begindocument/before

Examining the efficacy of Dueling Network Architectures

(Dated: January 23, 2021)

Dueling Networks for Deep Reinforcement Learning offer an alternative and in some cases a more reliable approach to training agents. In this paper we will evaluate the performance of a Duel Architecture Network trained on the Enduro environment. We'll use the mean score over 100 episodes to compare the performance of

I. INTRODUCTION

In modern Neural Networks find an ever increasing number of applications in our daily lives. For Reinforcement Learning they offer a diverse toolbox that can be used to train agents to perform important tasks. In the previous decade there have been many research there had been made several important advancement, pertaining to the training of Deep Neural Networks to solve the Reinforcement Learning Problem. But different methods can be applied to different sets of problems.

In Mnih et al[2] we are shown a method of training Agents using a Deep Neural Network(DNN) in order to approximate the Q-function of an environment, called a Deep Q-Netework. The agent performed very well, and was event to outperform humans only after a week of training. Only a short time after that, improvements to the original original algorithm were made, Hasselt et al.[3] , called Double DQN. It offered in some cases a less optimistic agent, which resulted in fewer prediction errors.

In this paper we will be looking at another approach, that builds upon the aforementioned methods. We'll be using a slightly modified Neural Network, that was proposed in Wang et al.[1]. The goal here is to test their claim that the neural network offers better policy evaluation and can be used to train a better agent for playing Atarai 2600.

II. BACKGROUND

A. Mathematical definitions

We currently use an episodic environment, as described in Sutton & Barto., that's provided by the Arcade Learning Environemnt. The game we're training the agent to play is Enduro, a racing game. The state s_t provided by the environment for timestep t, consists of N number of frames $s_t = \{x_0, ..., x_N\}$. For our current needs we have fixed $N = 4$. The action space provided by the Enduro-v0 environment is $|\mathcal{A}| = 9$. The rewards provided by the environment are clipped to $[-1, 1]$.

We use the definition of the values for a state-action pair (s, a) and a state s, according to the policy π:

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t \,|s_t = s, a_t = a, \pi\right](1)$$

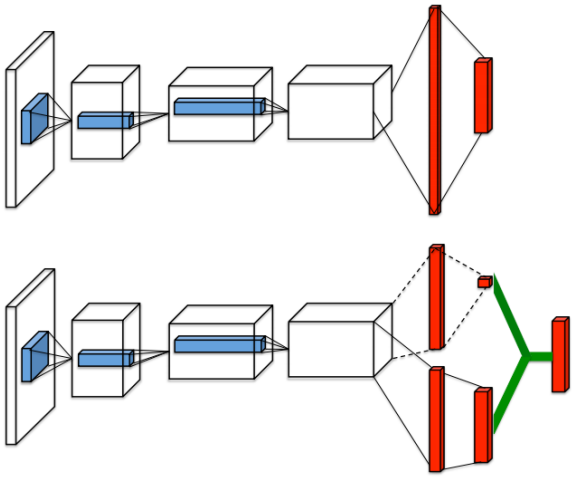

FIG. 1: This offers a visual representation of a Dueling Network compared to a normal DQN network. As we can see for the dueling network there are two streams that represent the Advantage and State value functions which are then combined in a single output layer. This figures was taken from the research paper in Wang et al.[1]

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^{\pi}(s,a)\right] \quad (2)$$

Using the state-action function(i.e. Q function) can be computed recursively in the following manner:

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t \,|s_t = s, a_t = a, \pi\right](3)$$

The optimal Q-function is also defined as $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$. Using the optimal deterministic policy we can see that $a = \arg\max_{a'\in\mathcal{A}} Q^*(s,a')$ and that $V^*(s) = \max_a Q^*(s,a)$. Following all of the above, we get that the optimal Q function satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}\, r + \gamma \max_{a}^{'} Q^*\left[(s',a')\,|s,a\right]$$

Another important piece of the puzzle in this paper is the advantage function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \quad (5)$$

If we use the optimal policy π we can see that $\mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}\left(s,a\right)\right]=0$. The advantage function gives us measure of how important an action is a given state. We can then rewrite the Q functiona as:

$$Q^{\pi}\left(s,a\right)=V^{\pi}\left(s\right)+A^{\pi}\left(s,a\right) \tag{6}$$

This equation is the basis for the Dueling Network as we will demonstrate later in this paper.

B. DQN and Double DQN:

For this paper we will be using a deep Q-network(Mnih et al[2]), also knowns the DQN algorithm, in order to approximate the values of the optimal Q-function.In order to get the most accurate results, for this particular network we'll be empoying the Double DQN(DDQN) algorithm(Hasselt et al.[3]), as it was used in Wang et al.[1].But first we will have to give an overview of Deep Q-Learning.

For Deep Q-Learning in order to approximate a Q-function we use a deep Q-network: $Q\left(s,a;\theta\right)$ with parameters θ. It's optimized for the following loss functions:

$$\mathrm{L}_i(\theta_i)=\mathbb{E}_{s,a,r,s'}\left[\left({y_i}^{DQN}-Q\left(s,a;\theta_i\right)\right)^2\right] \tag{7}$$and
${y_i}^{DQN}=r+\gamma\max_{a'}Q\left(s',a';\theta^{-}\right)$

where θ^{-} are the parameters for the target network, which is a second Q-network that's used only for training. A key improvement to DQN was to freeze the parameters of that network for a fixed number of iterations and update the *online network* with gradient descent. After that fixed amount of steps the target network is updated to use the parameters of the current online network and the process keeps repeating until the training run is over.

The gradient update can be written as:

$$\nabla_{\theta_i}L_i\left(\theta_i\right)=\mathbb{E}_{s,a,r,s'}\left[\left({y_i}^{DQN}-Q\left(s,a;\theta_i\right)\right)\nabla_{\theta_i}Q\left(s,a;\theta_i\right)\right] \tag{8}$$

This approach is an off-policy algorithm, since we produce datapoints using an ϵ-greedy policy which is different than the policy the agent is trying to learn. For the ϵ-greedy we use a schedule that decreases the epsilon as the number of iterations grow. We store all of the datapoints in replay buffer and sample from there. The batch size we've decided to use is 32 tuples structured as (s,a,r,s'). The replay buffer handles the encoding of the observed frames into a state of for arrays as described in the previous subsection.

Making a single change to the DQN algorithm offers superior results. DQN can lead to overoptimistic value estimates as shown in van Hasselt et al.[3], 2010. Changing the target equation in this manner:

$${y_i}^{DDQN}=r+\left(s',\arg\max_{a'}Q\left(s',a';\theta_i\right);\theta^{-}\right). \tag{9}$$

The gradient descent algorithm remains unchaged, but it just uses the target y_i^{DDQN} instead of y_i^{DDQN}.

III. DUELING NETWORK ARCHITECTURE

In the Dueling Network Architecture research paper by Wang et. al., we are presented with a different approach for constructing a Q-Network. This new network, refered to as the dualing network, consists of 3 convolutional layers just as described in the original paper for Deep Q-Learning(Mnih et al., 2015[2]), but instead of a single fully connected layer we create 2 parallel layers with 512 hidden units each. They can be looked as 2 streams of fully connected layers, one with $|\mathcal{A}|$ number of outputs and the other with a single output.The two layers are used to calculate the Advantage and State value estimates respectively. The key insight of the authors of the specified paper was that, is that there are many cases where estimating the value of each action choice is unnecessary. For the Enduro game, moving either left or right is unnecessary for most cases, save for when the agent would want to avoid a collision.

In the training the Dueling Network outputs the result of the two streams which is then combined in a post-processing step shortly before calculating the loss. The authors of the paper do specify that it should be done in the last layer of the Dueling Network so as to provide the same number of outputs as a normal(i.e. single stream) DQN network. But in the training code that step offers results that are equivalent to just combining them in a single final layer. The reason why we must combine the outputs is because we want this network to be trained using conventional algorithms, such as DDQN.

For our research we'll be using this equation in the combination step:

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

Lets explain why we need to use equation (10). We could still build a parametarized version of equation (6), described in the previous section such as this:

$$Q^{\pi}\left(s,a;\theta,\alpha,\beta\right)=V^{\pi}\left(s;\theta,\beta\right)+A^{\pi}\left(s,a;\theta,\alpha\right) \tag{11}$$

but in practice this results in poor performance for the Q-Network.

If we recall this expression for equation (6) and that $\mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}\left(s,a\right)\right]=0$ for the optimal policy π, and that for the deterministic policy $a^{*}=\arg\max_{a'\in\mathcal{A}}Q\left(s,a'\right)$, it follows that $A\left(s,a^{*}\right)=0$. This is what the network should approximate to after a large enough number of iterations. The authors of the original paper specify that equation (11) is unidentafiable in the sense that given Q we cannot recover V and A uniquely, which means that

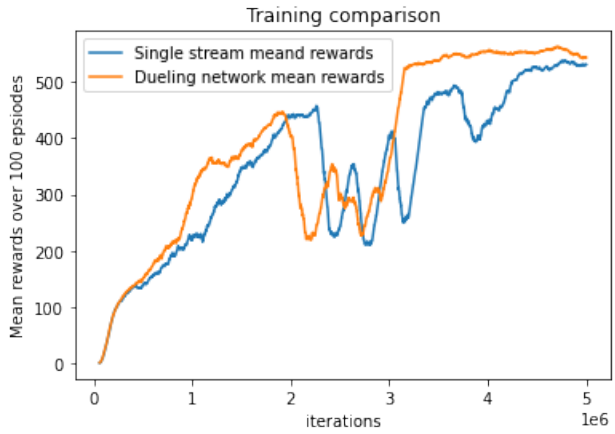

FIG. 2: In this example we compare the performance of a Dueling Network to that of a Single Stream Network based on the mean performance over 100 episodes. It should be noted that not all of the hyper parameters used for the training are the same for both of these networks. Due to some technical difficulties we weren't able to get results with the same random seed. But both of these networks use the same epsilon schedule for the epsilon greedy policy used to gather datapoints for the deep Q-Learning algorithm. As we can see the Dueling Network slightly outperforms the normal one in the beginning and but then starts to loose out. Both of them are able to get out of the slump in performance around the middle of the iteration range, but the Dueling Network remains stable and reaches a new maximum, while the Single Stream Network keeps zig-zagging until it starts to reach a similar mean value.

Q-Network will not be able to recognize all of the values for the different state-action pairs.

Equation (10) enforces a stable optimization process. There was one other version of the provided formula, but it's not relevant to this research paper. This is explained in more detail in Wang et al.[1]

IV. RESULTS

For our experiments we created a Normal Single stream Q-Network similar to the one described in Mnih et al.[2], but with 1024 hidden units in the fully connected layer of the network. This was done in order have an equal number of parameters in both the single stream and the Dueling Network. We're going to use this as a baseline for performance comparisons.

Both networks were trained for 5 000 000 iterations using a batch size of 32. We have iterated over 6 different groups of hyperparameters, but since we were using Google Colab for training, there were timeouts on several occasions. Unfortunately we weren't able to run all of the tests we wanted and the gathered data might prove inconclusive. But from what we've seen the Dueling Network does perform better at first. Also it's worth noting that both networks experience a massive drop in performance around 2 500 000 iterations(within a margin of several hundred iterations). We can see that in Fig. 2. This we think can be explained by the fact that the replay buffer is filled completely with data points acquired by using actions with the highest value provided by the Q-Network itself and that the gradient descent entering a region of a local minimum of the Loss function. And from the results gathered the Dueling Network is able to remain stable more often than the single stream network at times. We don't have enough data to make a conclusive statement on the results, but from the current data we are hopefull that this architecture can be used to offer more robust and better performing estimators for the Atari game environment.

V. CONCLUSION

After running several tests using both types of architectures we can see that Dueling Network shows promise for future use. In Wang et al.[1], they were able to run it over a multitude of games and in most cases it outperformed the single stream version of DQN. For future uses we could include Prioritized Experience Replay(Schaul et al.[4]) in the training algorithm in order to explore how it will affect it's performance. Also looing into how the training will go for other games is an interesting topic. Will it behave the same way as for Enduro? It also should be noted that it could be beneficial to explore domains outside of Atari Game Environments.

VI. REFERENCES

[1] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas Dueling Network Architectures for Deep Reinforcement Learning

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M.,Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. Human-level control through deep reinforcement learning. Nature, 518 (7540):529–533, 2015.

[3]van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. arXiv preprint arXiv:1509.06461, 2015

[4]Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In ICLR, 2016. [5]Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.

Acknowledgements I would like to thank the course lecturer Marin Bukov, Ph.D. for allowing me to use some of his code to help setup the training environemnt. Also if you want to look at the code used for the training go to: https://github.com/dmitsov/RL-Dual-Networks