Solving OpenAI's Lunar Lander using a Dueling Network Architecture

Sinan Harms

Abstract

The Dueling Network architecture first introduced by Wang et al. [2016] is ancomplementary approach of the Deep Q-Learning Networks. It's novelty lies in the separation of the value and advantage functions in the neural network. Thus it is able to learn which states are valuable, without having to learn the effect of each action for each state. This report demonstrated how a Dueling Network architecture can be used to solve OpenAI's Lunar Lander environment effectively.

Keywords: DDQN, DQN, Dueling DQN, Dueling Network, Reinforcement Learning

1 Introduction

Reinforcement learning algorithms can be classified in several families. The first one depends on whether the algorithm learns or uses the environment dynamics. This family, the so-called model based algorithm, uses the dynamics during the decision process. It needs to learn the transition probabilities, which are the probability of the agent transitioning from one state to another in order to approximate the model behind the unknown environment. If the algorithm doesn't make use of these dynamics it's called model free [Sutton and Barto, 2018].
The second big family for classifying algorithms are off-policy or on-policy. Off-policy algorithms learn a value function independently from the agent's decision process, meaning that the behavior and target policy can be different. The first one is the policy that the agent uses to interact with the environment and collect data, while the second one is what the agent is trying to learn or improve. This has the effect that the agent can explore the environment completely random using some sort of behavior policy and use the collected data to train a target policy which in return delivers high rewards. On-policy algorithms on the other hand require the behavior and target policy to be the same. These algorithms rely on the data that was collected by the same policy that it's trying to learn [Sutton and Barto, 2018].
Q-Learning and DQN algorithms are both model free and off-policy and therefore the Dueling Deep Q-Learning Network, short Dueling DQN, is also model free and off-policy. This report explores the theory behind the architecture, its implementation and evaluates its performance on the LunarLander gym environment by OpenAI.
It will start with a quick break down of the mathematics behind the Dueling Network architecture introduced by Wang et al. [2016]. Afterwards details of the implementation will be presented, like the environment, the specific network used for training and training process. The next part shows the result of the training and concluding with a short discussion of some challenges and pitfalls that were encountered during the process.

2 Theory

The main intuition behind the dueling network architecture is that for many states it is not necessary to calculate or rather estimate the value for each action as it is usually done in traditional Deep Q-Learning Networks (DQN). This also solves one of the problems DQNs usually suffer from,

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

namely the overestimation of the true rewards. The Q-values suggest that the agent is going to obtain a higher reward than what it will be in reality. To fix this a Dueling Network will split the Q-values in two different parts, the value function $V(s)$ and the advantage function $A(s, a)$. The value function $V(s)$ will predict the reward depending on state s, while the advantage function $A(s, a)$ returns the benefit of one action compared to all others. Considering a dueling network with to streams of fully-connected layers outputting scalars $V(s; \theta, \beta)$ and $A(s, a; \theta, \alpha)$, the addition of both result in the Q-values:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{1}$$

By this separation, the network can learn which states are valuable or not without learning the effect of each action every step. This comes in handy especially when an action does not necessarily affect the environment in a meaningful way [Wang et al., 2016].
However, when training a network like this a simple addition of the two streams or functions is not that simple. Given the function Q, determining the unique values for V and A is an almost impossible task, since there can be an infinite number of combinations of V and A for every value of Q. This leads to poor practical performance. This problem can be solved by implementing forward mapping:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha)) \tag{2}$$

This will force the advantage function to be of zero advantage at a chosen action, thus providing an estimator for the value function. From there values can be computed for the advantage function, therefore solving the problem. To further increase the stability of the optimization the $\max$ operator can be replaced by an average. This will change A and V to be off-target by a constant, but has the benefit that the advantages only have to change as fast as the mean without having to compensate any change in advantage regarding the optimal action's advantage. Hence the optimal action can be chosen based on $a^\star = \arg\max_{a' \in |\mathcal{A}|} Q(s, a; \theta, \alpha, \beta)$ [Wang et al., 2016]. Resulting in the equation:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_i A(s, a_i)) \tag{3}$$

With the dueling networks sharing the same input-output interface with traditional DQNs, there is no need to adopt a new training process. Any learning algorithm for DQNs can be recycled and used for a Dueling Network. Defining the loss of the model as the mean squared error (MSE)

$$\begin{aligned} &L(\theta) = \tfrac{1}{N} \textstyle\sum_{i \in N} (Q_\theta(s_i, a_i) - Q'_\theta(s'_i, a'_i))^2 \\ &\text{where } Q'_i = R(s_t, a_t) + \gamma \max_{a'_i} Q'_\theta(s'_i, a'_i) \end{aligned} \tag{4}$$

a gradient descent step can be used to update the model parameters θ [Wang et al., 2016].

3 Implementation

3.1 Environment

The environment in which the Dueling Network is to be trained is the LunarLander from the OpenAI gym environment. The goal here is to land a rocket at a set point of coordinates, the landing pad. The actions available to control the rocket's trajectory are do nothing, fire the left engine, fire the main engine and fire the right engine, making four discrete action (engine on/off). As for the observation space the environment provide eight states. Each one of the state gives information about the position (x,y), velocity (x,y), the tilt angle, the angular velocity and whether each one of the two legs of the rocket have made contact with the ground.
The rewards given for moving from the top of the environment to resting on the landing pad can range from $100 - 400$ points. Points are deducted if the rocket moves away from the target. A crash results in a reward from -100. If it comes to rest with a velocity of $(0, 0)$ an additional 100 points are rewarded. Also each ground contact of the legs is worth 10 points. In each frame the firing of the engines leads to a reduction of the reward of -0.3 per frame for the main engine and -0.03 for each of the side engines.
An episode ends when the rocket crashes meaning making contact with the surface with a velocity greater than $(0, 0)$ or it moves out of the viewpoint of the environment [OpenAI].
For this project the problem is regarded to be solved if an average reward of 200 points over at least 100 episodes is achieved.

3.2 Network

As described in the previous section the novelty of Dueling Network architectures is the separation of the advantage and value functions. This has to be implemented into a neural network. The following section describes the details of the implementation.
The paper which introduced the Dueling Network architecture originally intended to use it for pixel-based Reinforcement Learning tasks. The basis for this new architecture is a single Q-network architecture, therefore the lower layers of the Dueling Network consists of three `conv2D` layers followed by a `Flatten` layer for feature extraction. The output of these then serve as inputs for the aformentioned value and advantage streams. Both streams consist of two fully conected layers computing the values for $V(s)$ and $A(s, a_i)$. The last layer ouputting a single value for V and $\#actions$ values for A [Wang et al., 2016]. The graphic below depicts the described architecture:

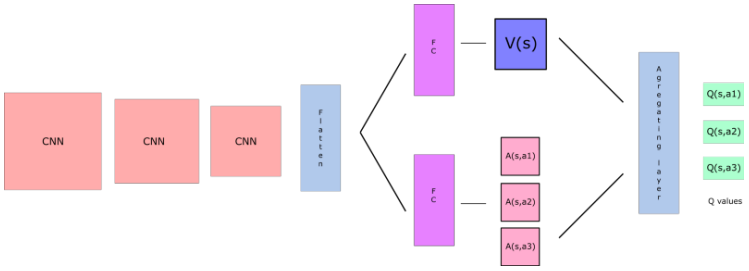


Figure 1: Architecture of a Dueling Network

As not every RL is necessarily pixel-based the proposed architecture might not be adequate or feasible for other tasks. So changes have to be made to adapt to the new task. This however only concerns the feature extraction part of the network. For the feature extraction any type of design can be used that is suitable for the task at hand. In the case of this project the convolutional layers where replaced by two fully connected layers to extract the features.
In more detail, the fully connected layers for feature extraction both consist of 128 units with an `relu` activation function. In both the value and advantage stream the number the number of units for the first fully connected layer where adjusted to 64 as the original paper intended to use 512.

3.3 Policy Learning

Using a model-free algorithm, like Q-Learning in this project, the agent relies on trial and error as it has no prior knowledge about the environment. The agent explores the environment and learns from the outcomes directly, without constructing an internal model. In the beginning the agent only knows the possible states and actions of the environment and gradually discovers the rewards and state transitions by exploration. The target here is to teach the agent how to behave under different circumstances or rather with actions to take. It can choose to select an action with an unknown outcome (exploration) or it can choose an action based on prior knowledge to get a good reward (exploitation). The action selection is based on an ϵ-greedy algorithm. With this selction process the agent uses both exploitation to take advantage of prior knowledge and exploration to look for new options. The ϵ-greedy approach chooses to explore with a probability of ϵ, thus selecting an action randomly and with a probibilty of $1 - \epsilon$ to exploit, i.e. select the action with the highest estimated reward [Sutton and Barto, 2018]. The aim is to have a good balance between exploration and exploitation, called exploration-exploitation-trade-off. In this project the ϵ-greedy algorithm was implemented with the use of an ϵ-decay. The aim is to exploit more optimal policies in later episodes, when a minimal value for $\epsilon = 0.01$ is reached. The initial value of ϵ was set to 0.6 with a decay rate of 0.99. This way the agent still has a high probabilty to explore in the early episode and afterwards exploit more from prior knowledge, but maintining a small probability of exploration.

3.4 Agent Training

To train the agent a Double DQN (DDQN) algorithm was used insted of a regular DQN algorithm. The advantage of the DDQN is that it solves the DQN's problem of overestimating the Q-values, meaning that the Q-values suggest a higher return for the agent than it will actually get [Hasselt et al.,

2016]. This is achieved by decoupling the action selection from the action evaluation. Therefore, the Bellman equation is changed to

$$Q(s, a; \theta) = r + \gamma Q(s', \arg\max_{a'_i} Q(s', a'; \theta); \theta') \tag{5}$$

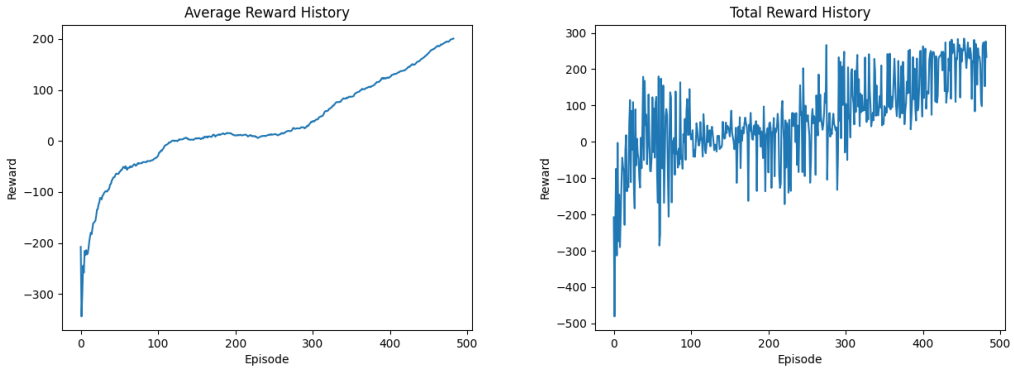
With this change the main neural network first chooses the best action a' among all possible actions and then the target network computes its Q-value to evaluate the action [Hasselt, 2010, Wang et al., 2016]. Through this overestimations can be reduced and result in improved final policies.
One of the main changes to the training process was to replace the loss function. The original paper intended to use the Mean Squared Error (MSE) for optimization, but it was replaced by the Huber loss. The Huber loss

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta, \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

is the balance between the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). It is less sensible to outliers and has also the advantage of being differentiable. This in theory makes it more robust and avoids exploding gradients, meaning that if the targets Q-values do not allign with the current ones, the gradient terms can rise to undiserable values, leading to alterations of the network weights that hinder the overall performance [Gokcesu and Gokcesu, 2021].
As for the optimizer employed to train the network, Adam with a learning rate of 0.0005 was chosen. Additionally required hyperparameters were the Experience Replay memory size, the minibatch size, the replacement frequency of the target network and the parameter τ for the softupdate. During training a minibatch of size 64 was sampled from 100000 experiences stored in the memory. They were sampled to perform a gradient descent step and update the target network every 4 steps with $\tau = 0.001$.

4 Results

In order to solve the problem, the agent has to reach two specific milestones during training. It first needs to learn to decrease the speed of the lander in order to hover and land safely, second to steer it on to the target. During several independent training runs these milestones occured everytime after a different amount of episodes and highly influenced the number of episodes needed to solve the game. The fastest model reached an average score of over 200 after only 419 episodes while the longest took 706, although having the exact same hyperparameters, but overall most of the trials concluded successfully between 400 and 500 episodes.



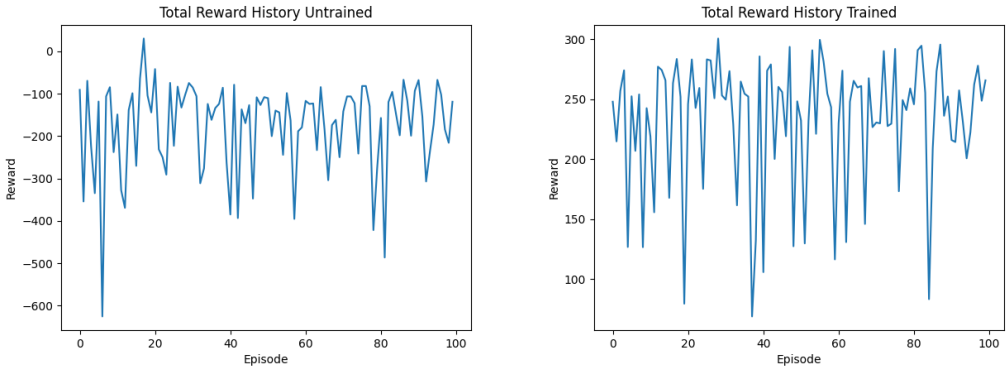
(a) Average Reward during Training

(b) Total reward history during training

Figure 2: Training history of the Dueling Network

Figure 2 shows the overall result of the training with regards to the average reward and total reward history concluding after 482 episodes. Looking at the Average Reward History one can observe that during the first 100 episodes there is a steady and steep increasement before plateauing around 0

for the next 200 episodes. After that the average reward grows almost linearly until it reaches the desired average reward of 200. When looking at these intervalls with regards to the total reward, figure 2 b) shows that during the first 100 there is a mixture of equally high positive and negative rewards. Between episodes 100 and 300 the total reward centered around zero, but with still a lot of high negative rewards. Afterwards the rewards per episodes increased drastically with most of the episodes scoring way above 0 and more importantly no more episodes with negative rewards were recorded. So regarding the two milestones described earlier one can arguebly say that the agent spent the first almost 300 learning not to crash the lander and afterwards until the end of the training process perfecting landing it on the target.



(a) Total reward history of an untrained agent with an average reward of -176.86

(b) Total average reward of a trained agent with an average reward of 233.58

Figure 3: Reward comparison of an untrained and trained agent

When comparing the results of the trained agent to a random agent, it shows that the training was successfull. With an average reward of 233 the agent performs reasonably well and most importantly no crashes (rewards close to 0) are recorded over a span of 100 episodes. But this has to be taken with a grain of salt, even if the average is over 200 figure 3 b) shows that there is still a high amount of fluctuation of the total rewards. There are still a number of episodes in which the agent scores between 100 and 150. This is not necessarily a bad sign as it still means that the lander landed safely without crashing, but an ideal result would be with less variance around the mean score. One can overlook this, considering the fact that with a mean score surpasing the 200 mark and most of the successful landings achieving rewards > 250.
When comparing it to result from implementations of simple DQNs, the Dueling DQN performs reasonably better. With convergence between 400 to 500 episodes it cuts down training by at least 100 episodes [1] and compared to some others only needed a fraction [2] [3].

5 Discussion

The advantage of the Dueling Network architecture lies in its ability to learn the state-value function efficiently. This leads to dramatic improvements compared to other deep Q-learning methods and drastically cutting down training times. However, during the implementation and training of the network several pitfalls were encountered. One of the obeservations was that this network architecture is very sensitive to hyperparameter changes. One relatively small change of one of the hyperparameters could completely influence the outcome drastically. While more common changes of hyperparameters like the learning rate or epsilon/epsilon decay rate only had more subtle effect like extending the number of episodes, other hyperparameter changes had more grave effects. For example, changing the replacement frequence from every four steps to five resulted a complete loss

[1] https://wingedsheep.com/lunar-lander-dqn/

[2] https://goodboychan.github.io/python/reinforcementlearning/pytorch/udacity/2021/05/07/DQN-LunarLander.html

[3] https://www.katnoria.com/nbdqnlunar/

of performance where the agent would plateau on certain average reward or even start to delearn.
Another observation that is interesting to point out is the dependency on the experience replay memory. Although this is more obvious as a larger memory size means less correlation of the samples and a more stable training, it was observed that when stored experiences didn't show a high variance the number of episodes needed for a successfull training was larger. This is mainly due to the fact that this type of memory only allows the agent to remember a certain number of previous episodes. If the episodes in the memory weren't that successfull, less progress could have been made. This could be fixed by the Prioritized Experience Replay (PER) memory, as proposed by the original paper. The key idea of the PER is to store experience based on the expected learning progress. That means that the experiences get weighted by their contribution to the improvement of the network and therefore experience tuples that are expected to lead to larger improvements with regards to the loss get sampled more frequently. For further/future experiments it would be interesting to implement this type of replay memory and analyze whether this can further improve the Dueling Network's performance.
The last thing to point out is the limitation of the number of steps per episode. The experiments have shown that by limiting the number of steps per episode stabilized the training process substantially. It has shown that whithout the limitation of the steps the agent would plateau and eventually delearn as soon as the ϵ has gotten small. The limitation not only stabilized the training process, but also sped it up. One can argue that this might be some sort of cheating as the limit could stop crashes prematurely, but it also goes the other way around. Successfull episodes could be ended early due to the limited step number and also exploration could be haltered. However, the limitation had mainly positive effects as it led to faster convergence and resulted in less fluctuations of the total episode rewards. A possible solution to counteract this limitation could be to adapt the memory size to a bigger stepsize, meaning that the memory could theoretically hold as many previous episodes with the default step size as with the limited one.
In summary one can say that there were a few challenges that had to be solved in order to successfully implement and train the Dueling Network architecture on the LunarLander environment. After solving these problems however the network showed very promising results and might even has some room of improvement by trying some of the afformentioned alternative solutions.

References

K. Gokcesu and H. Gokcesu. Generalized huber loss for robust learning and its efficient minimization for a robust statistics, 2021.

H. Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), mar 2016. doi: 10.1609/aaai.v30i1.10295.

OpenAI. Lunar lander. `https://www.gymlibrary.ml/environments/box2d/lunar_lander/`. Last Accessed: 12.07.2022.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.