Change in Behavior and Transfer with Change in Properties of the Environment

CB

Aditya Anantwar (19CS10006)

Abhinandan De (19CS10069)

Instructor: Prof. Aritra Hazra

November 11, 2022

CS60077

1 Introduction

This project is aimed towards analyzing the change in behavior of the reinforcement learning based system with change in properties of the environment. We do our analysis on the **CartPole-v1** environment which is a version of the cartpole problem (Brockman et al., 2016). It consists of a pole attached by an unactuated joint to a cart which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces to the left and right directions of the cart.

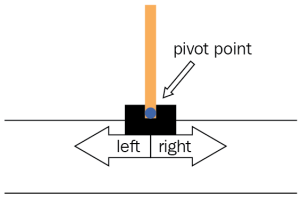

Figure 1: Open AI's CartPole environment

We simulate changes in properties of the environment by changing the physical properties of the cartpole such as mass, length and also try altering the magnitude of force and gravity. We utilize two popular RL algorithms: **Deep Q-Networks (DQN)** and **Proximal Policy Optimization Gradient Descent (PPO)** for analysis. In the initial phase, we trained our models after altering the physical properties and observed the change in rewards generated using the same training algorithms. For the next phase we attempt to carry out transfer learning across different environment configurations and we ultimately explore the zero-shot capabilities of out trained model.

2 Methods

2.1 Deep Q-Networks (DQN)

Q-Learning is one of the famous reinforcement learning algorithms that approximates the optimal action value function independent of the policy being followed (Sutton and Barto, 2018). It is also referred to as off-policy TD control where the central idea is to find out the optimal behaviour for a given state. To encounter the tradeoff between exploration and exploitation, we utilize an ϵ-greedy policy. A DQN, or Deep Q-Network, approximates a

state-value function in a Q-Learning framework with a neural network. DQNs solve the problem of unstable learning by the following techniques explained below.

2.1.1 Experience Replay

Deep neural networks (DNN) easily overfit current episodes. An overfit network does not produce various experiences. Experience replay stores experiences including state transitions, rewards and actions, which are necessary to perform Q learning and makes mini-batches to update neural networks.

2.1.2 Target Network

In TD error calculation, target function is changed frequently with DNN. Unstable target function makes training difficult. Target Network technique fixes parameters of target function and replaces them with the latest network every few thousand steps.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a)]$$

2.2 Proximal Policy Optimization (PPO)

PPO is a policy gradient method where policy is updated explicitly. The loss function for policy gradient is given as:

$$\Delta J(\theta) = E_{\pi_\theta}[\Delta_\theta \log \pi_\theta(a_t|s_t) A_t]$$

where A_t is the Advantage function. The main challenge of vanilla policy gradient is the high gradient variance. Using an advantage function helps reduce this variance by estimating how good an action is compared to average action for a specific state. The advantage function is given as:

$$A(s, a) = Q(s, a) - V(s)$$

If the advantage function is positive, we increase probability of the corresponding action and if it is negative, reduce the probability of the action. PPO is a first order optimization. The probability ratio between the new policy and old policy is given by $r(\theta)$.

$$r(\theta) = \pi_\theta(a|s)/\pi_{\theta_{old}}(a|s)$$

The objective function can be modified to:

$$J(\theta) = E[r(\theta)A_{\theta_{old}}(s, a)]$$

Without adding constraints, this objective function can lead to instability or slow convergence rate due to large and small step size update respectively. For this, PPO imposes r(θ) to stay within a small interval around 1, i.e., interval between 1 - ϵ and 1 + ϵ.

$$J(\theta) = E[\min(r(\theta)A_{\theta_{old}}(s, a), clip(r(\theta), 1 - \epsilon, 1 + \epsilon)A_{\theta_{old}}(s, a))]$$

The clip function truncates the policy ratio between the range [1 - ϵ, 1 + ϵ]

2.3 Our general approach

There are 5 physical properties which may be altered for the environment

- Mass of cart
- Mass of pole
- Force applied
- Gravity
- Length of Pole

We divide our work into two phases:

- **Deep RL**: First, we implement a deep RL setup (Huang et al., 2022) for approximating the functions. We choose Q-Learning to proceed with our analysis. Naturally, we use a neural network to approximate our Q-function. To reduce our variance, we utilize both experience replay and a separate target network for simulating fixed targets. We alter the properties in a mutually exclusive manner and make observations for 5 distinct values for each property.
- **PPO**: We utilize this algorithm to implement a policy gradient framework (Huang et al., 2022) to explore transfer learning. It is via this algorithm that we understand the extent to which we are able to reuse our old policy for a zero shot transfer to a new scenario. Firstly, we try and train the network by switching the environment configurations in fixed intervals. Secondly, we train the network for three different configurations and make an attempt to generalize for the last two configurations.

Algorithm 1 The general setup

```
Construct approximator                    ▷ This is our neural network
loop
    Alter environment configuration
    Apply the algorithm
    Check episodic returns
    Restore configuration
end loop
```

2.4 State Space

This time we used value function approximation to handle the continuous state space. The methods of PPO and DQN don't require table based lookups. Hence we were able to avoid discretization and hence significantly reduced the variance observed in the episode returns.

The state space in the CartPole environment is represented by this 4-tuple

$$(position, velocity, angle, angular\_velocity)$$

2.5 Experimental Setup

Since our work deals with change in environment, the first step was to figure out the exact configurations of the environment so that we'd be able to verify our algorithms. We make an improvement over our last setup and we keep the values in an arithmetic progression with personalized difference calculations so that the we are able to identify the generalization capabilities better. This is because a change that is significant for one parameter might not be as significant for another. This can be later verified via our graphs where the degree of alteration shows the effect of the change (See Table 1 for exact values).

2.5.1 DQN

Here, we test how a DQN network learns a behaviour as the various attributes of the environment are varied. For this we run the algorithm on the Cartpole-v1 environment while varying an attribute for each run. The attributes and there values can be seen in table 1.

Gravity	Mass of Cart	Mass of Pole	Length of Pole	Force Magnitude
9.8	1	0.1	0.5	10
78.4	10	3.1	105.5	100
19.6	4	1.1	35.5	40
156.8	13	4.1	140.5	130
39.2	7	2.1	70.5	70

Table 1: Values assigned to attributes of the environment during experimentation phase.

Attribute	Value 1	Value 2
Gravity	156.8	39.2
Mass of Cart	13	7
Mass of Pole	4.1	2.1
Length of Pole	140.5	70.5
Force Magnitude	130	70

Table 2: Fixed configurations for zero-shot vs. Cold start comparisons

2.5.2 PPO

This is where we explored our main motive: change in behavior with change in properties of the environment. To facilitate a comparative analysis, we set up our evaluation in the following manner:

- We reused all 25 configurations for each parameter from the DQN evaluation (See Table 1).
- We fixed the configurations for which we wanted to generalize (See Table 2).
- We implemented our policy gradient algorithm and trained it using a neural network as an approximator.
- While training, we changed the configurations of the environment at fixed intervals of $\sim 40K$ steps (Chandak et al., 2019).
- Hence, by running it for $\sim 200K$ steps, we complete the simulation for all 5 configurations of a given parameter.
- However, we followed two different techniques for the last step:

1. **Cold start**: In this case, we keep training as usual and observe some dips during configuration change. This is mainly used for verifying if pre-training in a different configuration relaxes the training requirements / increases the accuracy for a new unseen setup.
2. **Zero shot evaluation**: Here we keep training for the first 3 configurations and stop learning once we reach the 4th and 5th configurations. This shows us how the model is able to generalize in a previously unseen environment.

2.6 Observations

2.6.1 DQN

It is observed that in most cases, difficult environments, eg: when the value of the attributes is changed to a very high value like 156.8 for gravity, etc. the agent reaches the lowest value of reward by the end of training. That is, when the environment is difficult the agent finds it difficult to learn. Also in most cases, over time the agent usually ends up reaching the max reward value (500) showing that given enough time to train, a DQN does a good job learning a good policy. (See Figure 2)

2.6.2 PPO and transfer learning

For the zero-shot and cold-start settings, we change our values in a zig-zag fashion[1] so that the changes made to the model are significant. This is because a monotonic increase/decrease proved to be a relatively easy task and models were observed to train almost normally.

We first observe our cold start based models and make the following observations:

- The sharp declines at steps $\sim 40K, \sim 80K, \sim 120K, \sim 160K$ indicate locations of configuration switch (See Figure 3).
- The difference in the magnitude of decline(In Figure 4, we find a sharp decline at 40K and a mild decrease at 80K) shows the significance of the respective changes.

[1] If the values in increasing order are $[1, 2, 3, 4, 5]$, initialize in the order $[1, 4, 2, 5, 3]$ to maximize the effect of changes

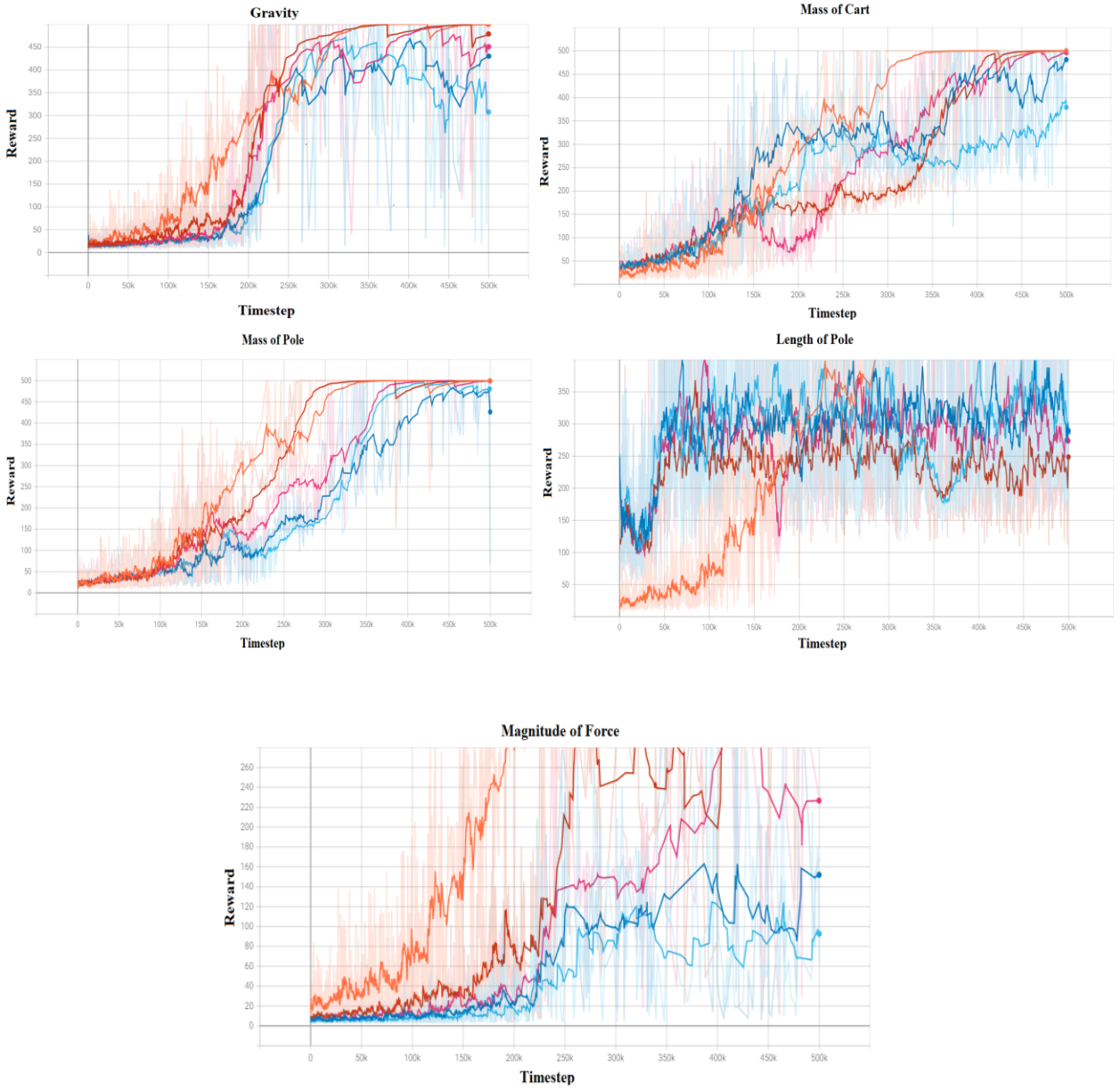

Figure 2: Rewards vs Time steps for runs of DQN while varying the various attributes of the environment. An attribute is changed in each graph with the attribute being varied mentioned on the graph.

A look at the graphs 5, 6, 7 beautifully depicts the difference between our setup. The graphs on the first column (to the left) are the ones which undergo training using proximal policy gradient and there are no drastic changes for returns. However, a glance at the graphs in the second and third columns reveals some sharp declines at steps $\sim 40K, \sim 80K, \sim 120K, \sim 160K$. These are the points where we change our environment configurations. As

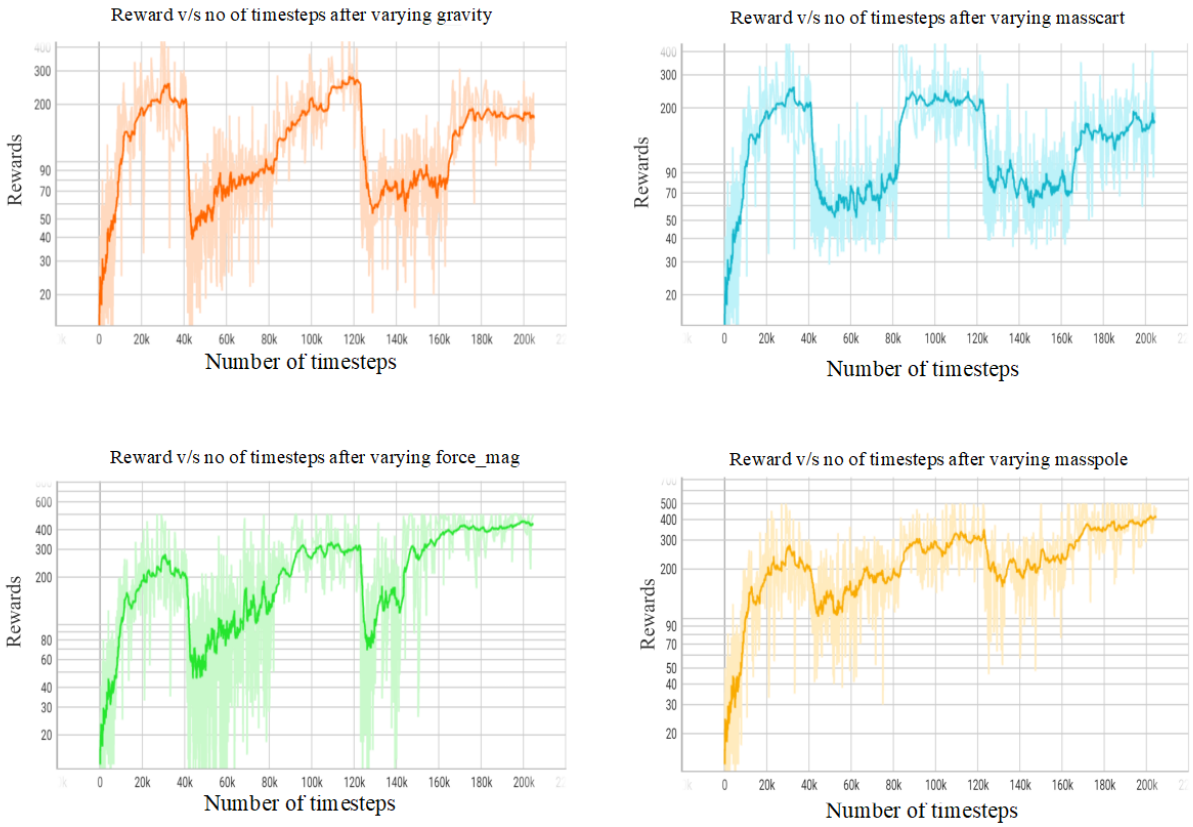

Figure 3: Transfer learning using PPO. Note the sharp declines at regular intervals

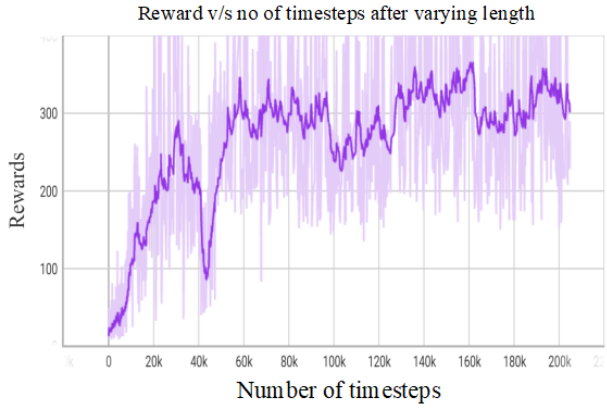

Figure 4: Rewards for transfer learning while training policy gradient with a dynamic environment. Changes are accommodated for one parameter at a time and happen at steps 40k, 80k, 120k and 160k. Note the sudden increase/decrease in the rewards owing to the changed environment. A change in policy via back-prop increases the rewards.

the algorithm trains after these sharp downfalls, we steadily keep improving. This varied

Attribute	**PPO**	**Cold Start**
Gravity (39.2)	**261**	232
Mass of Cart (7)	**229**	222
Mass of Pole (2.1)	275	**344**
Length of Pole (70.5)	279	**299**
Force Magnitude (70)	**395**	303

Table 3: Comparing average rewards in the last 40K steps for the vanilla PPO algorithm v/s the one augmented with transfer learning. Cold start performs better than PPO with lesser training under some circumstances

training makes our model robust and increases the generalization capabilities.

We can make two interesting comparisons:

- **Normal PPO v/s Cold-Start**: This shows the assistance given to the training via a cold start. Although the parameters are tuned for a different environment, the properties are still pretty similar. (See Figure 5, Gravity and Masscart) We find that the model learns to adapt to newer, previously unseen environments in lesser training steps. Interestingly, in some cases, we find the model to generalize even better than the vanilla policy gradient. (See Figure 6, Force_mag and Table 3)
- **Zero-Shot v/s Cold Start**: Here we find a slight difference in the generalization which is expected since cold start keeps training via back propagation. However, one must not underestimate the power of zero-shot transfer after training under different scenarios. It sometimes achieves even better accuracy than the vanilla policy gradient. (See Figure 6, Force_mag) We believe this happens because the configurations that it encountered previously for training required a lot more precision (with increased force). This led to a stronger performance in a relatively easier scenario.

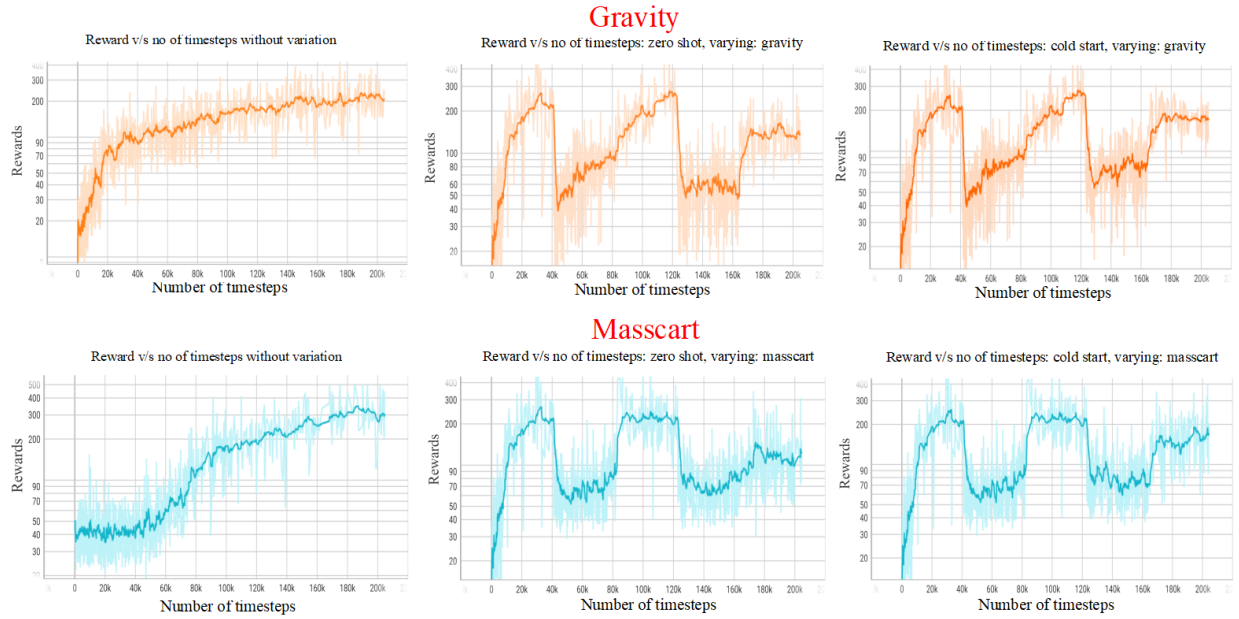

Figure 5: Transfer learning for gravity and masscart. Here we find the zero-shot and cold start values to be close, but lesser than the PPO based algorithm

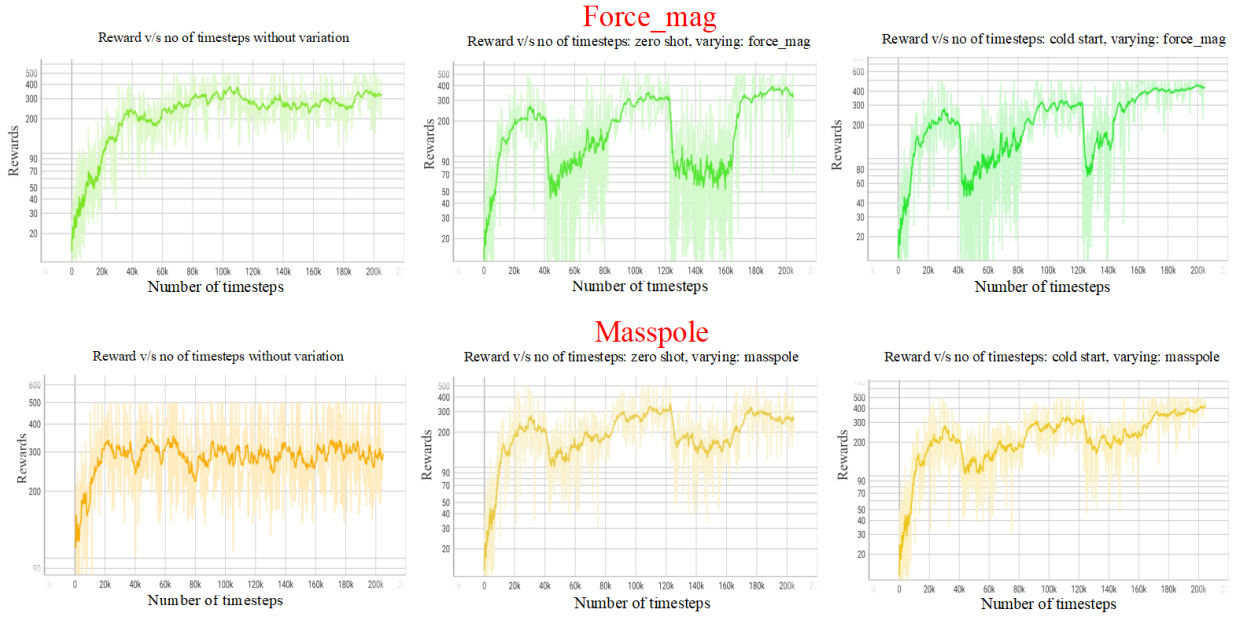

Figure 6: Transfer learning for force magnitude and mass of cart. Here we find the zero-shot and cold start values to be better than the PPO based algorithm.

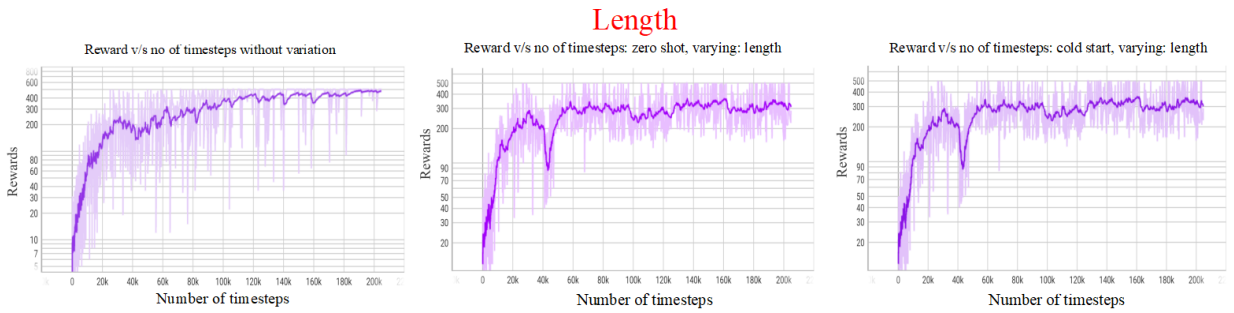

Figure 7: Transfer learning for length. Here, we find the zero-shot and cold start values to be close, but lesser than the PPO based algorithm

2.6.3 Conclusion

We extensively evaluate different RL algorithms right from tabular methods based on Bellman backups to value function approximation and advanced Policy Gradient methods on the CartPole-v1 environment. We find DQN to have a smoothening factor owing to experience replay. We witness transfer learning to be useful to generalize to new environments. In general, solving for tougher environments in the beginning may turn out to be helpful for models to do well under previously unseen environments provided a basic degree of similarity between the two settings.

References

Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J., and Zaremba W. (2016). *OpenAI Gym*.

Chandak Y., Theocharous G., Nota C., and Thomas P. S. (2019). *Lifelong Learning with a Changing Action Set*. DOI: 10.48550/ARXIV.1906.01770. URL: https://arxiv.org/abs/1906.01770.

Huang S., Dossa R. F. J., Ye C., Braga J., Chakraborty D., Mehta K., and Araújo J. G. (2022). "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms". *Journal of Machine Learning Research* 23.274, pp. 1–18. URL: http://jmlr.org/papers/v23/21-1342.html.

Sutton R. S. and Barto A. G. (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.