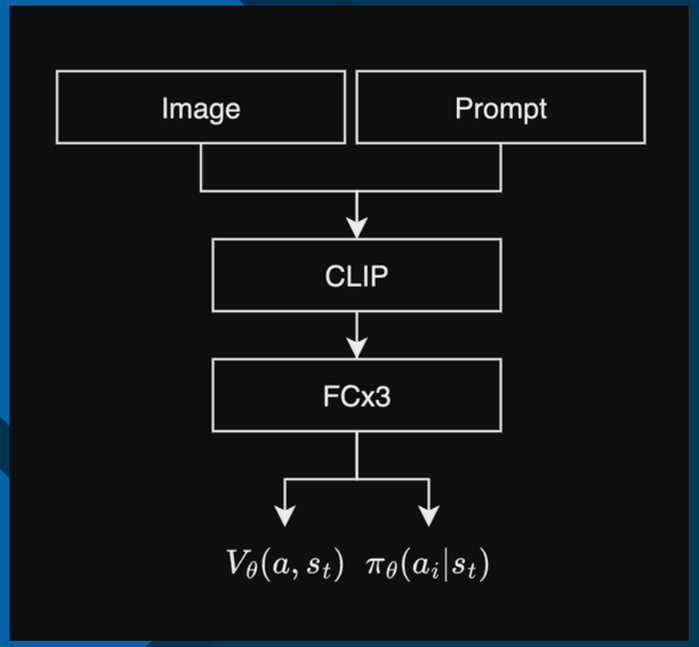


Compilation of all Research Papers read and code implementations

Compilation of all work done

CLIP for BabyAI

We implemented the CLIP encoder for BabyAI environments using PPO for training. The training was for 2500 episodes(50M frames), with results being slightly over random actions.



Major reasons why it didn’t give satisfactory results:

- CLIP is trained on real world images and text corpuses, and isn’t made for the sensor reading-like data which is just a 5x5 or 7x7 grid output.
- Text is an instruction to the agent, and not the description/content of the image of the environment.
- The pretrained model cannot distinguish between objects in the environment and the agent itself

Effectively, the model was blind and had no idea as to what it was prompted to do. As a result of this, the best strategy the model formed was to move around randomly till it eventually achieved the goal, which seldom occurred.

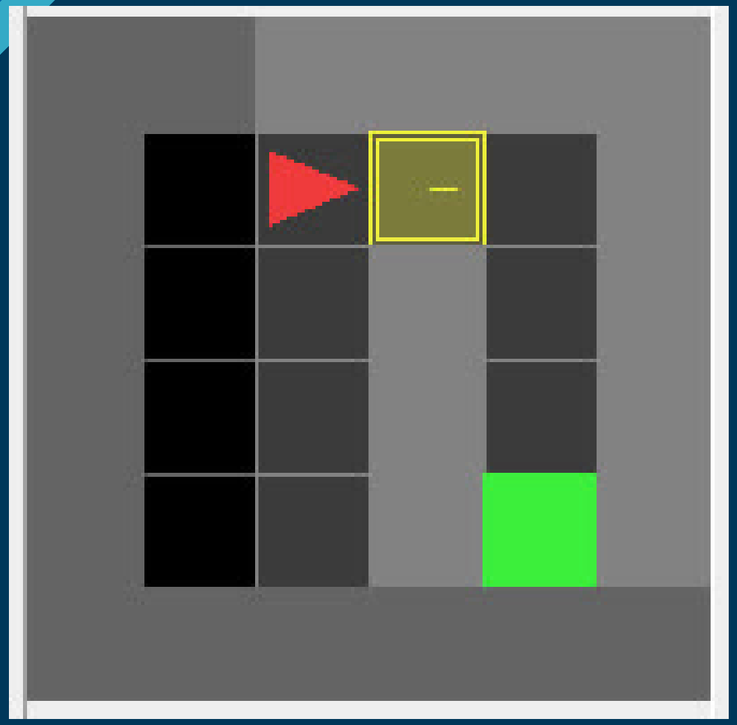
Our Solution

We used a custom 3 layer CNN to process the image. ResNets were an obvious next choice but the small number of layers meant the ResNet model performed equally well.

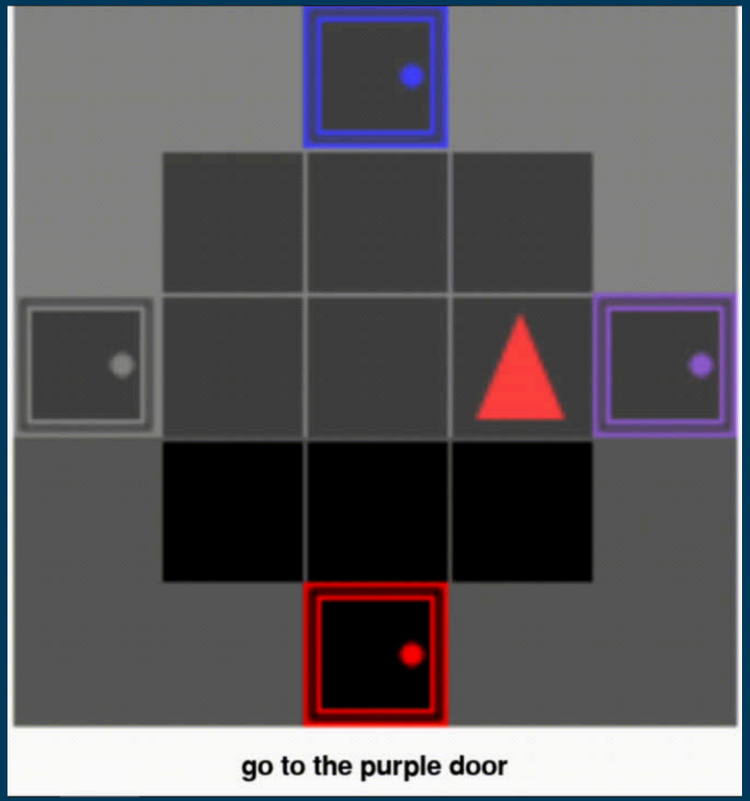
For processing the instructions, we used LSTMs. Like the previous training with CLIP, we used PPO to train all the models.

Testing on two tasks, the left simple DoorKey task model was trained for 40 episodes (80k frames). Right task was GoToDoor, where the model learnt in 500 episodes (1M frames). (on the next slide).

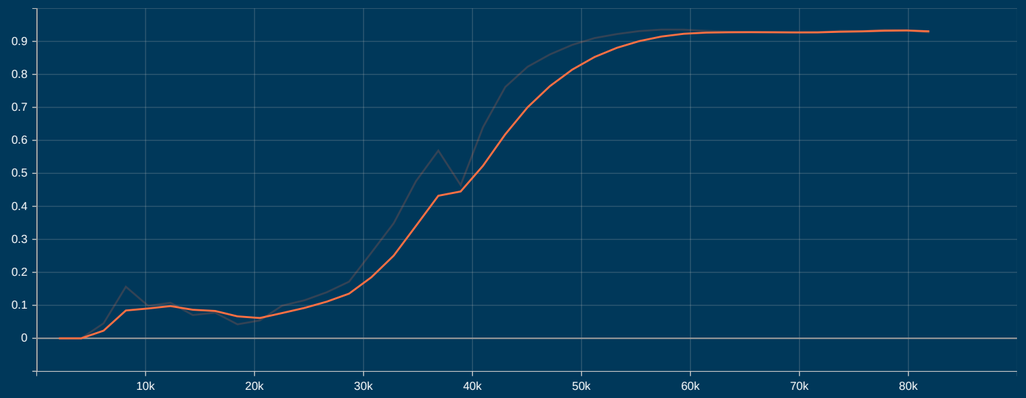
Numerically, on the two test environments, final CLIP mean rewards were 0.12% and 0.93%. The CNN+LSTM solution achieved an average reward of 93.04% and 92.4%, with a probabilistic policy. Evaluating with most likely action resulted in 100% mean rewards (Eliminating exploration).
Comparing training speed, we could solve the complex environments within 15 minutes, comparing with CLIP's saturation at around 25 minutes.



5x5 DoorKey Environment



5x5 GoToDoor Environment



TRPO

- TRPO, like other policy optimization algorithms, uses policy gradients to update the policy.
- The policy gradient represents the direction in which the policy should be modified to improve expected rewards.
- TRPO employs a surrogate objective function, typically the expected improvement in policy performance.
- This surrogate objective guides the policy update process.

- The trust region is defined in terms of a divergence measure between the new and old policies, such as the KL divergence.
- TRPO constrains the KL divergence between the old and new policies to ensure that the policy update is conservative and does not deviate too far from the current policy.

- TRPO computes the policy gradient and optimizes the surrogate objective function subject to the trust region constraint.
- It typically involves solving a constrained optimization problem to find the policy update that maximizes the surrogate objective while staying within the trust region.

TRPO

Advantages of TRPO:

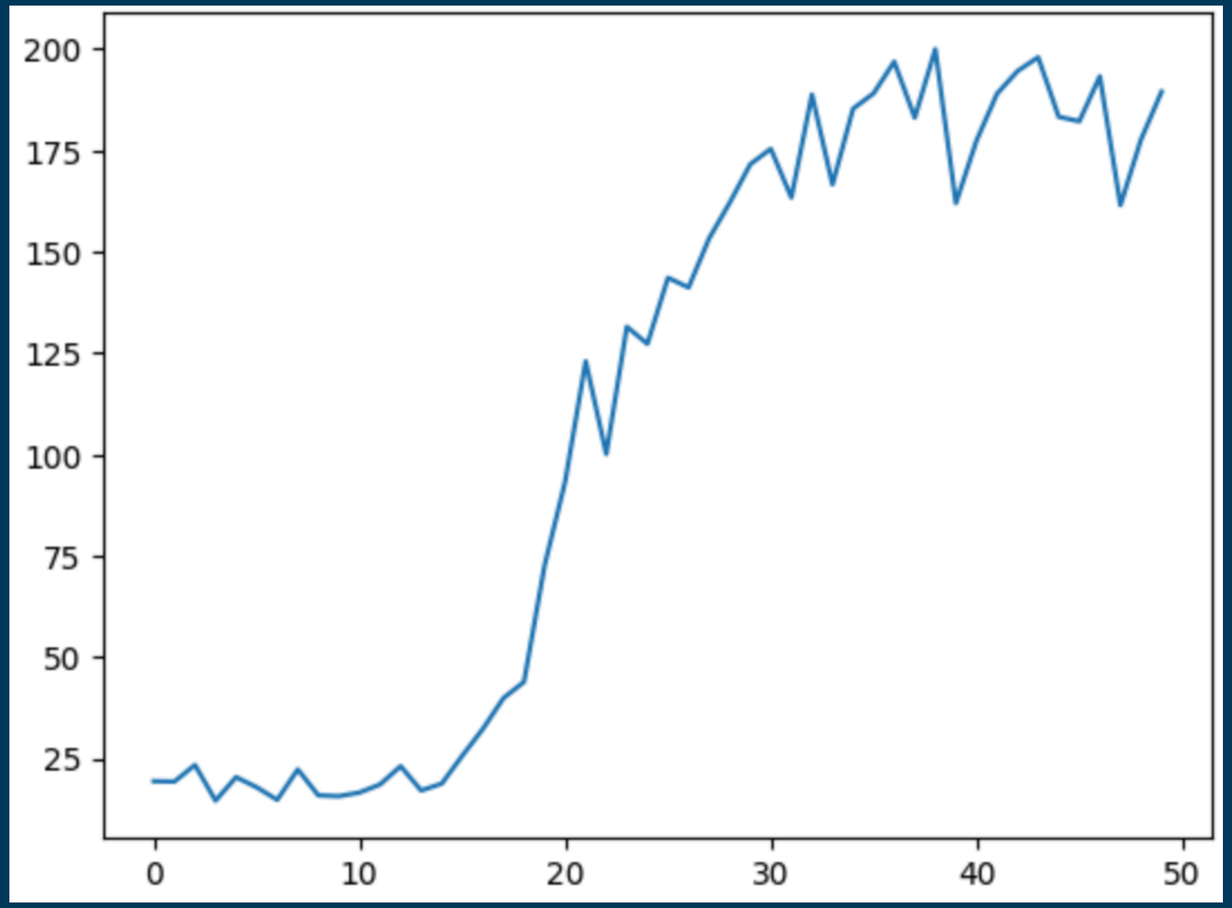
- Stability: TRPO ensures stable and monotonic policy improvements.
- Sample Efficiency: It makes efficient use of samples by constraining policy updates.
- Robustness: TRPO is robust to hyperparameter tuning and can handle a wide range of environments.

Disadvantages of TRPO:

- While TRPO offers several advantages, it may suffer from computational overhead due to the need for constrained optimization.
- Extensions and variants of TRPO, such as Proximal Policy Optimization (PPO), aim to address some of its limitations while retaining its benefits.

TRPO

Results with TRPO on cartpole:



VOYAGER

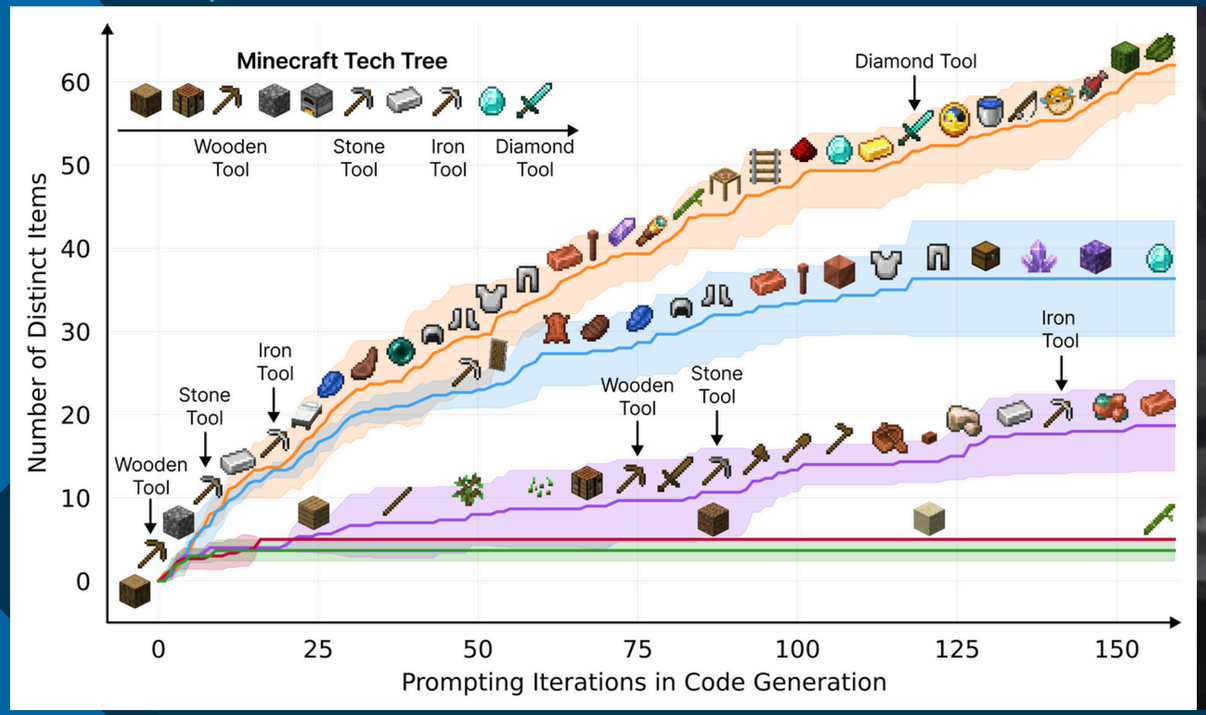
An Open-Ended Embodied Agent with Large Language Models

Voyager is an LLM-powered embodied lifelong learning agent in Minecraft that continuously explores the world, acquires diverse skills, and makes novel discoveries.

Voyager consists of three key components:

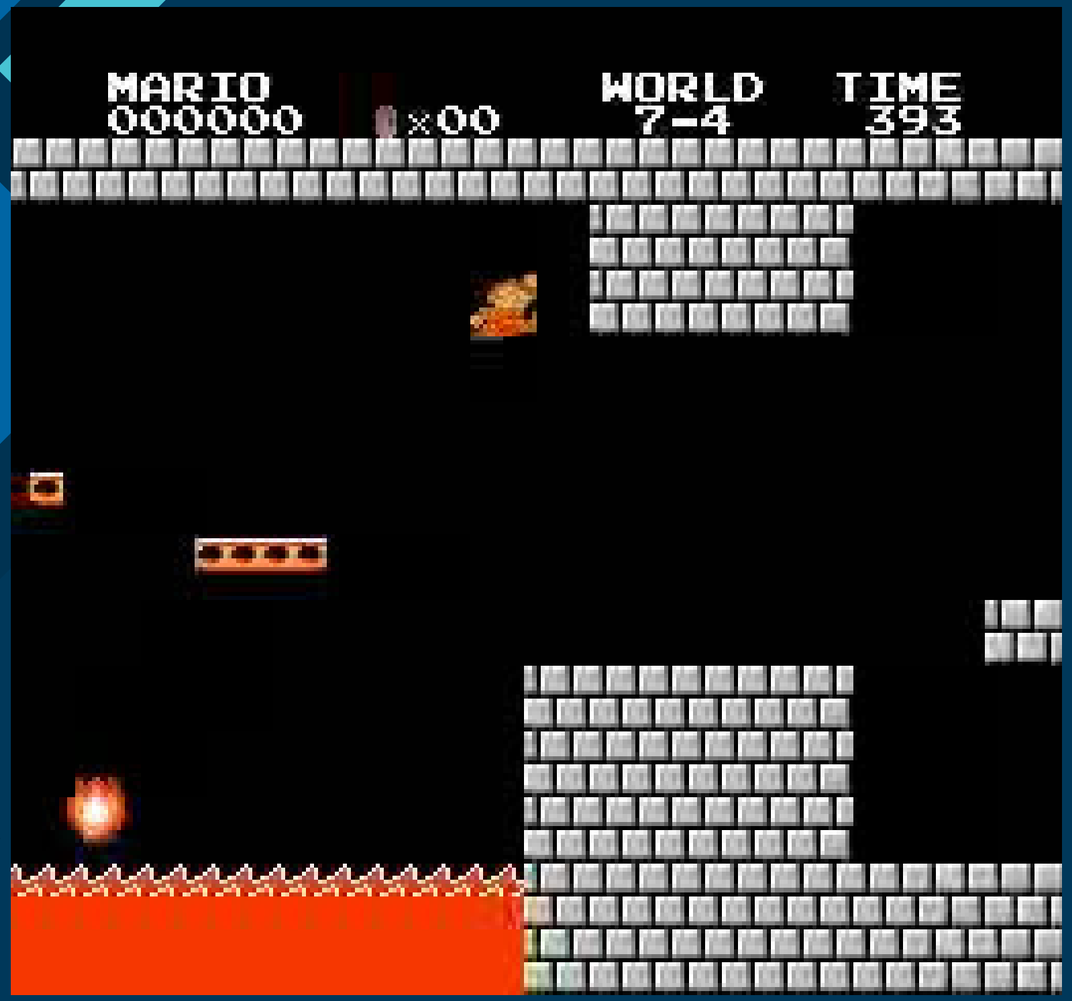
1) an automatic curriculum that maximizes exploration
2) an ever-growing skill library of executable code for storing and retrieving complex behaviors
3) an iterative prompting mechanism that incorporates environment feedback, execution errors, and self-verification for program improvement

VOYAGER


Minecraft Tech Tree
Wooden Tool
Stone Tool
Iron Tool
Diamond Tool
Number of Distinct Items
Prompting Iterations in Code Generation
0
10
20
30
40
50
60
0
25
50
75
100
125
150


Mine Amethyst

Mario with PPO



Curiosity Driven Learning

In this method, we look another aspect of reinforcement learning that we augment (or introduce where absent) the reward function. Instead of the environment providing the reward, which might not be the case in many situations, the model itself gives itself bonus points if it finds something new and/or useful. This drives the model to throughly explore the environment and the correlations between various objects.

The model is demonstrated to learn generalizable skills, instead of focusing only on the single environment it is dropped into. This approach of curiosity has it’s downsides too. Environments with extremely high supply of information, similar to a TV in room, provides so much information that the agent stops the exploration of the physical environment.

Curiosity Driven Learning

Generally faster times on Mario compared to PPO, along with finding out shortcuts in worlds like 1-1

Solving games in the Gym Arcade set of environments

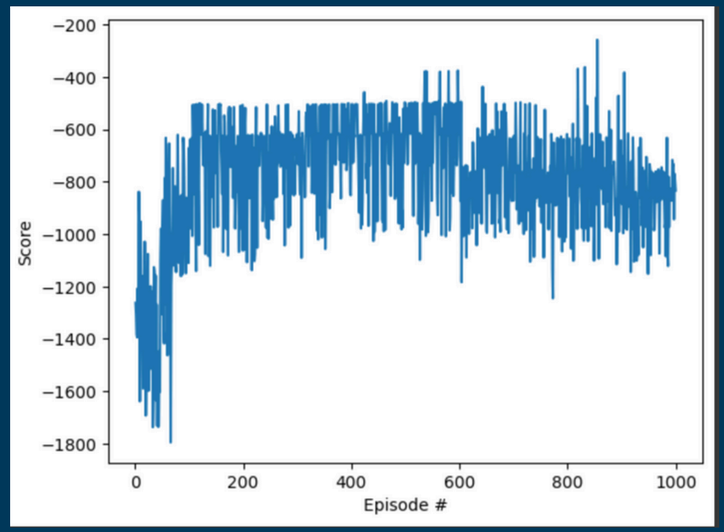
Curiosity Driven Learning

Multi agent applications

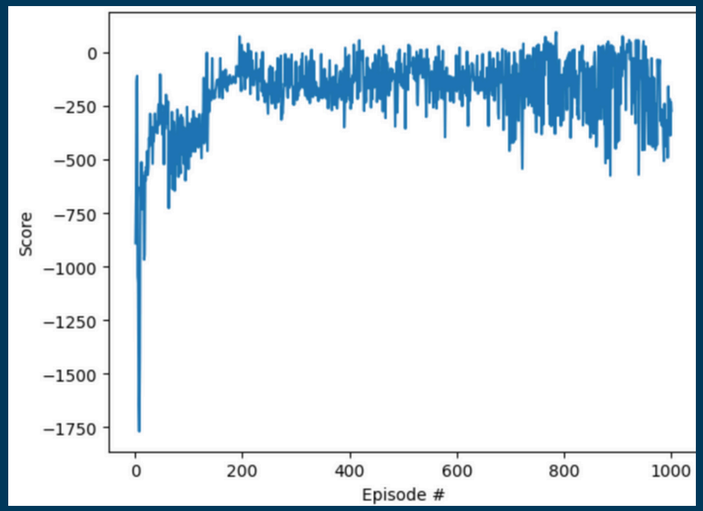
Agent stuck in front of a TV

DDPG

- Uses an actor-critic architecture, consisting of two neural nets: an actor and a critic.
- Uses a deterministic policy, meaning that given a state, the actor network outputs a specific action, unlike the a2c alogrithm which uses a stochastic policy.
- DDPG uses off-policy learning, and stores experiences in a replay buffer and samples mini-batches randomly during training.
- DDPG uses soft updates, where the target networks slowly track the main networks by interpolating their weights.



Results for Pendulum environment



Results for Lunar-Lander environment

Advantages of DDPG

- **Handling Continuous Action Spaces:**DDPG excels in handling environments with continuous action spaces
- **Deterministic Policy:** Its deterministic policy output allows for easier and more straightforward learning in continuous action spaces compared to stochastic policies.
- **Experience replay:** DDPG can leverage past experiences stored in a replay buffer, which enhances sample efficiency and stabilizes learning by breaking the temporal correlations in the data.

Limitations of DDPG

- **Hyperparameter Sensitivity:** DDPG can be sensitive to hyperparameters, and tuning them for optimal performance can be challenging
- **Sample Inefficiency:** While experience replay enhances sample efficiency, DDPG might still require a large number of samples to effectively learn
- **Exploration-Exploitation Trade-off:** Its deterministic policy might struggle with exploration as it can get stuck in local optima

Object-sensitive Deep-RL

1. Exploits object characteristics such as presence and positions of game objects in learning phase. Adaptable to most RL frameworks.
2. This also aims to make the agent more explainable and help model the loss function accordingly.
3. Localizes different objects in the environment that is then fed to the agent that then learns to classify them as ‘good’ and ‘bad’ objects.
4. The method makes use of object saliency maps built up from pixel saliency maps to see what information is the agent using to make a decision.

Object-sensitive Deep-RL

The w is approximated to get a linear mapping of the state as otherwise Q is a highly non linear function of s

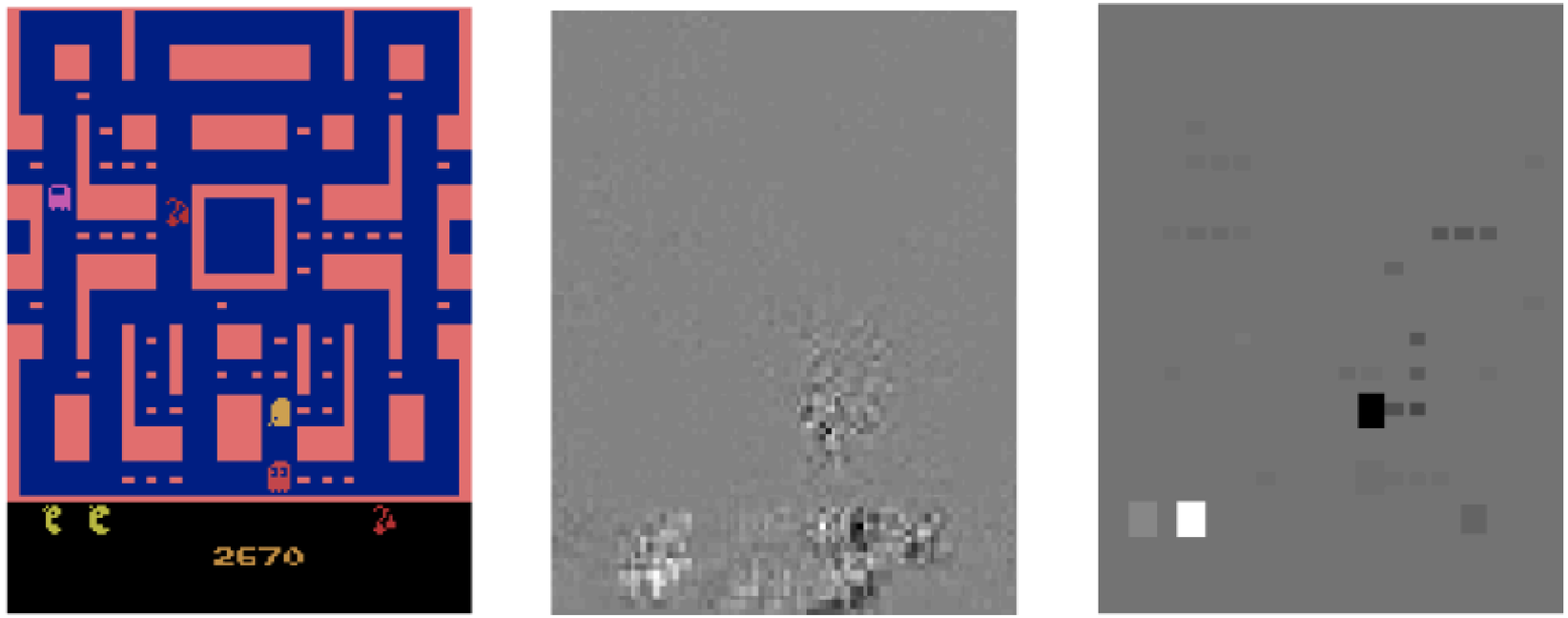
$$Q(s,a) \approx w^T s + b, \tag{10}$$

where w is the derivative of $Q(s,a)$ with respect to the state image s at the point (state) s_0 and form the pixel saliency map:

$$w = \frac{\partial Q(s,a)}{\partial s}|_{s_0} \tag{11}$$

$$w = Q(s,a) - Q(s_o,a) \tag{12}$$

Object-sensitive Deep-RL



(a) Original State (b) Pixel Saliency Map (c) Object Saliency Map

Figure 2: An example of original state, corresponding pixel saliency map and object saliency map produced by a double DQN agent in the game "Ms. Pacman."

RESEARCH PAPER IMPLEMENTATION

PRIORITIZED EXPERIENCE REPLAY

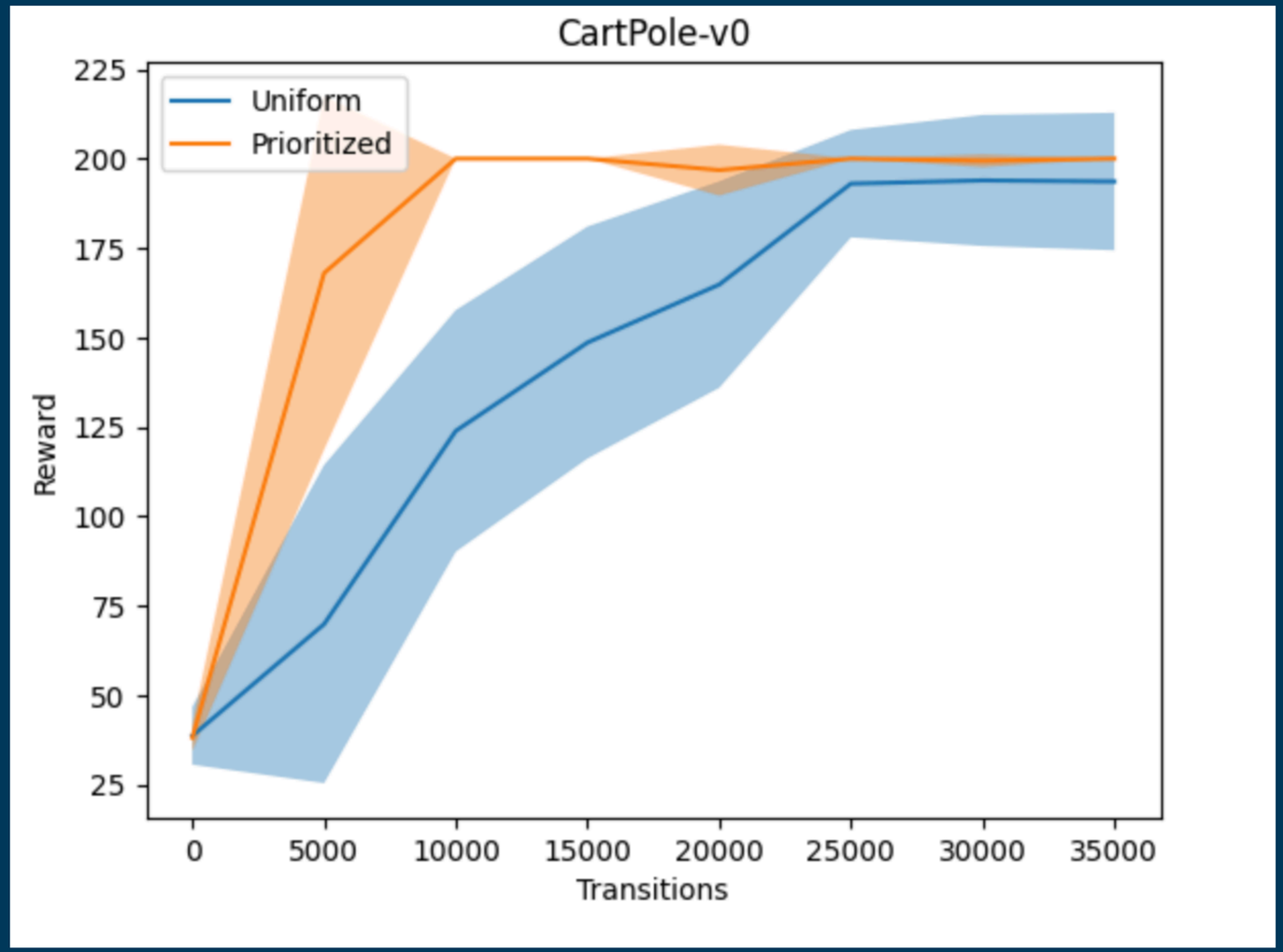
- In an ordinary DQN, we uniformly sample experience transitions from replay memory.
- Instead, we now use a framework for prioritizing experience, to replay more important transitions more frequently.
- The TD error is used as a basis to assign priority to transitions.
- We then assign the probability P(i) of sampling as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- Here, pi is the priority and alpha is a hyperparameter to control the level of prioritization

RESULTS OF PRIORITIZED REPLAY ON CARTPOLE

We see that our Rewards converge much faster in case of prioritzed replay, compared to uniform sampling



Future Work

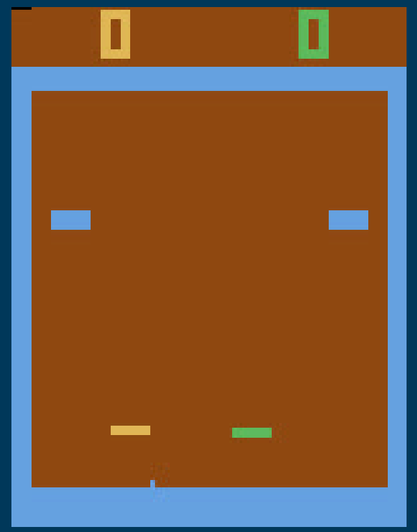
IMPLEMENTING RESEARCH PAPERS

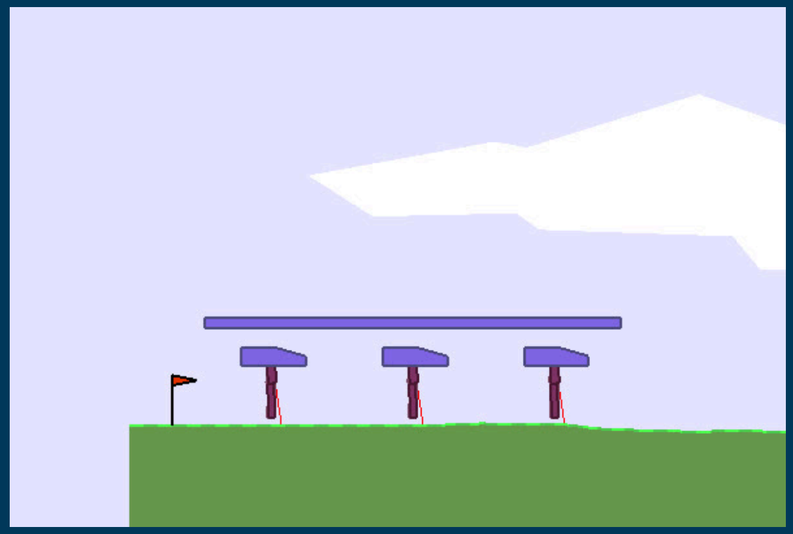
- **State-of-the-Art Algorithms:** We intend to review research papers on state-of-the-art algorithms to stay current with the latest research. This will enable us to leverage the most effective and efficient algorithms available.
- **Novel Algorithms and Environments:** We aim to search research papers for innovative environments and algorithms and plan to apply these methods to new and improved environments.
- **Implementations:** We propose to implement selected research papers to reproduce the results they present. Additionally, we plan to apply these techniques to various scenarios or environments to assess their effectiveness in different contexts.

MULTI -AGENT ENVIRONMENTS

- Shifting from single-agent setups, our focus turns to multi-agent standard environments like Pong, Tic tac toe, and Multiwalker to tackle increased complexity and interaction dynamics.

- We'll employ PettingZoo library to seamlessly integrate multi-agent environments, streamlining our development process and ensuring adaptability to more intricate scenarios.





COMPLEX BABY AI AND MARIO RL ENVIRONMENTS

- Transitioning from relatively straightforward and standard environments, our attention is now directed towards more complex environments. These environments incorporate both image and text inputs and present a higher level of difficulty, such as the Complex Baby AI environments and Mario RL.

- We plan to employ algorithms to these environments and evaluate their performance. These environments act as two-dimensional, simplified versions of three-dimensional real-world scenarios, providing a valuable testing ground for our algorithms.