Atari Carnival: Maximizing Score with DRL

Branden Lopez, Brandon Palomino
Department of Computer Science, San José State University, San Jose, California 95192, USA
Email: branden.lopez@sjsu.edu, brandon.palomino@sjsu.edu

***Abstract*—Since it's birth video games have spawned competitions to showcase individual skill and triumph over others. Since then people have developed creative ways to maximized score and have developed non-human agents to play games. In Deep Reinforcement Learning (DRL) an agent receives the state of the game and makes a reward maximizing move according to a learned policy. These policies are defined by the DRL architecture which vary in size, computation, and rely on various algorithms to drive it's development. In this paper we study two DRL algorithms Implicit Quantile and Deep Q Networks, their ability to perform on Atari Carnival, training time, and compare results.**

***Index Terms*—Reinforcement Learning, Deep Learning, Implicit Quantile Network, Atari, Gaming Agent**

I. Motivation

Carnival is a fixed 2D shooter developed by Gremlin and was released by Sega in arcades in 1980. Eventually, the game would be ported to home consoles such as the Atari 2600. Like many Atari games, the main objective was simple. The goal of the game is to shoot targets on the screen while conserving a limited supply of ammunition as long as possible. The strategic elements to achieving a high score in this game required maximizing hit shots while also saving resources, which is a common trope in many arcade shooters at the time. Because of this game design philosophy, we believe that a machine learning agent could be constructed to achieve a high in game score.

For this project, we will be using TensorFlow, which is an open source python toolkit consisting for machine learning and artificial intelligence. The machine learning agent will be trained and executed on Google Colaboratory and a local computer.

Applying deep reinforcement learning onto a game like Carnival is significant because as the technique can be applied to different types of video games. Atari games are simple constructed environments that can be learned upon, they provide us an insight on how video games are studied to develop precise ML models. While this is not necessarily a a new type of problem, our overall motivation is to implement an advanced DRL agent that achieves a high score in Carnival that can be compared to previous DRL attempts to gauge performance improvements.

II. Formal Problem and Evaluation Criteria

The selected version of Atari Carnival is Carnival-v0 and is built with OpenAI Gym. Once built a 214x160x3 image represents the state $\mathbf{s}$ of the game and contains information about where our agent is, the limited supply of ammo, targets, and the score. From this state the agent can select an action

$$\mathbf{a} \in A = \{Noop, Fire, Right, Left, RightFire, LeftFire\}.$$

These actions contain the keys to navigating the game, shooting targets, and overall interaction strategy. Once an action is taken, the state is updated to reflect that action. Targets vary in distances from the agent, return different rewards $\mathbf{r}$, and move with time. We hope that the agent will learn how these interactions work in order to maximize score.

Since the states are represented as an image it is imperative that a Convolutional Neural Network (CNN) is employed. In a CNN a tensor is fed to a convolution layer and kernels slide across the tensor, reducing the spatial size of the tensor and increasing its depth. The output is then used with other layers such as an activation function to help learn complex patterns in the data.

III. Methodology

The well known approach for DRL is to use a Deep Q-Network (DQN) [1]. Q-learning is a model free method which maps state-action pairs to a scalar value which represents the expected long-term reward if that action is taken; a DQN extends this by having a CNN parameterize Q, and replaying stochastic events from memory to speed up convergence. While this algorithm brought much attention to DRL, many new algorithms have been developed since then.

An Implicit Quantile Network (IQN) builds upon the idea of a DQN but differs as a distributional learning algorithm. In distributional reinforcement learning the distributions over returns is considered instead of the scalar Q-value [2]. Distributional algorithms have shown increased sample complexity and better long term results and we are hoping these improvements. To accomplish this an IQN attempts to learn the quantile or inverse Cumulative Distributional Function (CDF) $Z_\tau(s, a)$ from samples of some base distribution τ. Wrapping the sample in a distortion measure β which effects risk seeking behaviors. Leading to our Q-Value:

$$Q_\beta(s, a) := E[Z_{\beta(\tau)}(s, a)]$$

In addition to changing the method for approximating q, an IQN changes how the convolutional neural network estimates quantiles by adding an embedding. Let $\psi : s \longrightarrow \mathbb{R}^d$ be the convolutions used to arrive at our latent space and in a DQN $f : \mathbb{R}^d \longrightarrow R^{|A|}$ are the fully connected layers mapping the latent space to the state-action values. An IQN adds to this the embedding $\phi : [0, 1] \longrightarrow \mathbb{R}^d$ which contains the distortion

risk measure. Altogether these function are used by the IQN to approximate our state-value distribution $Z_\tau \approx f(\psi(s) \odot \phi(\tau))$ Where $\odot$ is the element wise product.

The technique of estimating inverses CDFs is known as Quantile Regression and in prior papers has been estimated using the loss Quantile Regression Huber Loss function. This loss function is as stated:

$$\rho_\tau^k(\delta_{ij}) = |\tau - \mathbb{I}_{\{\delta_{ij}<0\}}| \frac{L_k(\delta_{ij})}{k}$$

Where $L_k(\delta_{ij})$ is the Huber loss function.

$$L_k(\delta_{ij}) = \begin{cases} \frac{1}{2}\delta_{ij}^2 & |\delta_{ij}| \leq k \\ k(|\delta_{ij}| - \frac{1}{2}k) & o.w \end{cases}$$

Where δ_{ij} is the temporal difference error

$$\delta_t^{\tau,\tau'} = r_t + \gamma Z_{\tau'}(s_{t+1}, argmax_{a \in A} Q(s,a))) - Z_\tau(s_t, a_t)$$

We hope that these vast improvement which have been shown to converge to the true quantile distrubtion leads to a higher maximizing score and accomplishes it quicker than the DQN.

IV. Evaluation Settings

A large reason we chose Atari Carnival is due to it's lack of representation in the IQN paper. In this paper we view that IQN bests DQN in many tasks. The only time we see an DRL agent's score for Carnival with Upper Confidence Trees [3], where the score reaches a high of 5132. Because the score carnival displays is a natural indicator of playing ability, our goal is maximize this score. We will track the score vs episodes simulated to achieve this score. Similarly, we are interested in the number of steps the algorithm performs before termination and the time to train the algorithm.

V. Evaluation Results

As an IQN is built on a DQN we first observe how a DQN performs. The DQN is created according to Mnih's [1] algorithm in Tensorflow and ran for 25 episodes. The DQN results in figure 1 are lackluster, showcasing a variation and little performance over 25 episodes.

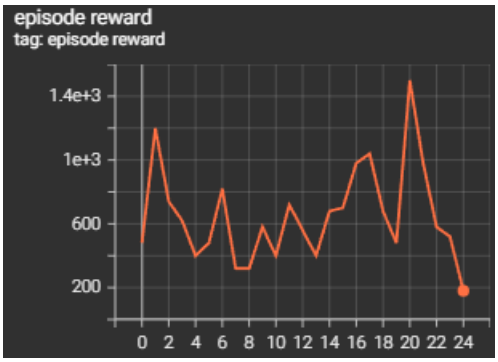

Fig. 1. Reward (Score) v Episode

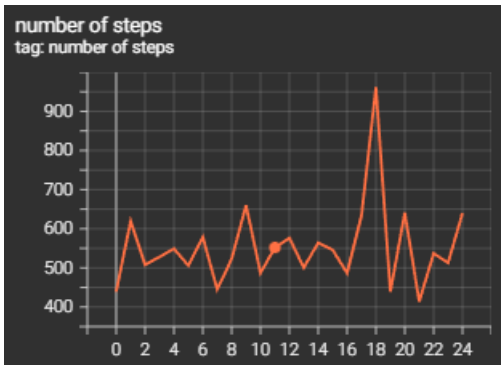

Fig. 2. Steps v Episode

Likewise, steps showed little improvement in figure 2. Episode training time was not recorded until the IQN was introduced but for a DQN stood around 8 minutes from visual inspection of our time function.

Next is the IQN. Due to software and hardware issues limited results could be ascertained for the IQN, to directly compare these results we compare them to a DQN with 5 episode of training. While debugging IQN software issues the CNN was modified in one training run with a max pooling layer, which significantly reduced the spatial information used to play a game and its results are included in figure 3.

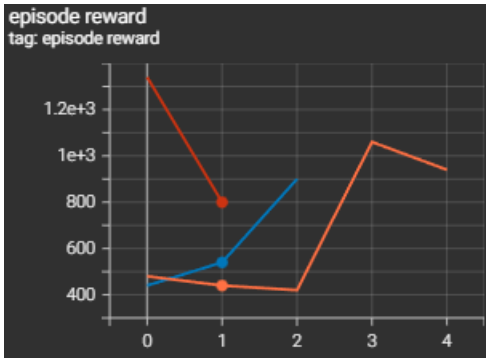

Fig. 3. Steps v Episode. DQN in orange, IQN in red, and IQN with maxpooling in blue.

Clearly the standard IQN starts off hotter than its other counterparts and during debugging, running this training again showed this to be true thru most of the runs. What is surprising is the Maxpooling IQN, despite the lack of information takes a quick jump over the DQN algorithm in episode reward.

In early episodes we do not train from replay memory until the number of samples surpasses 1000. Due to this the first two episodes fly by in matter of seconds. What is most interesting how the training time for the IQN with max pooling took much less time that the DQN. Of course a reduced spatial dimension requires less trainable parameters in a DNN is likely the cause for the significant reduction in training time, shown in figure 6.

Lastly, we view the number of steps each agent performs during an episode in figure 5 The steps all decrease after the

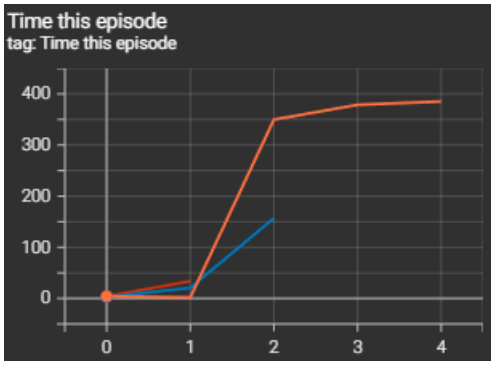

Fig. 4. Time v Episode. DQN in orange, IQN in red, and IQN with maxpooling in blue.

first episode but can logically be explained as the algorithm trying new actions while it attempts to maximize the score.

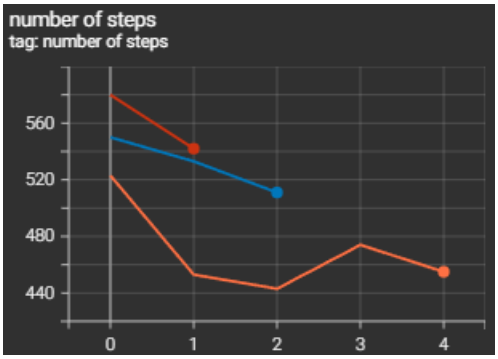

Fig. 5. Steps v Episode. DQN in orange, IQN in red, and IQN with maxpooling in blue.

VI. Discussion

While we would like to run these algorithms to convergence we did not because we lacked compute power and there was a memory leak in Tensorflow. In the DQN paper an epoch is considered as 10,000 steps and convergence dose not occur until 100+ epochs. Since our algorithm took 7 minute an episode where the average episode was 600 steps, it would be infeasible to see the algorithm converge given our limited time. The memory leak is an issue with understand the Tensorflow framework, when creating a model Tensorflow makes a graph containing the steps of the model; during our back propagation phase steps are continuously added nodes to the model. When nodes are continuously added, memory consumption increases until it will no longer fit on any GPU. Due to a novice understanding of Tensorflow we use operations that are unintentionally causing this problem, many operations that are provided by TensorFlow. In addition to convergence it would be beneficial to run until convergence many times to accurately determine if the early results are because of the robust algorithm and not simply because weight initialization was better on an individual run.

VII. Related Works

Video games continue to be a reliable environment for DRL algorithm development and testing. Maximizing a game score has been extensively studied in former works; while Carnival is not a part of the common Atari Benchmarks, only a single published paper (according to papers with code) has explored Carnival. To compensate for the lack of comparisons, research was done on with Google's Dopamine benchmark to explore DRL algorithm effectiveness on both the DQN and IQN. In figure 6, the benchmark showcases that the IQN is dooes vastly better in early training and converges to a higher score in the long term. In fact, google's benchmark supports that the IQN did better than UCT (5132) in Carnival. We believe that our agent could have achieved this high reward we did not have hardware and software limitations.

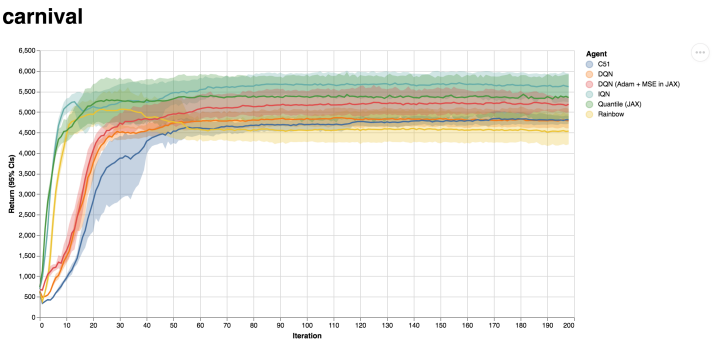

Fig. 6. Google Dopamine Benchmark for Carnival. DQN is orange, and IQN is teal.

VIII. Conclusion

The objective of this project was to create an original deep reinforcement learning agent that could achieve a high reward total in Carnival. In this project, the episode rewards simulated after conducting DQN and IQN show that there is potential for the agent. Based on our results, we can see that there is potential for IQN to produce an agent that can achieve high results in Carnival. However, further training and testing over more episodes is required to reach this goal. Reinforcement Learning provides a different perspective on how we view fundamental video game design principles. Understanding the direction in which video games are modeled will open more opportunities for new machine learning algorithms to be tested on. If current trends and studies continue, the range of viable applications of machine learning in the domain of video games will see exponential growth. As the cost of creating high-quality video games and existing game mechanics increase, machine learning opens up new areas of game-design space to be explored and experimented.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[2] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1806.06923

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013. [Online]. Available: https://doi.org/10.1613%2Fjair.3912