Implementation and Comparison of Latest Improvements in Deep Reinforcement Learning on Cartpole Environment

Apoorv Garg
Virginia Tech
apoorvgarg@vt.edu

Kumar Sai Bondada
Virginia Tech
kumarsaibondada@vt.edu

Neil Gutkin
Virginia Tech
neilg99@vt.edu

Abstract

The recent advances in Deep Q Networks have led to the development of an optimal policy and an optimal Q network by borrowing the concepts from deep learning. For example, the algorithm provides an optimal action plan for reward maximization in the CartPole problem using images from the environment. However, this is an initial work and faces several known limitations and several improvements have been made over this like Double Deep Q Networks, Actor-Critic Systems, Prioritized Experience Replay, Dueling Networks and Multi-step Learning. All these algorithms target a different class of environments and tackle different problems in enviroments like large action spaces, noisy updates, slow learning etc. This work aims at implementing these algorithms and studying the impact of these improvements on a common environment(CartPole-Images problem) with almost similar hyperparameters (model comlexity, epsilon greedy exploration-exploitation, learning rate, target update rate) and study the impact when these improvements are combined together.

1. Introduction

The idea of reinforcement learning is to learn an optimum policy for an agent which learns from its mistakes and improves its performance. The agent performs this by performing an action in an environment and maximizes its cumulative reward. There are two major class of algorithms: Policy gradients and Q-Learning for the agents to converge to an optimum policy. [1]

1.1. Policy Gradients

The policy gradient methods target at modelling and optimizing the policy directly. This policy is parametrized using parameters θ and is notated as $\pi_\theta(a|s)$. In this case, the gradient of total reward function is defined as:

$$\nabla_\theta J(\theta) = \mathbb{E}[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] \quad (1)$$

[1]The code for this project is available at Github

There are a variety of algorithms like REINFORCE, Actor-Critic, Off-Policy Policy Gradient, A3C [8], A2C[8], DPG[14], DDPG[7], TRPO[12], PPO[13] which perform extensions of this technique to deep and reinforcement learning versions. We will attempt at visualizing the impact of deep Actor-Critic algorithms on performance for the CartPole Environment.

1.2. Q Learning

This technique aims at optimizing a state-action value function(the Q-function) and then extracting the optimum policy at each state using an argmax of this optimized Q function. An optimal Q function satisfies the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma max_{a'} Q_i(s',a')] \quad (2)$$

This converges to the optimal solution with i i.e. $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$.

1.3. Deep Q Learning

Recently, [9] proposed the use of deep neural networks to solve the Q learning for reinforcement learning environments like Atari. They proposed the use of a neural network to parameterize the Q function with θ i.e. $Q(s,a,;\theta) \approx Q^*(s,a)$. This is performed by minimizing the following loss function at each step i:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \in \rho(.)}[(y_i - Q(s,a;\theta_i)^2)] \quad (3)$$

where,

$$y_i = r + \gamma max_{a'} Q(s',a';\theta_{i-1}) \quad (4)$$

Here, y_i is the temporal difference target, $y_i - Q$ is TD error, θ_{i-1} is the target model which is updated with a slower rate, ρ is the environment distribution representing the transition in states depending on actions and the procured rewards. The choice of action a' is usually an $\epsilon - greedy$ policy to encourage a mixture of exploration and exploitation to visit model states. This makes DQN an off-policy algorithm. The greedy(optimal) action is chosen with a probability of

$1-\epsilon$, and a random action is chosen with a probability of ϵ. The authors also propose use of a Replay Buffer, which is just a circular array to store transitions. The parameterized Q function is trained after each step in the episode using a randomly sampled batch of transitions drawn from this Replay Buffer, and (Equation 3) is used to compute the loss from this mini-batch.

2. Methods

2.1. Environments

Cartpole-v0 will be used as the environment with some major modifications. Cartpole is an inverted pendulum problem, where the centre of the pole is hinged and the hinge can be moved left or right by the agent, and hence the action space lies in a two-dimensional discrete vector space. The counter movement of the hinge negates the angular velocity of the pole caused due to gravity, and the pole stays in an erect position. The default environment outputs the cart position, velocity, pole position and pole velocity at tip as the state information and a numeric reward of +1 when the pole is within the specified coordinates, and the done state instead. This default environment will be used while evaluating the policy gradient algorithm variations. However, for the DQN variants a more complex state space will be used. The state will be a combination of 4 images (preprocessed by resizing, binarizing after conversion to grayscale format, and normalizing)with temporal variation stored in the channel dimension(the latest image on the top). It is assumed that if the agent generates a reward of 195.0 averaged over 100 episodes, the game is assumed to be solved.

2.2. Establishment of a DQN Baseline

A baseline performance is first established using the vanilla DQN with slight modifications. Training with a batch size of 128 is performed in every step during a single episode. A Mean Squared Error loss is used, and ADAM optimizer is used with a learning rate of 3e-5. It was observed that the parameter updates on taking a step in the optimizer were very high values. Gradient clamping is used as a trick to solve this and the gradients are clamped between a value of -1 and +1. An $\epsilon - greedy$ strategy to pick actions is chosen to encourage exploration towards the beginning of training and exploitation towards the end. An exponentially decaying epsilon is used with a minimum epsilon value of 0.01 and a decay after every episode with a factor of 1e-3. The variation of epsilon with episode length is shown in Figure 1. A replay buffer of size 10000 entries is chosen, and the Q model is updated into the target model after every 10 episodes. The model contains 3 convolution layers and 3 batch normalization layers [5] outputting 64 channels. The hidden layer is then flattened and 4 linear layers are employed with a hidden size of (512, 256, 64 and 2) respectively.

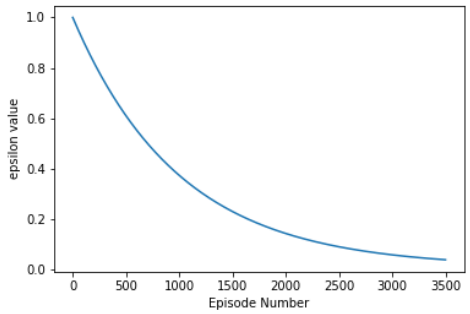

Figure 1: Exploration-Exploitation: Variation of epsilon with episode length

2.3. Double Q Learning (DDQN): Improvement in Training Loss Design

Equation 4 in Deep Q Networks provides the estimate of the target to the DNN parameterized Q value. This involves a term $max_{a'}Q(s', a'; \theta_{i-1})$ which calculates the maximum value over the estimated target network. This systematic overestimation introduces a maximization bias in learning. This can particularly be problematic since Q learning involves bootstrapping i.e. the target model is a copy of the Q-Network.[15] propose decoupling the maximization by implementing an argmax over the Q-network to propose an action at which the target model is evaluated to propose an estimate label for the Q-model. This also counters the effect of environment noise. TO realize DQN, Equation 4 can be updated to:

$$y_i = r + \gamma Q(s', argmax_a Q(s', a; \theta_i); \theta_{i-1}) \quad (5)$$

2.4. Prioritized Experience Replay: Improvement in Replay Memory Design

Prioritized Experience Replay impacts sample efficiency during learning a DQN by performing changes on the Experience Replay[11]. The algorithm provides priority(δ) to some samples based on the TD-error of the sample and draws random batch from the Experience Replay Memory weighted by this probability. The TD error and Probability equations for the Double DQN variant of PER are given by the following equations:

$$\delta_i = r + \gamma Q(s', argmax_a Q(s', a; \theta_i); \theta_{i-1}) - Q(s, a; \theta) \quad (6)$$

$$p_i = \delta_i + \epsilon \quad (7)$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (8)$$

However, this defies the Independent and Identically Distributed assumption, and hence the updates in the final parameters are weighted according to the choice of sample drawn. This process is called importance sampling.

$$w_i = \left(\frac{1}{N}.\frac{1}{P(i)}\right)^{\beta} \tag{9}$$

2.5. Dueling Deep Q Networks: Improvement in DNN Design

Since, we are moving to a higher dimensional state space after switching to images as representation of state space, some states are not as valuable. The dueling network [16] does not need to learn the effect of each action on every state value and hence in particularly important for this problem. The model modifies the DNN to predict the state value function which is independent of the action and the advantage, and then calculate the Q value using the following equation:

$$Q(s,a,\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right) \tag{10}$$

Here θ is the common DNN, α represents the parameters of the extra linear layer for advantage computation, and β represents the parameters of the extra linear layer for Value Function computation.

2.6. Multi-Step DQN: Improvement in Reward Design

Q-learning accumulates a single reward and then uses the greedy action at the next step to bootstrap. Alternatively, Multi-Step Learning[3] accumulates rewards of n steps and use that as rewards to speed up training. The updates rewards are given by:

$$R_t^n \approx \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \tag{11}$$

This reward is substitutes in equation 5 to compute the loss while training.

2.7. Noisy Nets: Improvement in Exploration Policy Design

[1] come up with an alternative to epsilon greedy exploration strategy by adding Gaussian noise to the fully connected DNN. This is implemented in Tensorflow using a noisy linear layer instead if the linear layer.

2.8. Rainbow: Combination of the above discussed techniques in a single agent

The idea behind this project is to combine all these algorithms and learn the effectiveness of the learned agent with and without the above algorithms on the modified Cartpole environment. This rainbow agent [4] will contain a package of DDQN+PER+Multi-step learning+Dueling Networks, DDQN+PER+Dueling Networks, DDQN+PER, DDQN and compare the results with the DQN baseline.

2.9. Policy Gradient Methods: Actor Critic

Two different Multi Layered Perceptrons are designed in the implementation of Actor-Critic algorithms. The value function network critiques the current state and the policy network is the actor which is updated in the direction critic tells. Note that this algorithm is developed using the pole position and velocity values, and the cart position and velocity values as the state space representation. The implementation os for Actor Critic with TD-0 learning.

3. Experiment Results

Table 1 represents the peak performance of each algorithm, and the number of frames processed by the algorithm during training phase. Figure 2 represents the performance of the baseline DQN algorithm per episode. It is worth noting that the performance drops after certain number of episodes. This is because of model overfitting [2] since the number of frames processed is very high, as visible in Table 1. It was also observed that manually thresholding the images led to much improved performance and higher rewards. Double DQN renders an improved performance with a maximum reward of 139 since it contributes to solving the overestimation problem along with gradient clamping as demonstrated in Figure 3. However, as shown in Figure 4, Prioritized Experience Replay stacked with Double DQN solves the environment in 2500 episodes. It is worth noting that value of β in the implementation of PER is linearly scaled from a minimum value of 0.4 to 1 depending on the batch processed. This $\beta - scaling$ renders much better performance for PER. After processing 10000 batches, the value stays at 1. The alpha value is set to 0.6 for PER as advised by the authors of the algorithm. On stacking Dueling Networks(Figure 5) on the previous iterations leads to more performance gains and solves the environment in 1900 episodes. Reward accumulation for 3 steps is found to be optimal for multi-step learning. It was also observed that multi-step learning is a major upgrade on the DQN, but has some incompatibility issues when integrated with the rainbow agent.[3] It was observed that Multi-step learning is the third rainbow agent to solve the Cartpole-images problem (Figure 6).

[2] More details in section 4.1

[3] More details are discussed in section 4.5

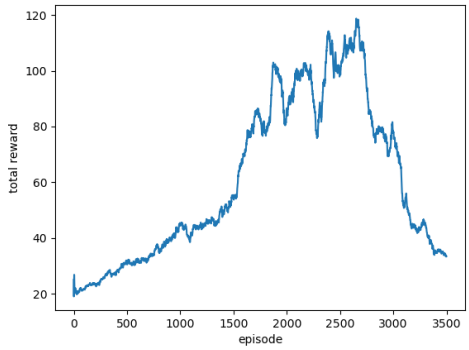

Figure 2: Baseline: DQN Performance (Rewards averaged over 100 episodes)

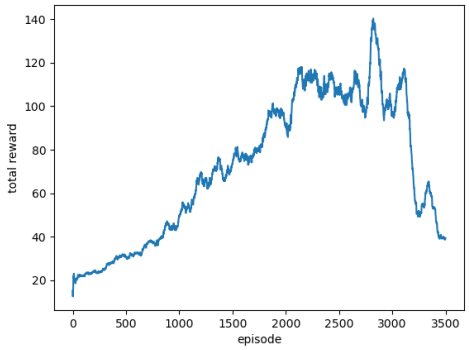

Figure 3: DDQN Performance (Rewards averaged over 100 episodes)

3.1. Training Hardware, Software Details and Hyperparameters

PyTorch is primarily used as the framework to exploit the DNN model creation, training and optimization. The models are trained on Google Cloud Platform on a hardware with 16 GB RAM, 4 vCPUs(Intel Xeon), 1 T4 GPU and 16 GB GPU memory, which enables the maximum batch size of 128, and maximum replay buffer size of 10000. OpenAI Gym is used to simulate the CartPole-V0 environment and the maximum reward output of the environment is set to 500 instead of 200.

The choice of hyperparmeters for DNN initialization was observed to be very important. The optimal hyper parameters are: all CNN layers are initialized with Kaiming initialization with $fan - out$ mode, and Linear layers are initialized with normal initialization with a standard deviation of 1e-3. The choice of decay in the $\epsilon - greedy$ policy is very important. An exponential decay rate is much more effective than polynomial decay rate. The minimum and maximum bounds of decay are also very important, and decay rate of 1000 per episode is found to be optimal for training performance convergence. It is also observed that higher learning rate for Adam optimizer[6] breaks training, and a learning rate of 3e-5 is optimal. Different loss functions (smooth L1 loss, Huber Loss and MSE Loss) were tried, and the MSE loss rendered the most optimal results.

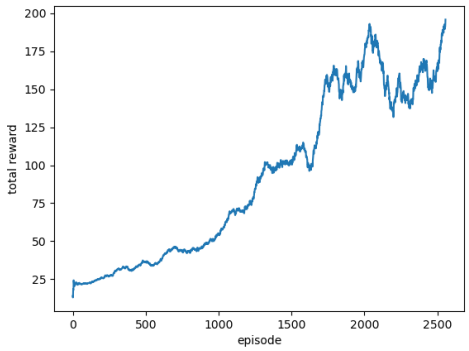

Figure 4: $PER + DDQN$ Performance (Rewards averaged over 100 episodes)

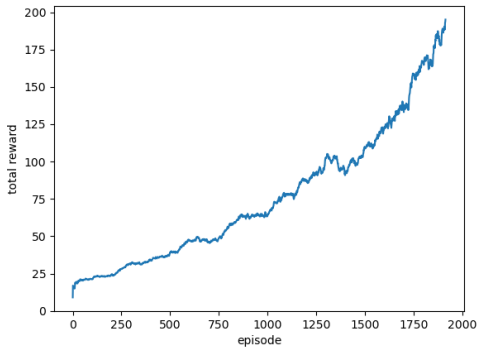

Figure 5: $PER + DDQN + Dueling$ Performance (Rewards averaged over 100 episodes)

4. Discussion

4.1. Sudden Drop 'Overfitting' on Excessive Training

The authors of [10] also demonstrate that the reward of DQN algorithms on RL environments will not keep increasing indefinitely(Section 3.2). They are also subject to catastrophic forgetting (Figure 1 and 2) on excessive training and the solution aroud this is to save the model state when the reward was maximum.

It is also worth noting that as the number of episodes to solve the environment game decreases, lesser number of frames are processed and hence the model does not over-

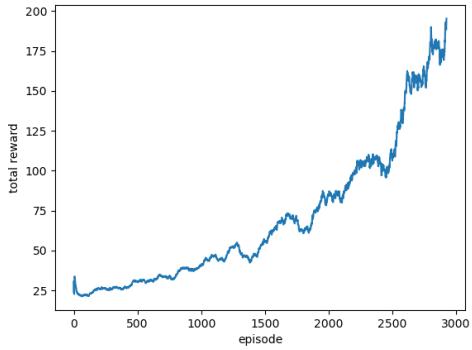

Figure 6: $PER + DDQN + Dueling + Multi - step$ Performance (Rewards averaged over 100 episodes)

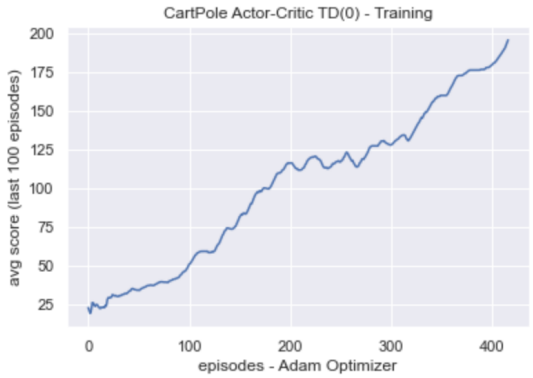

Figure 7: Actor Critic Performance. **Note: there is a major performance upgrade because the environment used here is cartpole-position and velocity as opposed to cartpole-images**

Table 1: Performance comparison and number of images processed for various algorithm combinations

Algorithm	Score	Episodes	Frames
DQN	119.5	Unsolved	625000000
DDQN	138.76	Unsolved	830000000
DDQN+PER	195.0	2500	498000000
DDQN+PER+ DUELING	195.0	1900	314000000
DDQN+PER+ DUELING+MULTI-STEP(n=3)	195.26	3000	436000000

fit. Is is visualized that since the DNN is similar for all approaches, the number of episodes to initiate is similar. This is the reason that there is a dip in the rewards of all curves after episode number 2600.

4.2. Does Image-Only State Representation of Cartpole Adequately Represent Observations ?

It was observed that using state information provided by the environment solves the problem in 250 episodes using DQN, However, DQN is unable to solve the image representation of the environment, and specialized algorithms solve the environment in 2000 episodes. It has been observed that using Cartpole position and velocity state space, the Actor-Critic agent solves the environment in less than 430 episodes(Figure 7). This makes us question the whether the images based state space is able to accurately and best represent the observations. If it is not, then can this problem be called a Partially Observable Markov Decision Process. In that case using Deep Recurrent Q Learning [2] can provide even better results. It is also worth studying that if DRQN performs better than standard DQNs, should the state space be improved by adding more effective sensors to the environment to make the state space represent the observations to a higher degree. This will be left to future work.

4.3. Best Model Configuration

It is clearly visible that the optimal model for solving this environment along with the modifications in algorithms is $DDQN + PER + DuelingNetworks$ which solves the environment in 1900 steps. It is also worth looking whether the performance of this rainbow algorithm on original Cartpole state space is highest. We leave that to the future work.

4.4. Training once in episode vs training once in each step of episode

All prior experiments demonstrate the training once in each step of the episode. This makes the model converge in 1500 episodes. However, Figure 8 demonstrates the results of training only one batch per episode. This figure also demonstrates the impact of noisy nets instead of $\epsilon - greedy$ policy, and it is shown to perform better than multi-step learning. It can be seen that the previous approach is much faster and performs many training steps without conduction too many environment steps. This can particularly be useful if taking an environment step is too expensive.

4.5. Multi-Step Learning

It was visualised that the performance of multi-step learning when used with DDQN+PER+Dueling Networks shows slower learning, but when experiments wereperformed on just multi-step learning on DQN, the results showed major improvements(Figure 9). More analysis has to be done on how to integrate multi-step learning with Dueling Networks. We leave this to future work.

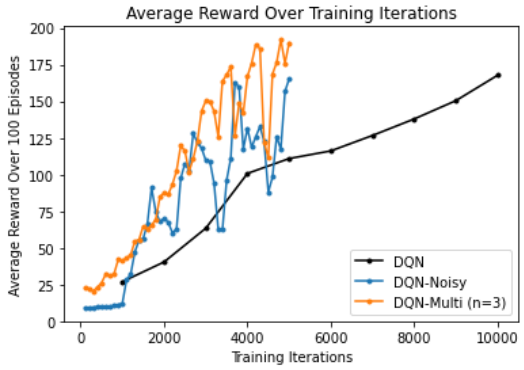

Figure 8: Performance of DQN, NoisyNets and Mult-step networks on Tensorflow framework

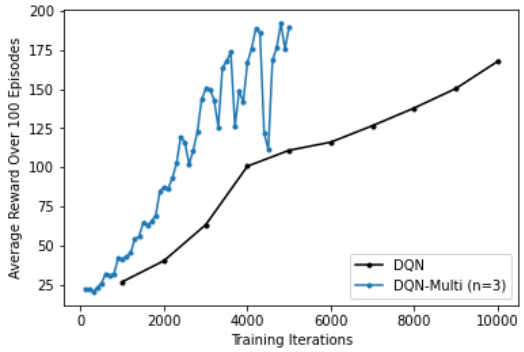

Figure 9: Performance of multi-step against DQN learning

5. Contributions

I worked on establishing the initial baseline for DQN, wrote the code for extending Double DQN, Double DQN + PER, Double DQN + PER + Dueling Networks. All 3 worked equally on designing the experiments, and the theory behind these algorithms. We leave the addition of Noisy Nets to the rainbow to future work. We also leave the visualization to other environment with more continuous action spaces to the future work.

References

[1] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration, 2017.

[2] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps, 2015.

[3] J. F. Hernandez-Garcia and R. S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target, 2019.

[4] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2015.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[10] M. Roderick, J. MacGlashan, and S. Tellex. Implementing the deep q-network, 2017.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2015.

[12] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization, 2015.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.

[15] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015.

[16] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning, 2015.