COMPARISON OF REINFORCEMENT LEARNING ALGORITHMS

Course: CSE 546: Introduction to Reinforcement Learning

Velivela Vamsi Krishna
vvelivel@buffalo.edu

Sudhir Yarram
sudhirya@buffalo.edu

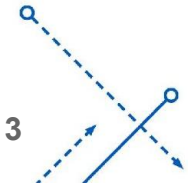
University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

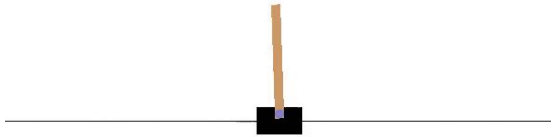
PROJECT DESCRIPTION

Algorithms Implemented

- DQN
- DoubleDQN
- A2C
- REINFORCE
- PPO (Proximal Policy Optimization)
- PPO with MultiProcessing

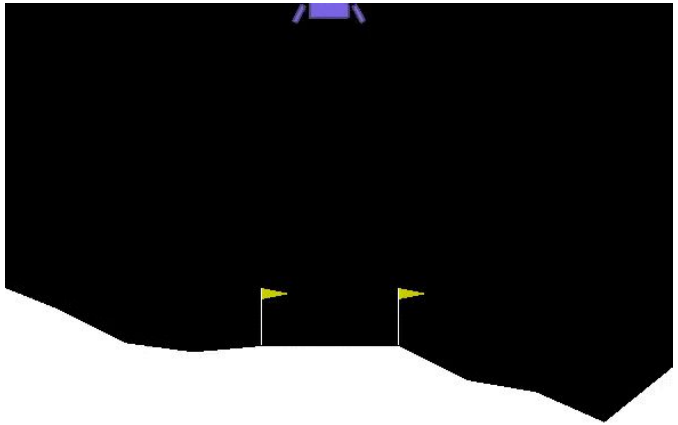


Environments



Cartpole

Space Invaders



Lunar Lander

DQN

- Uses experience replay to learn from all past policies.
- Freezes target Q-network to avoid the moving target issue
- Clip rewards or normalize network adaptive to sensible range

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

Double DQN

- Double DQN is a value based algorithm similar to DQN.
- Uses two networks to reduce this overoptimism, resulting in more stable and reliable learning.

$$Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$$

Advantage Actor-Critic algorithm

- Hybrid algorithm that combines both value based learning and policy gradients.
- Actor- Estimates action based on policy
- Critic – Estimates the Value function of the action taken by the actor and evaluates the action

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t))$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Reinforce Algorithm

- Reinforce is a policy gradient algorithm.

- Actions with higher expected reward have a higher probability value for an observed state.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)]$$

PPO (Proximal Policy Optimization)

- We maintain two networks, one with the current policy that we want to refine and second that we use to collect samples
- We clip the objective and calculate the ratio between the new policy and old policy

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \underset{\tau \sim \pi_k}{\mathrm{E}} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

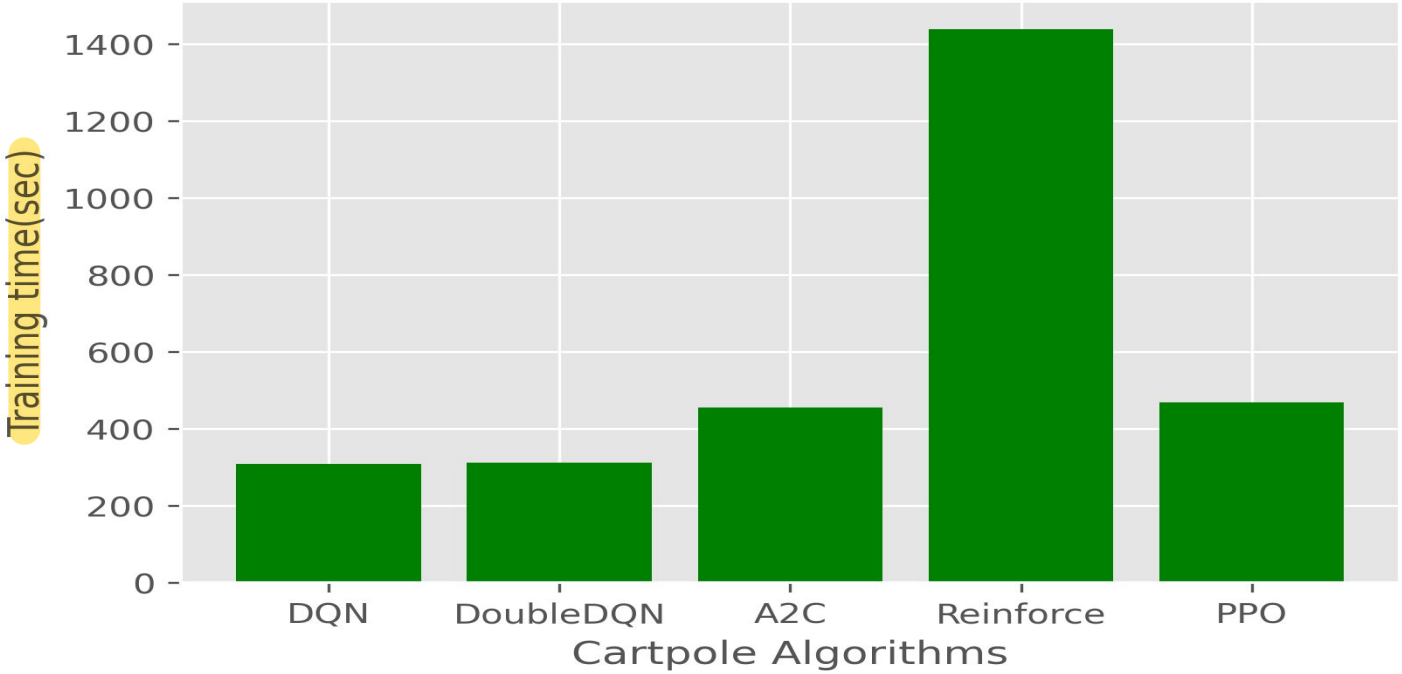
University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

RESULTS

Algorithms on Cartpole

Training time on Cartpole

We trained each algorithm for 2500 episodes on NVIDIA K80 GPU

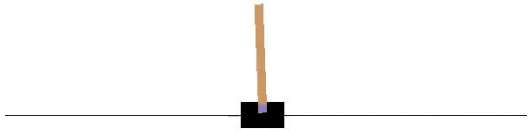


Analysis/Insights

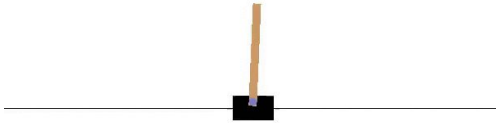
- Reinforce Algorithm converges in fewer steps when compared to other algorithms.
- Though PPO doesn't give better results in the initial phase, it starts giving significantly better results after sometime.
- Double DQN gives better result when compared to DQN
- A2C algorithms varies drastically with minor changes in hyperparameters.
- Conclusion : PPO is the best algorithm for solving this task. Even though PPO takes less time to train, it gives better and stable results when compared to other algorithms.

Cartpole

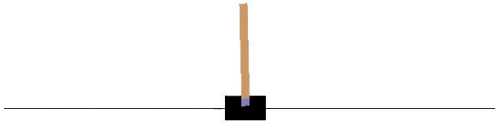
We were able to reach the threshold of 500 reward points.



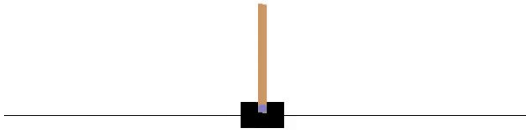
DQN



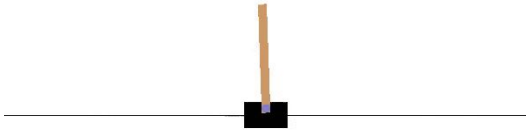
Double DQN



A2C

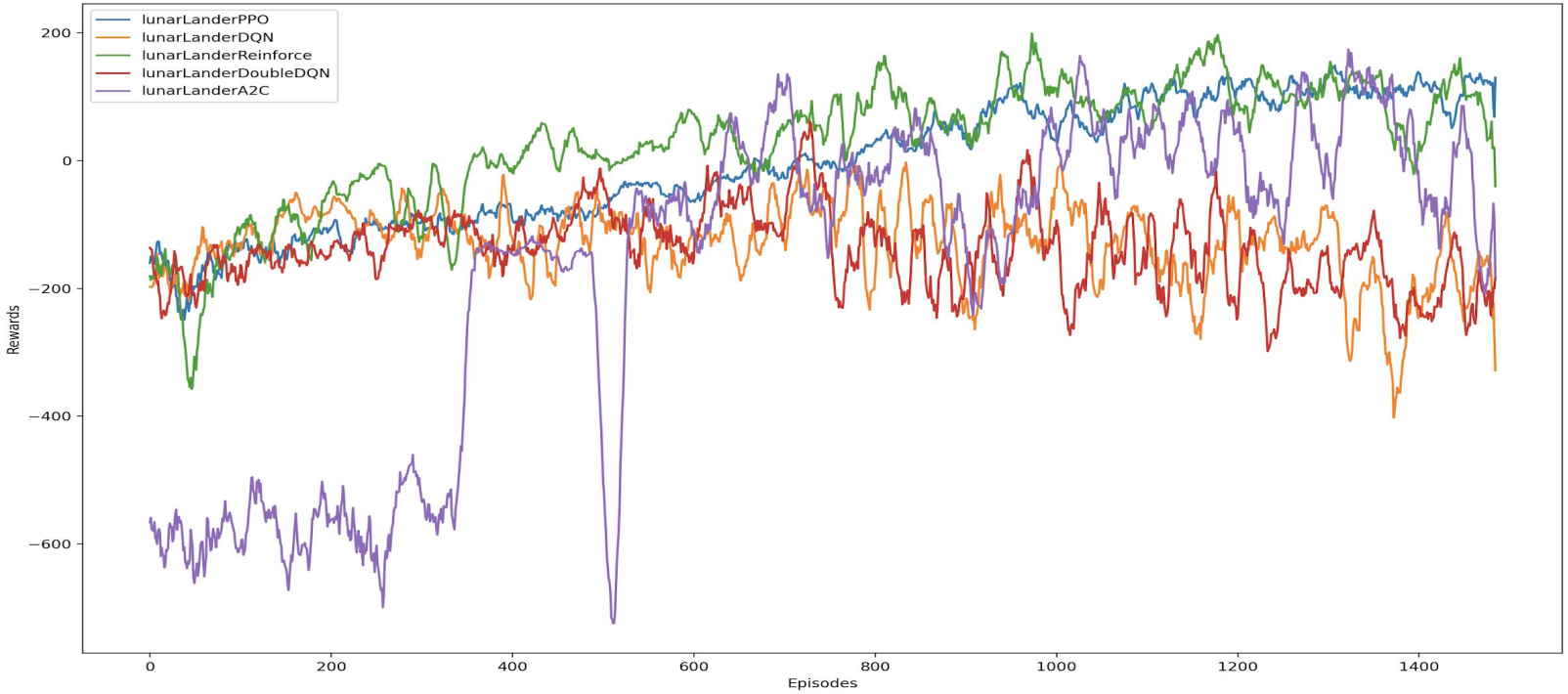


Reinforce



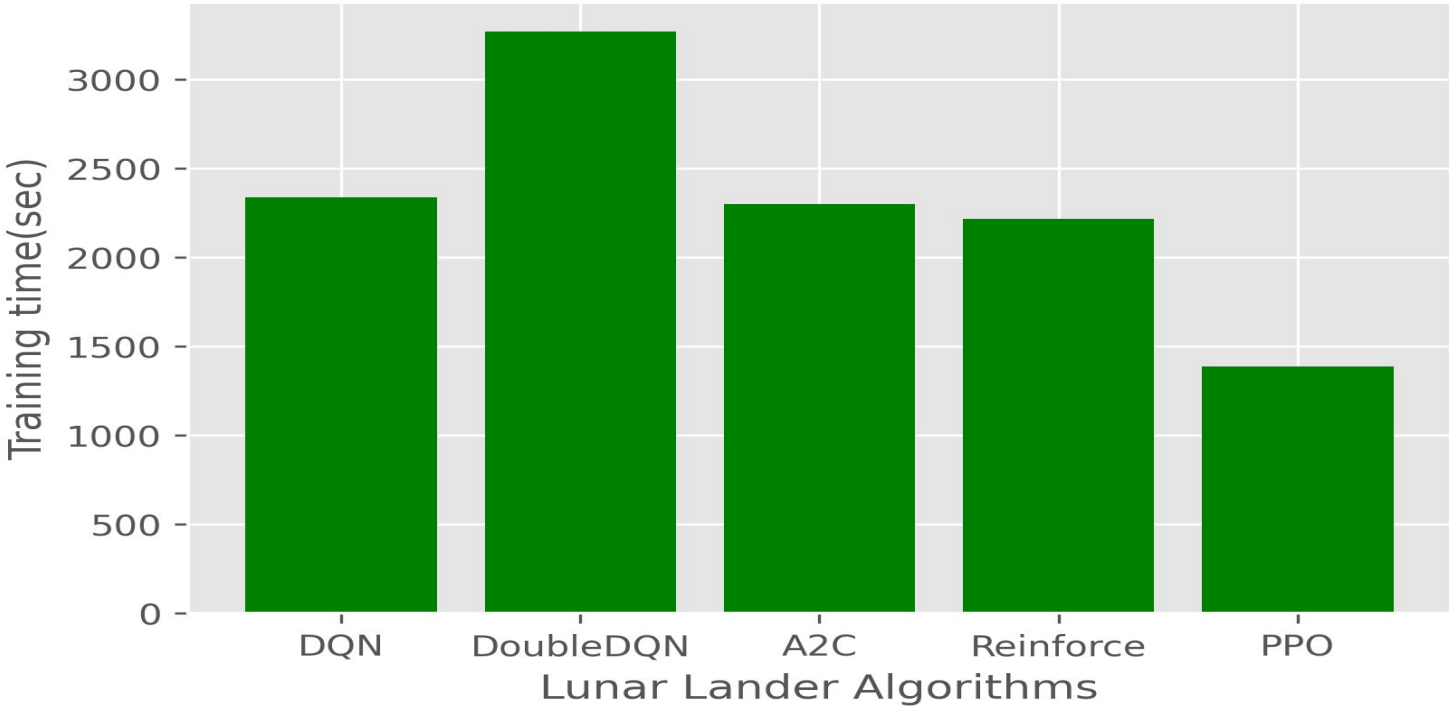
PPO

Algorithms on Lunar Lander



Training time on Lunar Lander

We trained each algorithm for 1500 episodes on NVIDIA K80 GPU

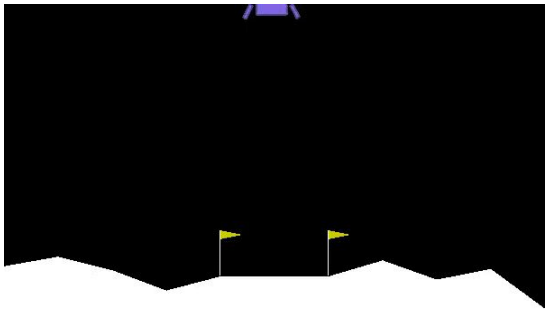


Analysis/Insights

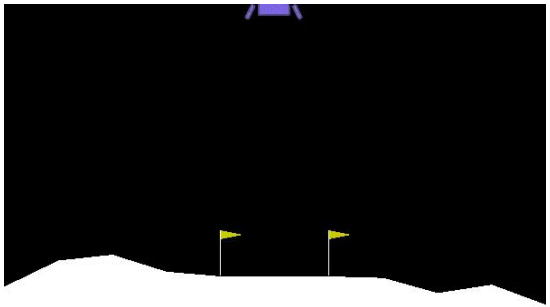
- Reinforce Algorithm, A2C and PPO gives significantly better results when compared to DQN and Double DQN
- PPO takes the least amount of time as the complexity of the environment increases.
- Double DQN gives better result when compared to DQN.
- A2C algorithms varies drastically with minor changes in hyperparameters.

Lunar Lander

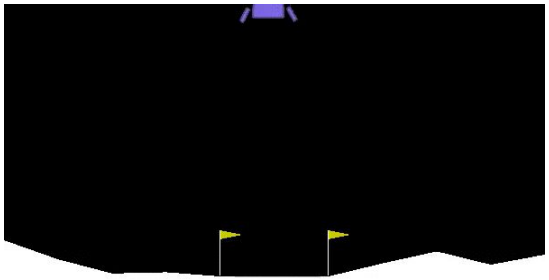
We were able to reach the threshold of 200 reward points in A2C,Reinforce and PPO after training for 1500 episodes



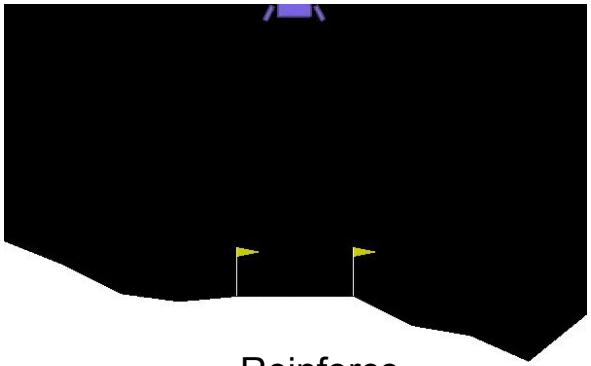
DQN



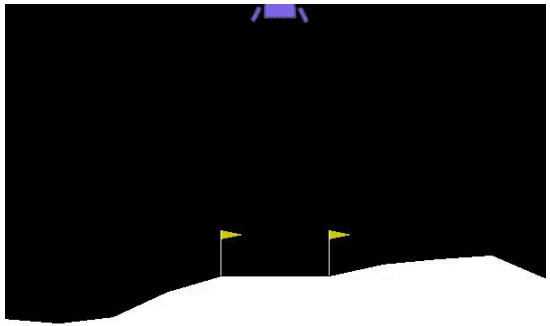
Double DQN



A2C



Reinforce



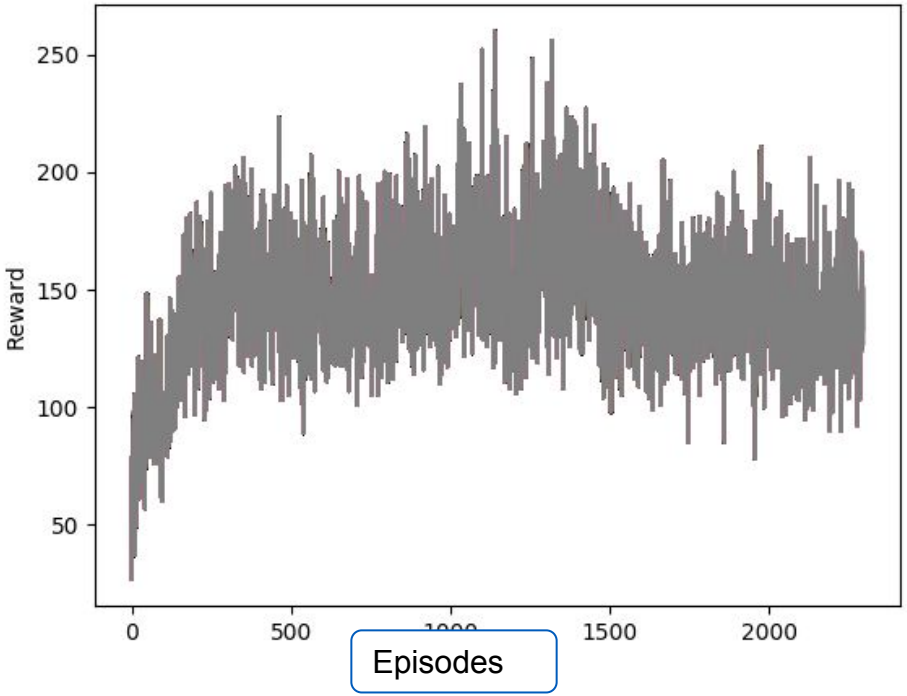
PPO

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Multi Processing

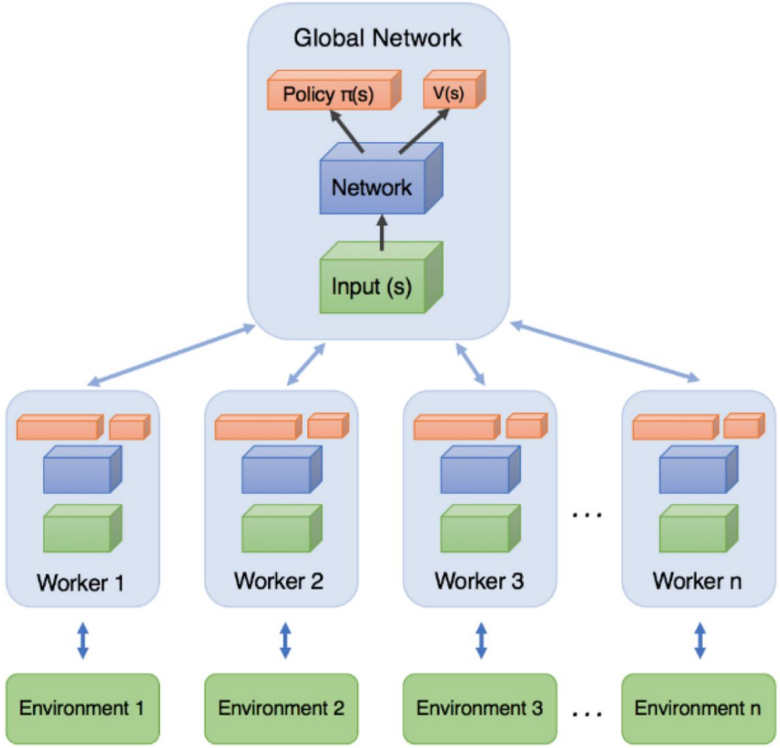
Space Invaders

Algorithm: PPO



Space Invaders

Algorithm: PPO

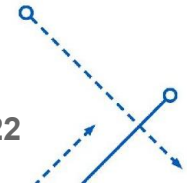


Space Invaders

```
1 import torch.multiprocessing as mp
2 os.environ['OMP_NUM_THREADS'] = '1'
```

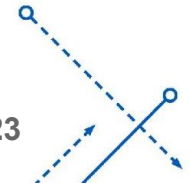
#Instantiate the processes

```
1 args_processes = 20
2 processes = []
3 for rank in range(args_processes):
4      p = mp.Process(target=train, args=(sharedagent, sharedoptimizer, rank, args, writer))
5      p.start() ; processes.append(p)
6 for p in processes: p.join()
```



Space Invaders

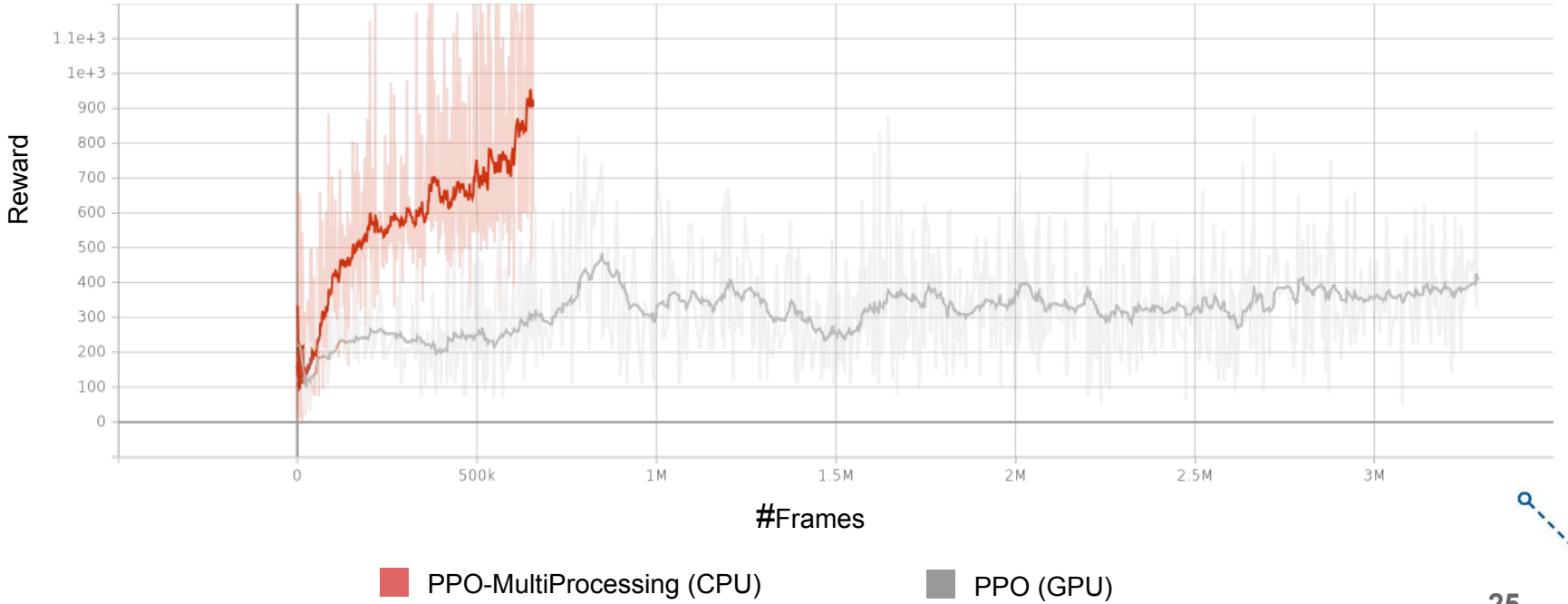
```
1 class SharedAdam(torch.optim.Adam): # extend a pytorch optimizer so it shares grads across processes
2   def __init__(self, params, lr=1e-3, betas=(0.9, 0.999), eps=1e-8, weight_decay=0):
3       super(SharedAdam, self).__init__(params, lr, betas, eps, weight_decay)
4       for group in self.param_groups:
5           for p in group['params']:
6               state = self.state[p]
7               state['shared_steps'], state['step'] = torch.zeros(1).share_memory_(), 0
8               state['exp_avg'] = p.data.new().resize_as_(p.data).zero_().share_memory_()
9               state['exp_avg_sq'] = p.data.new().resize_as_(p.data).zero_().share_memory_()

10       def step(self, closure=None):
11           for group in self.param_groups:
12               for p in group['params']:
13                   if p.grad is None: continue
14                   self.state[p]['shared_steps'] += 1
15                   self.state[p]['step'] = self.state[p]['shared_steps'][0] - 1
16           super.step(closure)
```



Space Invaders

```
1 sharedoptimizer.zero_grad()
2 loss.backward()
3 nn.utils.clip_grad_norm_(agent.parameters(), args.max_grad_norm)
4 for param, shared_param in zip(agent.parameters(), sharedagent.parameters()):
5       # sync gradients with shared model
6       if shared_param.grad is None: shared_param._grad = param.grad/args_processes
7       else : shared_param._grad += param.grad/args_processes
8 sharedoptimizer.step()
```

Space Invaders

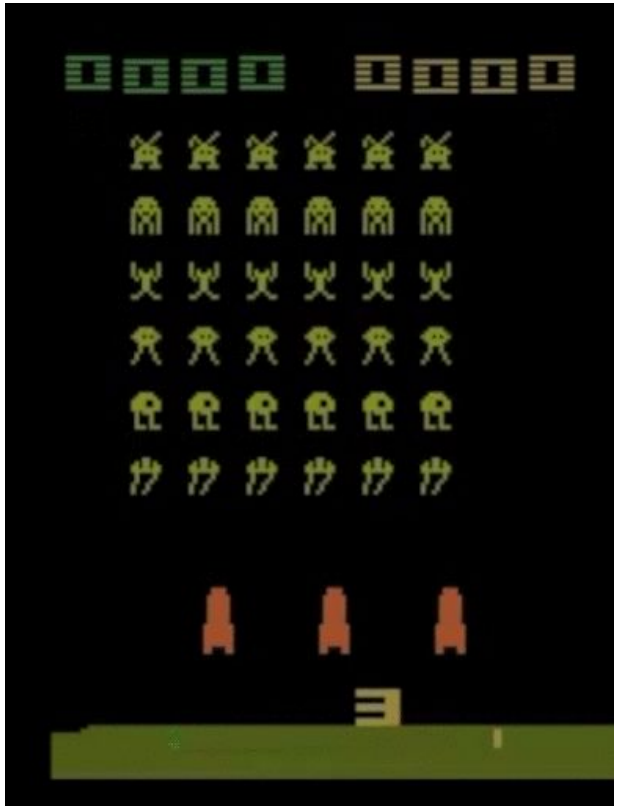


Analysis/Insights

- Learning from diverse experiences is a key component to build efficient RL models.

Space Invaders

- PPO with Multi Processing on CPU



Summary

- We experimented with multiple algorithms. PPO is the state of the art algorithm and by far the best algorithm, it achieves the maximum reward in less steps and with very less variance. It even takes less time to train.
- To incorporate learning from diverse experiences using multiprocessing is very beneficial.

References

- Playing Atari with Deep Reinforcement Learning-
- https://spinningup.openai.com/en/latest/algorithms/ppo.html
- https://arxiv.org/abs/1707.06347

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

THANK YOU