MsPacman-V5

Using DQN [1], DDPG [2] & SAC [3]

Aakash Kumar Dhal
(Department of Computer Science)
University of California, Riverisde
Riverside, United States of America
adhal017@ucr.edu

***Abstract*—We will explore three different Reinforcement Learning Algorithms to solve an Atari Environment call MsPacman-V5.**

***Index Terms*—Deep Q-Learning, Deep Deterministic Reinforcement Learning, Soft Actor Critic methods**

I. Introduction

Here we are trying to solve an atari game using three different Reinforcement Learning algorithms that have been marked as tipping points in the history of RL development. The first algorithm that is being used is Deep Q- Learning. Deep Q-Learning (DQL) is a reinforcement learning algorithm combining Q-Learning with deep neural networks. It approximates the Q-function, which estimates the value of taking an action in a given state, using a deep neural network called a Q-network. DQL utilizes experience replay, storing state transitions in a memory buffer and randomly sampling from it to break correlation between sequential experiences and stabilize training. It also employs target networks, which are periodically updated copies of the Q-network, to mitigate instabilities from rapid policy changes. The objective is to minimize the loss between predicted Q-values and target Q-values, updating the network using backpropagation. The second algorithm that was used was Deep Deterministic Policy Gradients or DDPG. It is an actor-critic algorithm for continuous action spaces, combining ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). DDPG uses two deep neural networks: the actor network, which directly maps states to actions, determining the best action in a given state; and the critic network, which evaluates the action output by the actor by estimating the Q-value. Both networks are updated using backpropagation: the critic by minimizing the loss between its predicted Q-values and the target Q-values, and the actor through policy gradients, guided by the critic's insights. DDPG employs experience replay and target networks, similar to DQN, to enhance stability and performance. The third algorithm that we are trying to implement is Soft Actor Critic, which in itself is a family of algorithms but here I am implementing its vanilla version. Soft Actor-Critic (SAC) is an off-policy, actor-critic algorithm optimized for continuous action spaces, emphasizing stability and efficiency. It incorporates the maximum entropy framework, which encourages exploration by not only maximizing expected reward but also maximizing entropy—diversity of the action distribution. SAC employs three networks: two Q-networks (critic) to reduce overestimation bias by taking the minimum of their predicted Q-values, and one policy network (actor) that outputs a probabilistic distribution of actions. These networks are updated via backpropagation, with the actor adjusted to maximize a trade-off between entropy and Q-value. SAC also uses experience replay for efficient learning and employs separate target networks for the critics to stabilize training.

II. The Problem

The problem that I want to solve is an atari game called MsPacman-v5. It has a continuous observation space but discrete action space. It is best suited for DQN that was developed to solve problems which had continuous action spaces that were otherwise not solvable using tabular methods. Here although we receive three input channels for each image we do not put all of them inside our neural networks. We take an average across the channel dimension hence giving us a grayscale image. This is then fed to the neural networks for training and inference as well. Another problem that I faced was the rewards that I was getting out of the environment. They were very sparse and if these were used to train our model then it did not learn anything. There a custom reward function was developed where extra rewards were added to the regular rewards based on three conditions. Here the rewards could be both positive as well as negative. So if we achieved a reward of 10(the only reward) from the environment, I would double it. If we lost a life I would add a negative reward of -10. And finally if our agent stays at a single position, which is 2 consecutive frames are the same then I would add a negative reward of -1. This enhanced our model learning as the returns were no longer varying in tighter bounds, it was exploring enough and hence getting higher per episode returns in some cases.

III. Technical Methodology

Now that our problem statement is ready we can get into the nitty gritties of our solution. For all the solutions that have been proposed, the Replay Buffer. It stores past experience tuples (state, action, reward, next state). By sampling randomly from this buffer, it breaks correlations between consecutive

experiences, enabling more stable and efficient learning. This mechanism allows algorithms to reuse past data, improving sample efficiency and convergence. This improves our learning by a huge margin.

A. Deep Q Learning

The fundamental building block of this model, the QNetwork employs a convolutional neural network. It is built using PyTorch's 'nn.Module', employs a convolutional neural network architecture suited for input data that is spatially structured, such as images. It starts with three convolutional layers ('conv1', 'conv2', 'conv3'), each followed by a ReLU activation function to introduce non-linearity. These layers progressively increase in depth from 32 to 128 channels, capturing increasingly complex features. A max pooling layer ('pool') reduces dimensionality and computational complexity after each convolutional layer. A dropout layer ('dropout') is incorporated to prevent overfitting by randomly disabling neurons during training. The network transitions from convolutional layers to fully connected layers ('fc1', 'fc2'), where 'fc1' compresses the flattened features into a 512-dimensional space, and 'fc2' maps these features to the action dimensions specified by 'dim_action'. This architecture enables the network to learn a mapping from states (input images) to action values, crucial for decision-making in environments represented visually. The Q-network learns by minimizing the loss between its predicted Q-values and target Q-values derived from a separate, periodically updated target network. This target network stabilizes learning by providing consistent, older Q-value estimates, reducing the moving target problem where Q-values change too frequently for effective learning. The Q-target is updated regularly using a method called Polyak averaging [4].

B. Deep Deterministic Policy Gradients(DDPG)

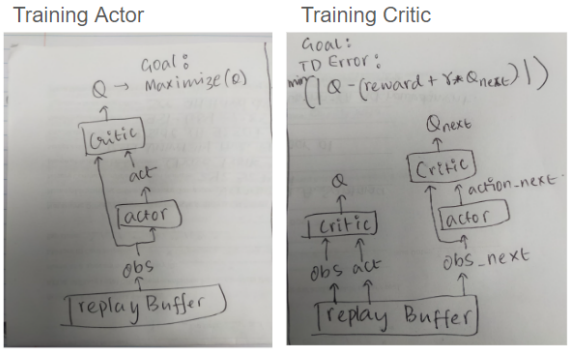

Fig. 1. Structure of DDPG Training

The DDPG implementation is almost the same fundamentally, but has an implementation trick that makes it useful especially when we have a continuous action space along with continuous observation space. It has two CNNs, not one unlike DQN. The first is called Actor and the second is called Critic Network. The actor takes in the environment's observation and spits out action(deterministic and not a probability distribution over all actions). Then we have the critic network that takes both the action and the observation for that action and calculates the Q value. Our goal will be to maximize the Q value as much as possible. The training is also tricky in this case. We have used two target networks that are used as a baseline to stabilize learning. The weights will be averaged after given steps using Polyak averaging. The only tricky part in the entire process is adding the action tensor to the observation for Q network. The solution that has been proposed here adds the action tensor in the denser latent space and not in the sparse convolutional phase. The remaining process is almost similar.

C. Soft Actor Critic(SAC) Methods

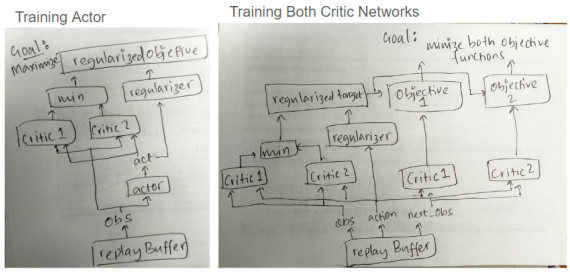

Fig. 2. Structure of SAC Training

The SAC implementation is a bit trickier as it contains a one extra critic network to stabilize learning even further. They are useful for solving the problem of overestimation bias that can occur with single Q-value estimations. Overestimation bias arises when the estimated Q-values consistently exceed the true Q-values, leading to suboptimal policy learning and performance degradation. The twin critic architecture mitigates this issue by having each critic independently approximate the Q-value function from the same experience tuples (state, action, reward, next state) but potentially learning slightly different Q-value estimations due to different initialization and stochastic gradient descent paths. During the update phase, SAC uses the minimum value between the two critics' outputs for each state-action pair. This conservative approach of taking the minimum helps in reducing overestimation by ensuring that only the smaller, and presumably less biased, of the two estimates is used to update the policy. This technique not only stabilizes training by providing more reliable value estimates but also promotes more robust exploration and learning in complex environments where overestimations might lead the agent into suboptimal policies.

IV. Numerical Results

Although the level of sophistication(in terms of algorithmic complexity) increases as we go from DQN to DDPG to SAC, the performance decreases. The main reason can be attributed to the complexity of the algorithms and their enormous training times. The number of parameters to be trained increases significantly as we go from one algorithm to another.

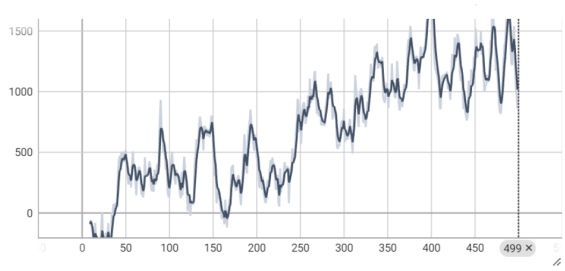

Fig. 3. Episodic Returns from DQN

DQN's learning was much more stable and reached a maximum reward of 1670 during the last episodes.

DDPG's learning was much more unstable although it was tending to gain higher rewards.

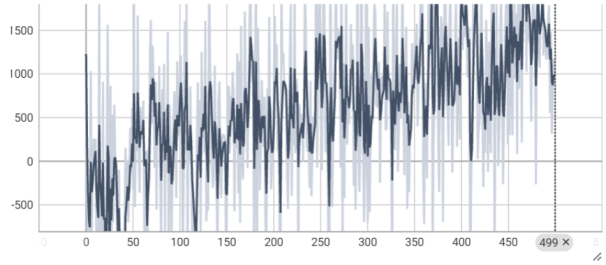

Fig. 4. Episodic Returns from DDPG

SAC model was too sophisticated for the problem at hand. Moreover Pacman V5 has a discrete action space and SAC has double the quantity of parameters to finetune. Maybe this would have converge in 500 more iterations.

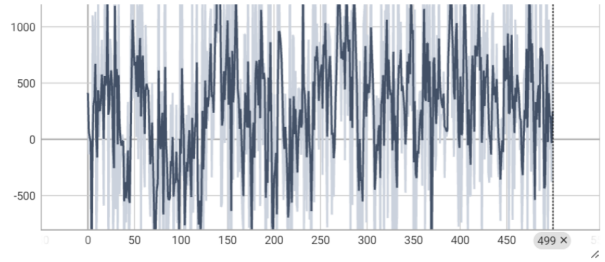

Fig. 5. Episodic Returns from SAC

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236.

[2] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie 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 (2015): 529-533.

[3] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning* (pp. 1861-1870).

[4] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1-17.