ECE239AS Final Report

Dueling Network Architectures for Deep Reinforcement Learning

Sarat Bhargava
805219546
sarat2895@ucla.edu

Eden Haney
005221845
ehaney@ucla.edu

Radhika Nayar
805226085
radhica@ucla.edu

Abstract

Most approaches for improving deep reinforcement learning performance focus on the learning algorithm itself while using standard neural networks not necessarily designed for reinforcement learning. In this project, we study and implement a potential improvement to the standard neural networks used in Deep Q-learning applications, known as the dueling architecture. Given only raw pixel observations and game rewards, we implement the dueling architecture on top of the Double Deep Q-Network in the Space Invaders games on the Atari environment. We show that the dueling architecture outperforms the stand-alone implementation of the Double Deep Q-Network algorithm on the Atari 2600 domain.

1 Introduction

Deep Reinforcement Learning (DRL) has made its impact in the machine learning community due to many recent successes. However, most implementations of DRL use standard neural networks, such as convolutional networks, multilayer perceptrons (MLPs), long short-term memory networks (LSTMs) and autoencoders. The main focus of recent advances has been on designing improved RL algorithms while simply incorporating existing neural network architectures into RL methods. The Dueling Network Architecture for Deep Q-learning, or the *dueling architecture*, is a unique neural network architecture that is customized for model-free RL (1). It improves the performance of standard deep Q-learning algorithms such as the Deep Q-Network (DQN) (2) and Double Deep Q-Network (Double DQN) (3).

The dueling architecture, visualized in figure 1, is similar to the common single-stream Q-network used in DQN (2). However, the dueling architecture incorporates two streams that represent state-value function and state-dependent action advantage function which are combined via a special aggregating layer to produce an estimate of the state-action value function Q.

This architecture uses the insight that it is unnecessary to estimate the value of every action choice for each state. By explicitly separating two estimators, the dueling architecture can learn which states are valuable, without having to learn the effect of each action for each state.

2 Background

In reinforcement learning (RL), an agent learns by interacting with its environment. The agent transitions between different scenarios of the environment, referred to as states, by performing actions which yield rewards. The goal of RL agent is to learn optimal policies for sequential decision problems that maximize the future cumulative reward.

UCLA ECE 239AS Reinforcement Learning Spring 2020.

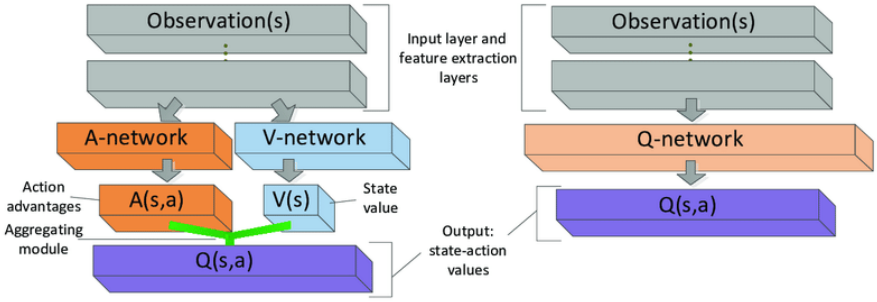

Figure 1: The dueling Q-network (left) and a popular single stream Q-network (right). The dueling network has two streams to separately estimate state-values and the advantages for each action; the output module (green) combines them. Both networks output Q-values for each action. From (4)

2.1 Q-learning

Q-learning is a simple yet powerful model-free value-based RL algorithm that finds the optimal action-selection policy using a Q-function. The Q-function uses the Bellman equation and takes two inputs: state (s) and action (a) as shown in equation 1. At any given state, the agent will take the action that will eventually yield the highest cumulative reward. One way to implement this is to store Q-values for all the possible state-action combinations in a table and then update the table using the Bellman equation as an update rule given in equation 2.

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right], \text{ and} \\ V^{\pi}(s) &= \mathbb{E}_{a\sim\pi(s)}\left[Q^{\pi}(s,a)\right] \end{aligned} \tag{1}$$

$$Q^{*}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^{*}(s',a') | s, a\right] \tag{2}$$

However, the Q-learning algorithm does not scale well when the number of states and actions become very large. The amount of memory required to save and update the table would increase as the number of states increases and the amount of time required to explore each state to create the required Q-table would be unrealistic.

2.2 Deep Q-networks

Deep reinforcement learning (DRL) can be used to scale up the prior work in RL to high-dimensional problems. DRL applies deep neural nets for representing the value functions within reinforcement learning methods. Using function approximation and representation learning properties of deep learning, DRL can efficiently deal with the curse of dimensionality and attain superhuman performance in several challenging task domains (4).

The deep Q-network (DQN) algorithm introduced by Mnih et al. (2) achieves human-level performance on Atari series games from pixel input. In DQN, the Q-learning's table is replaced with a neural network Q(s,a; θ) that approximates the Q-values. At each iteration, the network's parameters (θ) are updated by minimizing the following loss function:

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q\left(s,a;\theta_i\right)\right)^2\right] \tag{3}$$

with

$$y_i^{DQN} = r + \gamma \max_{a'} Q\left(s',a';\theta^{-}\right) \tag{4}$$

where θ^{-} represents the parameters of a fixed and separate target network.

Using two key techniques, target network and experience replay, DQN can learn value functions in a stable and robust way. The first technique is fixing the parameters of the target network Q(s',a'; θ) rather than calculating the TD error based on its own rapidly fluctuating estimates of the Q-values. The second technique, experience replay, uses a buffer for storing a dataset $D_t = \{e_1, e_2, ..., e_t\}$ of experiences $e_t = (s_t, a_t, s_{t+1}, r_{t+1})$ from many episodes (4). The Q-network is trained by sampling mini-batches of experiences from D uniformly at random. The experience replay makes it possible for training off-policy and enhancing the efficiency of sampling data.

2.3 Double Deep Q-networks

To reduce the overestimated Q-values in DQN, van Hasselt et al. (3) proposed the double DQN algorithm. Double DQN algorithm is same as DQN but with the target y_i^{DQN} replaced by y_i^{DDQN}. In Q-learning and DQN, the $\max$ operator uses the same values to both select and evaluate an action which leads to overoptimistic value estimates. To mitigate this problem, Double DQN algorithm uses the following target:

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q\left(s', a'; \theta_i\right); \theta^-\right) \tag{5}$$

2.4 Other Related Work

Baird et. al (5) first introduced the idea of separating value and advantage functions and updating them sequentially. This separation avoided the representation problem in Q-learning, where the Q function differs greatly between states but differs little between actions in the same state.

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

Prioritized experience replay introduced by Schaul et al. (6) builds on top of the Double DQN architecture and further improves the effectiveness of the experience replay. Unlike DQN and Double DQN where replay buffer is sampled uniformly, their key idea was to sample experience tuples with higher expected learning progress more frequently. This led to faster learning due to quicker convergence in sparse reward environment and improved final policy quality across most games on the Atari domain.

3 Architecture

The dueling architecture is similar to a single Q-network architecture where the lower layers are convolutional, such as in DQN. However, instead of following the convolutional layers with a single sequence of fully connected layers, it uses two streams of fully connected layers, which separately learn action-advantage function $A(s, a)$ and state-value function $V(s)$. These two streams are then combined via a special aggregating module (the green part of figure 1) to output a set of Q-values, one for each action. Because the output of dueling architecture is a $Q(s, a)$ function, it can be used with many existing Deep Q-learning algorithms such as DQN and Double DQN.

Unlike DQN which focuses on estimating every state-action pairs' value, the dueling architecture is able to ignore states which are not valuable and only learns the valuable states. Thus, the training efficiency of dueling architecture is much improved. Essentially in a RL task, the Q-values generated by the dueling network are more advantageous to the performance improvement than DQN.

Equation 1 and 6 give us $V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s, a)\right]$ and $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, respectively. Thus,

$$\begin{aligned}
&\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s, a)\right] = \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s, a)\right] - \mathbb{E}_{a\sim\pi(s)}\left[V^\pi(s)\right] \\
&\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s, a)\right] = V^\pi(s) - V^\pi(s) \\
&\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s, a)\right] = 0
\end{aligned} \tag{7}$$

Moreover, for a deterministic policy, $a^* = \arg\max_{a'\in\mathcal{A}} Q\left(s, a'\right)$, it follows that $Q\left(s, a^*\right) = V(s)$ and hence $A\left(s, a^*\right) = 0$.

To aggregate the layers we can consider making one of the streams of fully-connected layers output a scalar $V(s; \theta, \beta)$, while the other stream outputs an $|\mathcal{A}|$ dimensional vector $A(s, a; \theta, \alpha)$. Here,

θ denotes the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers.

Simply substituting these values into equation 6 would give us:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{8}$$

However, as $V(s; \theta, \beta)$ and $A(s, a; \theta, \alpha)$ are not true estimators for $V^{\pi}(s)$ and $A^{\pi}(s)$, $Q(s, a; \theta, \alpha, \beta)$ is only a parameterized estimate of the true Q-function.

Instead, we aggregate the streams by subtracting the average advantages as in equation 9.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A\left(s, a'; \theta, \alpha\right) \right) \tag{9}$$

According to (1), this form has the advantage of increasing separability of the two streams and increasing the stability of the optimization over other aggregation techniques they tried.

4 Implementation

This project has been implemented in Python using Tensorflow, OpenAI-Gym, and OpenAI-baselines libraries. We have learned to play the Space Invaders game using the Dueling DQN and Double DQN algorithms. OpenAI-Gym provides two different versions of the game: SpaceInvaders-ram-v0 represents every instance of the game as 128 length feature vector and SpaceInvaders-v0 represents every instance of the game as a 210x160x3 image. In the case of SpaceInvaders-ram-v0, for input we used the 128 length feature vector as is whereas for SpaceInvaders-v0, we pre-processed the images by gray scaling, down-sampling, and cropping to obtain an image of size 84x84. The RL agent takes as input the current pre-processed image and the history of past three pre-processed images to estimate the direction of movement/change in various objects in the image.

Convolutional Neural Network has been used to learn the optimal Q-function of SpaceInvaders-v0. This network has three convolutional layers followed by 2 fully connected layers. The first hidden layer convolves 32 8x8 filters with stride 4 with the input image and applies a ReLu activation function. The second hidden layer convolves 64 4x4 filters with stride 2, again followed by a ReLu activation function. The third hidden layer convolves 64 3x3 filters with stride 1. The final fully connected hidden layer with 512 units is connected to the flattened output of the previous convolutional layer. The output layer is same as the length of the number of possible actions, which is 6. In the case of Dueling DQN, we have 2 streams of separate fully connected hidden layers of size 512 diverging from the final convolutional layer as shown in the figure 1. The first stream computes the value function of the given state while the second stream computes the advantage function for actions. So the output layer for value function stream and advantage function stream are of size 1.

In order to train the above neural network, we used the Adam optimizer with mini-batch size of 32, learning rate as 1e-6 and trained for 500,000 frames, which is approximately 1,800 games. The behavioral policy used is epsilon-greedy with epsilon varied from 1.0 to 0.1 in a linear manner over 300,000 frames and the replay buffer size was 100,000. We check-pointed the model every 20,000 frames to get the best model as the RL training algorithms are very noisy.

Dense Neural Network has been used to learn the optimal Q-function of SpaceInvaders-ram-v0. This network has input layer of size 128 and two fully connected hidden layers with 128 neurons with ReLu activation function. The output layer for both Duel DQN and Double DQN follows the same architecture as above. We used Adam optimizer with mini-batch size of 32, learning rate as 1e-4 and trained for 200,000 frames. The discount factor used during the training was 0.99 and replay buffer size was 10,000. The behavioral policy is epsilon-greedy with epsilon varied from 1.0 to 0.05 in a linear manner over 50% training duration. We have check-pointed the model every 10,000 frames, evaluated the model using a separate environment for 20 episodes and recorded the mean and std of total rewards and episode length.

5 Results

5.1 Experiments on Space Invaders-Image version

Figures 2 and 3 visualise the metrics we examined. In figure 2 we see the estimated state-action values for double DQN (top) and the dueling architecture (bottom). We observe that Double DQN Q-values are much higher than the dueling architecture. This may correspond to overestimation in Double DQN.

In figure 3, the top plot shows the total rewards, or final score, from each game instance while the bottom plot shows the number of frames that the agent "lived" for in the game. We notice that both agents perform similarly in terms of how long they lived but that the dueling architecture is able to achieve a higher final score.

Interestingly, we recorded videos of each agent playing through one game instance after being trained for all 480k game instances. In these videos (link) we noticed that the Double DQN agent tends to hide behind a barrier, extending its life without scoring points. In contrast, the dueling agent hides behind the same barrier but is also successful in increasing its final score.

By comparing figures 2 and 3, in both Double DQN and the dueling architecture, we note that the improvement in the number of frames lived seems to correspond to an increase in the Q-values. This seems to suggest a true improvement in the state-action values after learning for around 250k frames ($\sim 900 gameinstances$).

Finally, in table 1 we look at the final score statistics of each algorithm. The table scores statistics correspond with what we saw in figure 3; double DQN hasn't learned how to score after 480k frames, while the dueling architecture has. We see the dueling architecture mean has increased from the baseline (which was taken using random weights). Double DQN's mean score has decreased but does so consistently. This supports the concept that double DQN has learned to stay alive, but not to score points.

Table 1: Score Statistics

Architecture	Mean	Variance
No Training	174.7	82.0
Double DQN	1.33	3.92
Dueling DQN	216.0	126.5

5.2 Experiments on Space Invaders-RAM/Feature vector

The RAM version of the game has been relatively easy to train compared to the Image version of the game and we are able to train both Double and Duel DQN. As described in the previous sections, we have trained the dense networks with two hidden layers to estimate the optimal Q function using Duel DQN and Double DQN algorithms. We have observed that both learning algorithms have some noisy improvements in terms of both average reward obtained and average length of the episode through out the training process. So we have check-pointed the model and evaluated the average episodic reward to perform early stopping. The results of our best trained models obtained using Duel DQN and Double DQN are shown in the table 2 while figure 5 shows the average rewards over training duration at each checkpoint.

Although both Double and Duel DQN algorithms have learned to play the game better than random actions, the Duel DQN agent has learned a clever strategy of hiding behind one of the bunkers and shooting only when the enemy object is straight above. This strategy minimized the chance of death and the agent was able to score as high as 800. The Double DQN agent has also learned to shoot but it does not always wait behind the bunkers and sometimes even shoots the bunker and gets killed. This reason why this happens is Duel DQN has better estimates for optimal action which is stay behind and shoot, although Double DQN also learnt the same due to its noisy estimates some times takes wrong action and gets killed.

In both RAM and Image versions of the game, duel architecture agent has learned a better strategy compared to Double DQN agent given the same compute time. We believe if given more time to train,

Figure 2: The Q-values of Double DQN (top) and Dueling DQN (bottom). The plot shows data points over 500k iterations averaged over 100 samples.

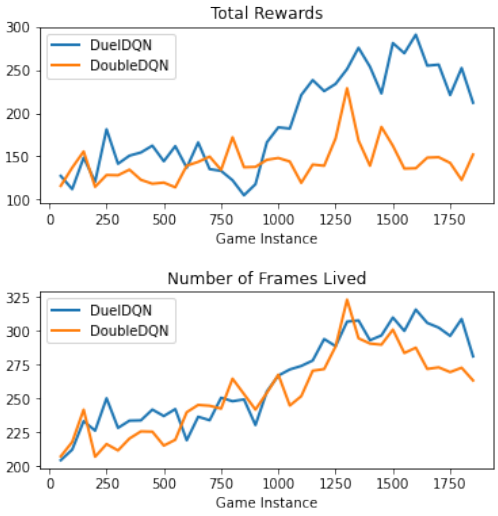

Figure 3: Total rewards (top) and number of frames alive per game instance (bottom). The plot shows data points averaged over 100 samples for both Double DQN (orange) and Duel DQN (blue).

the Double DQN agent for SpaceInvaders-Image would have learned to shoot and perform better, similar to the case of SpaceInvaders-RAM. However, due to computational limitations we are unable to train.

Table 2: Score Statistics of Best network obtained with Early stopping

Architecture	Mean	Variance
No Training	259.7	32
Double DQN	291.5	137
Dueling DQN	351.25	188

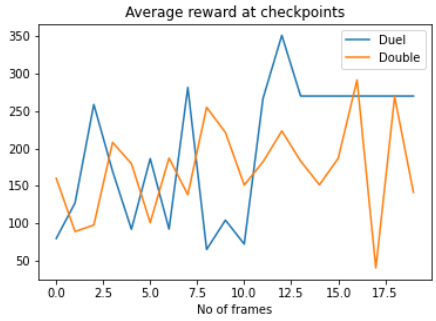

Figure 4: Average episode reward achieved by agent at each checkpoint

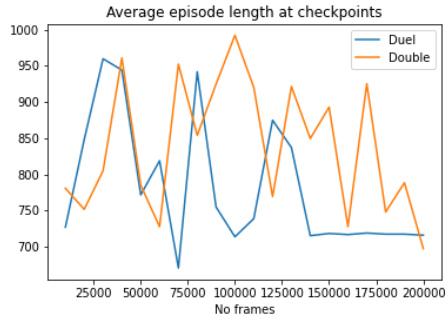

Figure 5: Average episode length achieved by agent at each checkpoint

6 Conclusion

In this project, we introduced a new neural network architecture called dueling architecture that explicitly separates the representation of the state values and state dependent action advantages. The main advantage of this factoring is to generalize learning across actions without imposing any

change to the underlying reinforcement learning algorithm. Based on our results, we can conclude that the dueling architecture far advances the double DQN algorithm in Space Invader Atari game environment where only some states are valuable. Furthermore, we can support the claim that the dueling architecture leads to faster convergence than a traditional single stream Q-network.

References

[1] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv:1511.06581 [cs]*, Apr 2016. arXiv: 1511.06581.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529–533, Feb 2015.

[3] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv:1509.06461 [cs]*, Dec 2015. arXiv: 1509.06461.

[4] M. Wu, Y. Gao, A. Jung, Q. Zhang, and S. Du, "The actor-dueling-critic method for reinforcement learning," *Sensors*, vol. 19, p. 1547, Mar 2019.

[5] I. Baird and L. C., *Advantage Updating*. Nov 1993.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *ICLR*, 2016.