Pre-Training Deep Reinforcement Learning through Imitation in Discrete Action-Space Environments

Daniel Mistrik*
(dm743)

John Guo*
(jg843)

Abstract

Deep-Q-Networks have shown to have above-human-expert capabilities in certain settings but such optimal performance is often conditioned on a significant amount of training time limiting its practicality. One avenue of improving DQN efficiency is through pre-training and in this project we propose a new Imitation-based pre-training regimen in finite action-space environments, inspired by the successes of applying human preferences to deep reinforcement learning. Our approach yields a capable and efficient Imitation-based Q-learning framework although results show the improvements made to DQN are marginal which we attribute to their sensitivity to information-rich reward functions as well as rugged loss environments.

1 Introduction

Q-Learning[17] is a popular reinforcement algorithm that has been shown to work in both theoretical [18] and practical [8] settings where state and immediate reward are known but transitions are not. A key limitation to vanilla, tabular Q-learning is its inability to effectively infer the Q-value of neighbouring state-action pairs from training samples making it computationally ineffective to run on complex and/or high-dimensional environments. In the last decade, research in deep reinforcement learning has shown Deep Q-Networks (DQN) based on neural networks to be a good solution for high-dimensional input [11]. However, the success of DQN's has largely been attributed to large computational power and long training time [13] making them infeasible to be used in many practical applications with limited compute power.

There have been several solutions aimed at improving both overall performance and training efficiency by concentrating on the nature of Q-Learning [16, 6]. However, there has been less attention paid to the possibility of improving DQN performance by looking at it through the lens of neural network pre-training. Contemporary research argues that DQN pre-training is beneficial due to its ability in aiding learning of feature representations from high-dimensional input [4] but in this project we are interested in the possibility that even in lower-dimensional environments where the features are more or less provided, DQN's can benefit from pre-training due to it being able to provide a better initial weight in a jagged loss environment. Our inquiry is inspired by existing research showing the crucial role pre-training plays in producing successful learners [7].

This project is meant to be a general investigation into the effectiveness of DQN pre-training and therefore we chose Imitation Learning (IL) as our pre-training regimen under the assumption that most tasks that one would apply DQN's in, a human can execute at a 'expert' level. As in DQN, the pre-training approximates the optimal Q-function rather than training the optimal learner directly but will use a loss function inspired by RLHF [3] by treating every training sample and the action chosen as the 'preference' under a generalized cross-entropy loss. One can therefore view our approach as a special case of Inverse Reinforcement Learning with the caveat that we only pre-train the rewards function and proceed with the DQN 'tuning it' rather than just learning an optimal policy off of it.

*Equal Contribution

Preprint. Under review.

The main contribution of this project is the loss function's integration into pre-training and an analysis of DQN effectiveness with(out) pre-training in 'medium-dimensional' environments [2]. Thus we use only the most standard IL algorithm which is Behavioral Cloning (BC) [10] and use vanilla DQN.

1.1 Related Work

Improvements to DQN have long been an active area of research. Most approaches concentrate on altering DQN directly by addressing its outright issues like overestimation in the case of Double Q-Learning [16] or improved exploration with Noisy-Nets [5]. Our approach differs from these in that it leaves the DQN algorithm untouched, treating the problem more as neural network optimization where pre-training plays a significant role [7]. This makes our project similar to [4] which likewise pre-trains DQN with human preferences over a cross-entropy loss however their focus is on feature-learning in high-dimensional environments while we conjecture that even in an environment with the features provided, DQN's still struggle to effectively become optimal.

Integrating BC into Q-Learning has been shown to have significantly higher performance than vanilla DQN. This was the case for SQIL [14] which replaced TD error with squared soft Bellman error (L2 error with the target being the bellman equation for the Q-function modified with the log of a sum of exponentiated Q-functions over all the possible actions given the next state). This approach fundamentally alters the Q-learning algorithm and while it does require a finite action space, like ours, our approach is less complex and does not alter the Q-Learning algorithm.

Our pre-training step can also be considered as preference learning [19] but differs from most algorithms in that it is unconcerned with being able to determine a preference between two arbitrary trajectories. Rather it is interested in being able to determine a preference among all possible 1-length trajectories (that is the same state and all possible actions). The departure from binary preference and a strict setting of the trajectory length hyperparamter is what differs it from RLHF [3] as well.

2 Approach

2.1 Pre-Training

Due to the unconventional training objective, we are optimizing a Q-function rather than a policy directly, for IL, our approach is inspired by the work of [3] which itself is founded in preference learning [19]. One could therefore view our approach as a generalization of [3] to k trajectory segments for each human preference (the preference remains to choose the 'best' one) with a trajectory segment length of 1 where the human preference is taken implicity rather than explicitly.

The motivation behind our approach is the assumption that we can treat the human expert as an optimal player who operates on a latent Q-function. Thus in much the same way as [3] we can use $\hat{Q}$ to model the probability that the human expert chooses an action over another given state s:

$$\hat{P}(i > j|\ \hat{Q}, s) = \frac{exp(\hat{Q}(s,i))}{exp(\hat{Q}(s,i)) + exp(\hat{Q}(s,j))} \quad [1]$$

To apply this to our multi-trajectory setting we make the observation that we only care about the optimal action being picked over the un-optimal ones. In other words, we do not care about which action is the 2nd or 3rd most optimal and thus only care about $\hat{P}(i > j|s)$ when i is the optimal choice. Therefore our overall cross-entropy loss is:

$$\mathcal{L}(\hat{Q}|\ a^*, s) = \sum_{a \in \mathcal{A}: a \neq a^*} \hat{P}(a^* > a|\ \hat{Q}, s)$$

As our loss-function is over a single state-action pair, it allows us to use every sample collected from a human expert as a single preference (as compared to an entire trajectory segment as in [3, 19]). This allows for human demonstrations to be implicitly treated as feedback on the latent Q-function $\hat{Q}$ we try to approximate enabling very fast data-collection (as a reference our project used 10,000 samples for each environment and data collection took 10 minutes).

[2] We define 'medium-dimensional' environments slightly arbitrarily as an environment with a multi-dimensional state that is greater than 2 dimensions but also smaller than an image (roughly 255 dimensions)

2.2 DQN

As mentioned in the introduction, we use the vanilla DQN implementation with none of the improvements discussed above. The reason is that we wanted to see the isolated effects of imitation pre-training on DQN. A discussion on the setting of hyperparameters is in the appendix.

2.3 Environments & Set-Up

We used 3 environments for this project, all sourced from the Gym environment [2] and one of its environment extensions, box2d. Due to the constraints of our pre-training step all the environments have a discrete and finite action space, and a real 'medium-dimensional' state. We proceed with a short description of each of the environments used and their accompanying reward functions:

1. MountainCar-v0:
   Mountain car is a deterministic MDP of a car trying to scale a mountain [12]. The car is stochastic-ally initialized somewhere in a valley between two mountains (the right one being the goal). The state is a 2-dimensional real-valued vector which describes the x-position of the car and its current velocity. The action-space is composed of 3 possible actions: accelerate to the left, accelerate to the right, don't accelerate at all. The reward function returns -1 in all states. The episode length is 200 but is cut short if you reach the goal state.
2. LunarLander-v2:
   LunarLander is a physics-based environment where your aim is to land a spaceship on the landing pad on the surface of the moon. The state is an 8-dimensional real-valued vector describing your position, velocity, angular velocity and whether or not your landing legs are touching the surface. The action-space is composed of 4 possible actions which are either which of your 3 engines (main, left or right engine) you should fire or whether you should fire none of them. The reward function is complex and we recommend you consult the documentation for a detailed description but in general is higher if the spaceship approaches the landing pad, slowly and at no angle. The episode terminates once the lunar lander either crashes, lands safely or goes out of view.
3. Acrobot-v1:
   Acrobot is composed of two links formed in a chain with one joint fixed and the other actuated with the goal being to get the tip of the link connected to the actuated joint above a certain height [15]. The observation space is a 6-dimensional real-valued vector describing the angle and angular velocity of the two joints. The action-space is composed of 3 actions: apply 1 torque to the actuated joint, apply -1 torque to the actuated joint or apply no torque. The reward function is -1 for all states. The episode has length 500 but can end abruptly once your actuated link is above the height threshold.

3 Results

3.1 Pre-Training

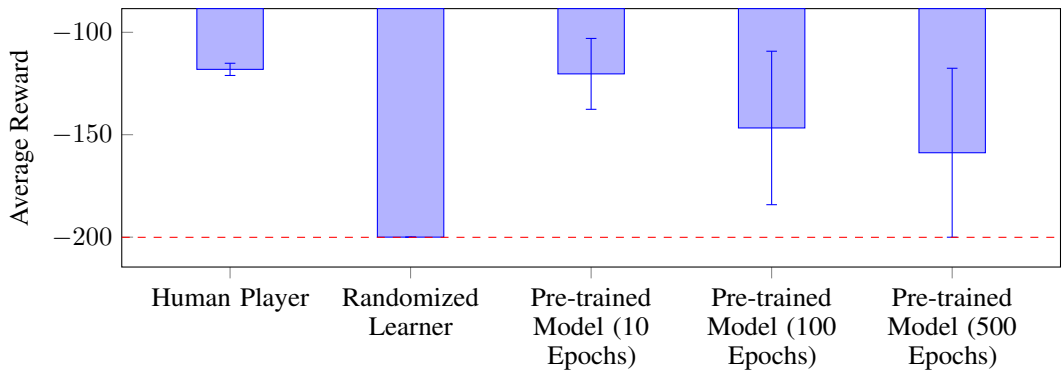

Figure 1: Average rewards & standard dev. for mountaincar-v0 (red line is minimum possible reward)

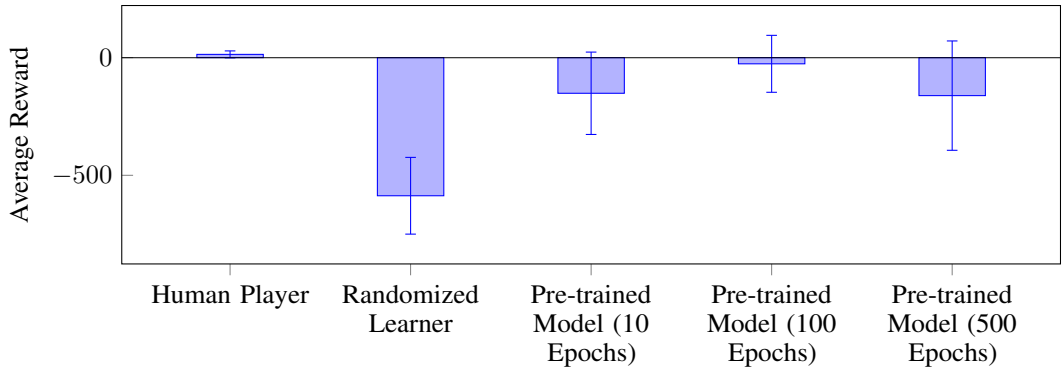

Figure 2: Average rewards & standard dev. for lunarlander-v2

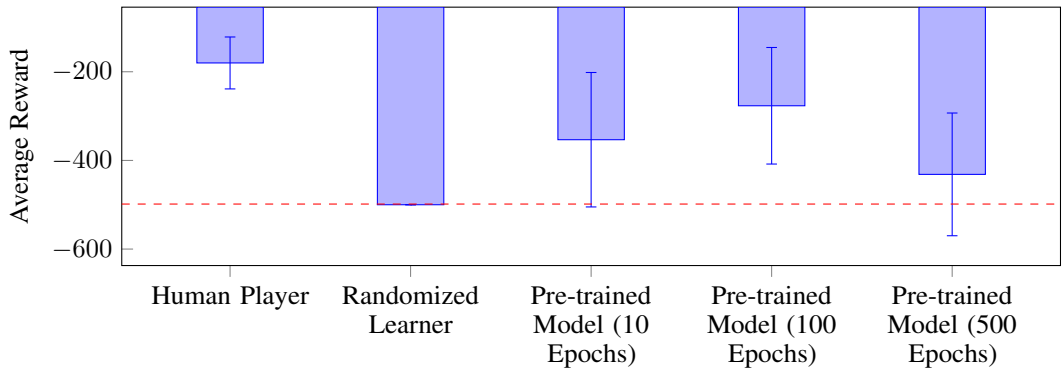

Figure 3: Average rewards & standard dev. for acrobot-v1 (red line is minimum possible reward)

From Figures 1-3 we see that every pre-training model improves on the performance of the randomized learner showing meaningful learning has occured in each instance. For every environment there is also a hyperparameter tuning (10 or 100 pre-training epochs) for pre-training that approaches human player performance.

For LunarLander-v2 and Acrobot-v1 the most optimal epoch length was 100 while for MountainCar-v0 it was 10. A possible explanation for this could be the fact that MountainCar-v0 is a deterministic MDP and there is a far 'simpler' problem to master than the real-world physics problems that are LunarLander-v2 and Acrobot-v1. While they differ on most optimal epoch length, all environments share the worst performing epoch length which is 500. This can be rationalized as possible over-fitting on the expert trajectories exacerbating the distributional shift that BC suffers from.

Another shared phenomenon is that the pre-trained models all have a significantly higher variance than the human player (due to the randomized learner often getting the minimum possible reward we are unable to compare them). This can be again explained by distributional shift as well as the good performance causing higher variance under the bias-variance trade-off. Among hyperparameter tuning, there isn't a clear trend in relation to variance but one can observe a slightly smaller variance in the optimal hyperparameter tuning compared to the others.

The pre-training performs best relative to the expert in MountainCar-v0 with near 'optimal' performance. This is most likely due to the simplicity of the problem itself as discussed above. More interestingly, the worst performance relative to the human expert is in Acrobot-v1 even though its action space and observation space is smaller than LunarLander-v2 (where the best pre-trained model performs better relative to the human expert). An interesting explanation for this could be that this problem was also the most difficult one to master by the human expert implying a connection between the difficulty to learn the problem by the human expert and the difficulty to be able to mimic the expert's demonstrations by the neural network learner.

3.2 Pre-Training + DQN

Figure 4: Performance Graph (+ s.d.) for MountainCar-v0

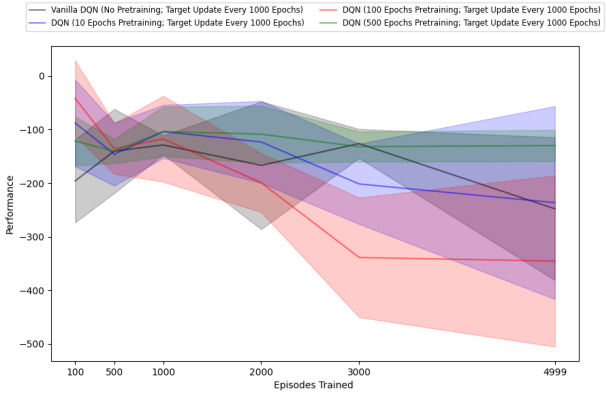

Figure 5: Performance Graph (+ s.d.) for LunarLander-v2

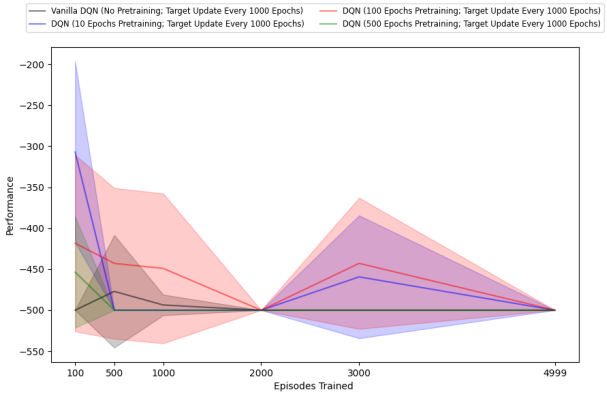

Figure 6: Performance Graph (+ s.d.) for Acrobot-v1

Figures 3-6 show that while the pre-trained DQN models all out-perform vanilla DQN when the number of episodes trained on is small (<1000) this advantage becomes marginal to non-existant as we increase the number of episodes. We also observe a large differences between the performance of the pre-trained models based on the various number of pre-trained epochs.

The disappointing performance of imitation pre-training has a number of possible explanations. For Acrobot-v1 and MountainCar-v0 we observe that Vanilla DQN performs very poorly (almost always getting the minimum possible reward) and the reason for this could be that the reward functions are very information-poor. By this we mean that the rewards don't provide any information on whether the state and/or action in the correct direction as no matter the state-action pair the reward will always be -1. This is a problem for DQN which samples individual state-action pairs from the buffer with no long-term trajectory that could provide context on the optimality of a provided action (i.e. we don't know which actions could get us to an endstate except for the Q-function which because of the constant negative reward is continually de-valued).

Due to the fact that the Q-value for a state-action pair will generally decrease every time its sampled due to the constant negative reward we introduce pessimism to sampled state-action pairs which include the human expert's trajectory. This leads to a preference for less-sampled or not visited state-action pairs as their Q-value will not be subtracted like the visited pairs are leading the DQN to encourage the learner to deviate from the expert's path worsening the distributional shift that affects BC. This is supported by the fact that as we sample more episodes (and thus continually down-weight the experts state-action Q-values) the benefit for pre-trained DQN models over vanilla DQN decreases as the learner appears in more and more states the pre-trained model never visited (and thus any benefits to pre-training are lost).

Unlike MountainCar-v0 and Acrobot-v1, LunarLander-v2 has informative rewards. However, like the above, the pre-training performs poorly which we attribute to an overly rugged loss environment. This explains why any initial benefit to pre-training is immediately lost as well as the very inconsistent performance curve after. As our loss environment is rugged gradient descent can overshoot local minima ending in even higher loss areas making optimization general and, as the pre-trained starting state is itself in a rugged environment, any overshooting can eliminate the benefits of pre-training.

To address this we could reduce the learning rate for DQN treating it more as a fine-tuning step rather than the main body of training (in our experiment the learning rate was fixed for all the models). Another possible issue was raised in [4] who observed that pre-training performed poorly when the output layer wasn't normalized due to a tendency to produce very-large weights. A limited analysis into our training failed to find this phenomena (of very large weights in the output layer) in our pre-trained models but we don't deny it could be one of the causes of poor performance.

4 Conclusion

Our work has shown that Imitation pre-training has little effect in improving vanilla DQN in discrete action spaced, medium-dimensional environments. This implies the fundamental limitations of DQN are not addressed by pre-training or are due to a very rugged loss environment (which would reduce any benefits of pre-training) or a need for specific hyperparameter tuning that was outside the scope of this project.

We have however, shown that the our preference-imitation pre-training by itself provides an efficient and capable learner even in difficult and information-poor reward settings (an example of the benefits of IL in settings with bad reward functions). Further work in this direction could include a rigorous comparison of our imitation learning approach compared to well-known baselines like Curriculum Offline Learning [9] or RLHF [3].

5 Division of Work

This project was completed by Daniel Mistrik and John Guo, both of whom contributed equally to it. Daniel Mistrik was in charge of conceiving, implementing and testing the pre-training portion while John Guo was in charge of implementing and testing the DQN implementation as well as setting up the software infrastructure for this project and doing a majority of the overall testing. The paper was written by Daniel Mistrik with reviews, corrections and annotations by John Guo while the video script was written by John Guo who also did the editing and submission for it.

References

[1] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[4] G. V. De la Cruz Jr, Y. Du, and M. E. Taylor. Pre-training neural networks with human demonstrations for deep reinforcement learning. *arXiv preprint arXiv:1709.04083*, 2019.

[5] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration, 2019.

[6] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

[7] Geoffrey E. Hinton, Simon Osindero, and Yee W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[8] F. Kirchner. Q-learning of complex behaviours on a six-legged walking machine. In *Proceedings Second EUROMICRO Workshop on Advanced Mobile Robots*, pages 51–58, 1997.

[9] Minghuan Liu, Hanye Zhao, Zhengyu Yang, Jian Shen, Weinan Zhang, Li Zhao, and Tie-Yan Liu. Curriculum offline imitation learning, 2022.

[10] Donald Michie, Michael Bain, and Jean Hayes-Michie. Cognitive models from subcognitive skills. *IEE control engineering series*, 44:71–99, 1990.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[12] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

[13] R. Pina, H. Tibebu, J. Hook, V. De Silva, and A. Kondoz. Overcoming challenges of applying reinforcement learning for intelligent vehicle control. *Sensors*, 21(23):7829, 2021.

[14] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards, 2019.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[16] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[17] Christopher Watkins. Learning from delayed rewards. 01 1989.

[18] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. 1992.

[19] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

Appendix

An Appendix is either provided in the following pages or if this is the last page, please reach out to the author's for a copy.

A Training Regimen

All of our code and models are available on our github repository(Might be private until the submission window has ended to prevent AI violations)

Unless otherwise stated, all our neural network architectures were composed of two hidden layers each of 10-times the size of the input-layer (which is equal to the state dimensions) with the output layer corresponding to the value of the Q-function given the input as state and the index as action (and thus has a dimension equal to the number of actions).

A.1 Pre-Training

Our dataset of 10,000 samples was collected by a one human player per environment. Data collection was done in one sitting and took a little over 10 minutes.

Training was over either 10,100 or 500 epochs with a constant learning rate of 0.001 using the Adam optimizer.

A.2 DQN

B Empirical Results

Unless otherwise stated, all results are the average reward over 10 episodes ran after the respective model was trained.

B.1 Pre-training

Table 1: Peformance on MountainCar-v0

Model	Epochs	Average Reward
Human Player	N/A	-118.1 ± 2.98
Randomly-Initialized Model (No-Training)	N/A	-200.0 ± 0.0
Pretrained Model	10	-120.3 ± 17.30
Pretrained Model	100	-146.7 ± 37.46
Pretrained Model	500	-158.8 ± 41.24

Table 2: Peformance on Acrobot-v1

Model	Epochs	Average Reward
Human Player	N/A	-180.1 ± 58.59
Randomly-Initialized Model (No-Training)	N/A	-500.0 ± 0.0
Pretrained Model	10	-353.3 ± 151.66
Pretrained Model	100	-276.7 ± 131.48
Pretrained Model	500	-431.5 ± 138.44

Table 3: Peformance on LunarLander-v2

Model	Epochs	Average Reward
Human Player	N/A	13.84 ± 14.75
Randomly-Initialized Model (No-Training)	N/A	-586.94 ± 163.04
Pretrained Model	10	-151.46 ± 175.01
Pretrained Model	100	-26.26 ± 120.98
Pretrained Model	500	-161.43 ± 232.33

B.2 DQN Grid-Search

B.2.1 MountainCar-v0

Table 4: Grid Search on 0 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
500	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
1000	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
2000	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
3000	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
4999	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0

Table 5: Grid Search on 10 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-113.8 ± 1.99	-109.5 ± 16.35	-155.3 ± 41.71
500	-129.4 ± 26.76	-108.1 ± 16.85	-114.8 ± 23.61
1000	-133.9 ± 28.85	-114.6 ± 26.76	-108.3 ± 0.46
2000	-168.9 ± 25.46	-142.6 ± 19.74	-133.1 ± 22.30
3000	-161.1 ± 24.43	-105.4 ± 1.2	-128.1 ± 32.52
4999	-200.0 ± 0.0	-147.1 ± 43.23	-113.7 ± 2.69

Table 6: Grid Search on 100 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-156.8 ± 45.78	-133.9 ± 26.64	-155.0 ± 38.51
500	-143.6 ± 49.45	-128.0 ± 32.28	-137.7 ± 42.97
1000	-149.0 ± 3.46	-154.2 ± 44.02	-105.3 ± 5.12
2000	-143.3 ± 4.52	-195.3 ± 10.57	-129.2 ± 28.75
3000	-141.9 ± 3.08	-188.1 ± 25.05	-127.8 ± 32.84
4999	-176.2 ± 23.61	-193.5 ± 8.24	-200.0 ± 0.0

Table 7: Grid Search on 500 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-163.6 ± 37.20	-161.1 ± 39.03	-175.4 ± 33.54
500	-200.0 ± 0.0	-200.0 ± 0.0	-188.8 ± 17.11
1000	-200.0 ± 0.0	-200.0 ± 0.0	-199.8 ± 0.4
2000	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
3000	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0
4999	-200.0 ± 0.0	-200.0 ± 0.0	-200.0 ± 0.0

B.2.2 LunarLander-v2

Table 8: Grid Search on 0 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-364.48 ± 66.13	-550.20 ± 149.99	-195.95 ± 77.51
500	-115.57 ± 18.60	-147.35 ± 31.65	-140.28 ± 78.55
1000	-131.35 ± 25.40	-146.83 ± 33.89	-128.80 ± 17.61
2000	-118.25 ± 40.57	-151.89 ± 66.75	-166.88 ± 119.21
3000	-151.60 ± 27.01	-114.98 ± 54.98	-126.22 ± 27.09
4999	-143.24 ± 28.72	-134.56 ± 47.36	-247.95 ± 133.31

Table 9: Grid Search on 10 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-197.28 ± 140.31	-122.64 ± 71.49	-87.74 ± 80.87
500	-166.37 ± 84.66	-216.19 ± 61.13	-146.07 ± 58.86
1000	-125.92 ± 54.14	-91.69 ± 33.45	-103.90 ± 49.56
2000	-316.65 ± 116.14	-219.44 ± 91.93	-123.30 ± 76.45
3000	-353.07 ± 130.51	-305.29 ± 73.68	-201.58 ± 74.95
4999	-355.87 ± 57.51	-302.54 ± 88.05	-236.54 ± 180.04

Table 10: Grid Search on 100 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-186.47 ± 102.30	-209.20 ± 126.91	-41.68 ± 70.65
500	-194.46 ± 106.66	-239.00 ± 130.97	-135.76 ± 47.23
1000	-268.18 ± 69.58	-256.22 ± 133.58	-117.55 ± 80.13
2000	-243.60 ± 77.07	-176.53 ± 65.53	-199.31 ± 54.95
3000	-423.86 ± 116.56	-315.94 ± 59.15	-338.78 ± 111.59
4999	-813.98 ± 121.73	-301.35 ± 104.78	-345.56 ± 159.71

Table 11: Grid Search on 500 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-128.28 ± 26.89	-104.16 ± 42.94	-121.47 ± 45.43
500	-118.30 ± 25.10	-124.40 ± 23.02	-140.55 ± 22.60
1000	-119.94 ± 47.40	-134.28 ± 26.45	-103.80 ± 46.32
2000	-139.30 ± 19.52	-139.76 ± 49.32	-108.75 ± 53.14
3000	-105.56 ± 58.48	-122.14 ± 50.54	-132.09 ± 28.06
4999	-139.91 ± 24.54	-114.33 ± 36.86	-129.96 ± 29.29

B.2.3 Acrobot-v1

Table 12: Grid Search on 0 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
500	-500.0 ± 0.0	-500.0 ± 0.0	-477.1 ± 68.7
1000	-500.0 ± 0.0	-500.0 ± 0.0	-493.7 ± 12.6
2000	-177.2 ± 109.2	-500.0 ± 0.0	-500.0 ± 0.0
3000	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
4999	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0

Table 13: Grid Search on 10 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-500.0 ± 0.0	-500.0 ± 0.0	-307.0 ± 110.56
500	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
1000	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
2000	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
3000	-500.0 ± 0.0	-500.0 ± 0.0	-459.4 ± 74.98
4999	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0

Table 14: Grid Search on 100 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-500.0 ± 0.0	-398.9 ± 132.82	-418.3 ± 107.7
500	495.0 ± 15.0	-500.0 ± 0.0	-442.9 ± 92.0
1000	-500.0 ± 0.0	-500.0 ± 0.0	-449.1 ± 91.4
2000	-500.0 ± 0.0	-487.7 ± 36.9	-500.0 ± 0.0
3000	-500.0 ± 0.0	-490.7 ± 27.90	-442.9 ± 80.0
4999	-500.0 ± 0.0	-492.2 ± 23.4	-500.0 ± 0.0

Table 15: Grid Search on 500 pre-train epochs

Episodes trained on	Target Update Frequency (Episodes)		
	100	500	1000
100	-500.0 ± 0.0	-450.0 ± 66.56	-453.6 ± 67.9
500	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
1000	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
2000	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
3000	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0
4999	-500.0 ± 0.0	-500.0 ± 0.0	-500.0 ± 0.0