Self-Supervised Dueling Networks for Deep Reinforcement Learning

Xiang Xu∗
Electrical and Computer Engineering
Carnegie Mellon University
xiangxu@andrew.cmu.edu

Paul Pu Liang∗
Computer Science Department
Carnegie Mellon University
pliang@andrew.cmu.edu

Abstract

Sparse rewards in the field of reinforcement learning limit the convergence speed of deep reinforcement learning algorithms. In this paper, we examine the possibilities of using self-supervised signals as auxiliary rewards. In order to better model the environment dynamics, we propose a 2 stream architecture: the controllable stream represents state features that our agent can interact with and modify while the uncontrollable stream represents the state features from the environment that our agent has no control of. To learn about the uncontrollable stream, we propose replay memory aggregation, a method to explicitly collect data about agent-independent environment simulations. Finally, our self-supervised signals include reward prediction, one-step lookahead best action prediction and next frame regularization. These simple and intuitive regularization terms that can be added to the loss function and are independent of the model chosen. We experiment with different settings and found that these self-supervised rewards can help improve the convergence speed and the final performance of the trained model.

1 Introduction

Recently, deep reinforcement learning has achieved outstanding performance thanks to new techniques such as experience replay and fixed target network that help to stabilize training. However, one issue still remains with DQN: it requires hundreds of hours of training time in order to match human performance. This is mainly due to the issues of sparse reward, small learning rate for updating network, and slowly updated target network. Currently, there are several new methods that try to speed up the training of DQN. One approach is to add more reward signals into the training phase by introducing auxiliary tasks (Shelhamer et al., 2016), (Jaderberg et al., 2016). By learning these additional tasks, the model can learn a better state representation. As pointed out by the author in (Shelhamer et al., 2016), learning a good state representation can be much harder than learning the state-action value. Another approach is to reuse previous memory by matching new state to old similar states (Pritzel et al., 2017). In this paper, we propose an alternative approach to auxiliary rewards using a 2 stream architecture: the controllable stream that represents state features that our agent can interact with and modify while the uncontrollable stream represents the state features from the environment that our agent has no control of.

The remainder of the paper is structured as follows. Section 2 described our motivation for proposing the controllable and uncontrollable streams. Section 3 reviews previous work on deep reinforcement learning, self-supervised learning and approaches to solve the problem of sparse rewards. Section 4 describes our proposed approaches and illustrates our model. We explain the experiments and results in section 5. We conclude our work in section 6 and propose avenues for future work in section 7.

∗indicates equal contribution, order determined by coin flips

2 Controllable and Uncontrollable Streams

To solve the problem of reward sparsity, we propose an alternative approach to auxiliary rewards using a 2 stream architecture:

1. The *controllable* stream represents state features that our agent can interact with and modify.
2. The *uncontrollable* stream represents the state features from the environment that our agent has no control of.

This separation allows the model to learn action dependent environment interactions and action independent environment dynamics. We also let the model learn to predict the future reward, action and state as extra supervised signals.

For example, in the game of Pong, the controllable stream may represent the position and velocity of the agent’s board while the uncontrollable stream may represent the position and velocity of the ball. By explicitly dividing the state into dueling streams, we want our agent can learn a better state representation, which is similar to the original dueling network (Wang et al., 2015). The difference is that instead of separating the action-independent advantage function from the action-dependent state value function, we actively separate the agent-controllable states from the agent-uncontrollable states.

Even though this seems similar to model-based reinforcement learning, our model does not explicitly use supervised learning to learn the state transition and reward function. Instead, we let our agent learn a good state representation by itself. Furthermore, the two stream approach allows us to use additional self-supervised rewards to further speed up training. We add the constraint that values of states which the agent has no control of should not change significantly for successive frames. This forces the model to learn action-independent state representations and helps reuse of such state representations. This can be seen as similar to the method described in (Pritzel et al., 2017) where the state representation from the CNN is used as a key for memory lookup.

3 Related Work

Deep Reinforcement Learning is a rapidly developing field, with advancements in deep Q-learning allowing computers to master a variety of games in the Atari Learning Environment (Mnih et al., 2013). More recently, double Q-learning (van Hasselt et al., 2015) and dueling networks (Wang et al., 2015) have been introduced to address problems that standard deep Q-learning faced. Double Q-learning aims to prevent overestimation of q-values (van Hasselt et al., 2015) and dueling networks aim to leverage the fact that q-values can be more accurately estimated by combining a state value function and an action-dependent advantage function (Wang et al., 2015).

However, it is well known that deep reinforcement learning still suffers from the problem of sparse rewards, and neither of these popular methods have been able to address this. Recent literature has approached this from several directions. The first direction has been to use a memory module to store past experiences (Pritzel et al., 2017), partially inspired by using a special memory module to store and learn quickly from rare events (Kaiser et al., 2017). Other directions are explained further in (Pritzel et al., 2017). As an overview, advances in exploration (Osband et al., 2016), hierarchical reinforcement learning (Vezhnevets et al., 2016) and transfer learning (Rusu et al., 2016), (Fernando et al., 2017) have also made significant progress towards improving reward efficiency in deep reinforcement learning algorithms.

Self-supervised learning has seen more attention from the computer vision community. Since there is much more unlabeled data than labeled data, self-supervised learning leverages on the spatial relationships between partitions of an image (Noroozi and Favaro, 2016), (Doersch et al., 2015) or the temporal relationship between frames of a video (Wang and Gupta, 2015), (Misra et al., 2016). These self-supervised methods define surrogate losses and synthesize the annotations from the data, allowing better feature extraction.

Since self-supervised methods make use of unannotated data, as auxiliary losses for RL they promise to mine more from the data already available to the policy (Shelhamer et al., 2016). However, to our knowledge, there is only 2 papers on applying self-supervised feature extraction methods for deep reinforcement learning. (Shelhamer et al., 2016) proposes a sequence of self-supervised feature extraction techniques to boost performance, such as learning dynamics and inverse dynamics, image
reconstruction and reward binning. Jaderberg et al. (2016) defines auxiliary reward tasks such as pixel control and value function replay.

Our paper aims to improve upon their work and incorporate self-supervision into a dueling network architecture by using next frame regularization as an auxiliary reward signal and feature extractor. While Oh et al. (2015) also attempts to predict next frames, they do not do so in the context of improving reinforcement learning algorithms with auxiliary reward signals. Therefore, our work is the first to combine self-supervised next frame regularization to learn environment dynamics in a reinforcement learning setting. Our method includes simple and flexible regularization terms that can be added to the loss function independent of the model chosen.

4 Methods

4.1 Dueling Networks

For our model, we divide the state representation into two different streams. It is natural for us to follow a similar model design as the Dueling DQN (Wang et al., 2015). In the Dueling DQN, the state-action value function is separated into an estimation of the state value function as well as an action advantage function. The predicted $q$-values is then a function of the state-value function $V(\cdot)$ and the action advantage function $A(\cdot, \cdot)$:

$$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a'))$$

(Wang et al., 2015) showed that by dividing the action-independent estimation from the action-dependent estimation using two streams, the model can learn to focus on different parts of the image. This generally results in improvement over the original DQN.

Building on top of the Dueling DQN architecture, we will be adding additional streams after the convolutional layers to represent auxiliary predictions and rewards. The remainder of this section will cover our proposed self-supervised rewards and end with our final model architecture, which we call the Self-Supervised Dueling DQN.

4.2 Reward Prediction

The first self-supervised task is reward prediction, a regression problem. We define an auxiliary stream $M_r$ on top of the last hidden layer of Dueling DQN $M_{DQN}$. $M_r$ estimates the reward $\hat{r}$ obtained from being at state $s$ and taking action $a$. Denoting $f_r$ as the feedforward function, this regression loss term is as follows:

$$\hat{r}_i = f_r(s,a; M_r, M_{DQN})$$

$$L_r = \frac{1}{N} \sum_i \|r_i - \hat{r}_i\|^2_2$$

for $(s,a,r,s',t) \in D_1$.

We also experimented with predicting a binary classification of whether a reward is received or not by taking action $a$ from state $s$, although we settled on reward regression since this task is more difficult and Shelhamer et al. (2016) already employs a similar reward classification task.

4.3 One-step Lookahead Best Action Prediction

The second self-supervised task is to predict the future best action, a classification problem. We define an auxiliary network $M_a$ on top of the last hidden layer of Dueling DQN $M_{DQN}$. $M_a$ estimates the best action $\hat{a'}$ to choose in the next state $s'$ when the agent is at state $s$. This method can be viewed as a one step lookahead of what the next best action $\hat{a'}$ will be in state $s'$ after the model transitions to state $s'$ from $s$ using its own predicted action $a$. Denoting $f_a$ as the feedforward function, the cross-entropy loss term for classification is:

$$\hat{a}'_i = f_a(s; M_a, M_{DQN})$$
\[ L_a = \frac{1}{N} \sum_i -a'_{i} \log(\hat{a}'_{i}) \]

for \((s, r, s', t) \in D_1\) and where \(a' = \arg \max_a Q(s', a)\).

Note that it is important to take a one-step lookahead for best action prediction. If we do not, we are simply predicting the best action from the current state \(s\), which is exactly what our predicted \(q\)-values are already doing. As a result the regularization would not achieve any additional effect.

### 4.4 Replay Memory Aggregation

We use \(a = -1\) to indicate no action taken by the agent.

In the standard replay memory \(D_1\), each \((s, a, r, s', t) \in D_1\) represents action dependent state transitions. We note that the replay memory is of fixed size and is aggregated by the DQN’s replay simulations. Since the exploration \(\epsilon\) decays, the \((s, a, r, s', t)\) data points in the replay memory represent increasingly better state-action transitions as training proceeds. These reasons make the replay memory suitable as a dataset for training a DQN. However, this is insufficient for modeling agent-independent environment dynamics. We inspect that trained models almost never choose \(a = -1\) as the best action to take. As a result, the replay memory does not contain enough data points for us to learn agent-independent environment dynamics.

To solve this problem, we propose a second replay memory \(D_2\), we sample random states and observe the action independent state transitions. We only have datapoints of the form \((s, a = -1, r, s', t) \in D_2\), where \(a = -1\) indicates no action taken. This is important for modeling the uncontrollable stream where the agent does not interact with the environment.

We found that this currently only works for games in the OpenAI environment that provides an option for the agent not taking an action. This was possible with the 3 environments we tested on, which were Pong, SpaceInvaders and Breakout. In each of these games, setting \(a = 0\) represents no action taken by the agent (although we denote \(a = -1\) as no action taken for clarity). Figure 1 below shows a sample sequence of the environment simulation for the game of Pong when the agent does not take any action.

![Figure 1: Pong environment simulation with no actions taken by the agent.](image)

Aggregating the replay memory gives us more data to learn features for the uncontrollable stream in modeling agent-independent environment dynamics. These dynamics include natural movements of the bullets in SpaceInvaders or the ball and opponent in Pong. Pong is also an interesting example: in reference to figure [1] we see that the movement of the opponent (in orange) is also part of the agent-independent environment dynamics. By focusing on these dynamics in the uncontrollable stream, the model could possibly treat the opponent as an expert and learn the opponent’s movement relative to the ball.

For more figures of agent-independent environment simulations for other environments, please refer to the Appendix, section [8.1].

### 4.5 Next Frame Regularization

We enforce frames close to each other with no action taken and no reward in between to have similar \(Q(s, a)\). Else, we compute \(Q(s, a)\) using the Bellman equation. This can be seen as dividing
the expected reward function into what will happen if no action is taken and what will happen is an action is taken. State transitions without any action taken in between represents the *uncontrollable* stream: the intuition is that if no action is taken and no significant reward is obtained, then two frames close to each other should have similar expected rewards. This loss term is as follows:

\[ L_f = \| Q(s, a = -1) - \gamma \max_{a'} Q(s', a') \|^2 \]

for \((s, a = -1, r = \epsilon, s', t) \in D_2\), the aggregated replay memory.

There are 2 subtleties in the formulation of this loss term.

Firstly, regularization should only be applied when the rewards involved in the state transition is small. Therefore, we only select data points \((s, a = -1, r = \epsilon, s', t) \in D_2\), the aggregated replay memory. This is because huge changes in rewards, such as the case when the agent does not take an action but loses a life in Pong, leads to drastically different predicted \(Q(s, a = -1)\) and \(\max_{a'} Q(s', a')\).

As a result, we only enforce \(Q(s, a = -1)\) and \(\max_{a'} Q(s', a')\) to be similar when the intermediate reward obtained in transiting between \(s\) and \(s'\) is small.

Secondly, note the multiplicative discount factor of \(\gamma\) in the loss term. This is because:

\[ Q(s, a = -1) = r + \gamma \max_{a'} Q(s', a') \quad \text{for } (s, a = -1, r = \epsilon, s', t) \in D_2 \]

\[ \approx \gamma \max_{a'} Q(s', a') \quad \text{when } r = \epsilon \text{ is small} \]

Therefore it is reasonable to impose that \(\gamma \max_{a'} Q(s', a')\) and \(Q(s, a = -1)\) are regularized to be similar in \(L_f\). Although this makes the regularization term is very similar to the standard DQN loss function, we believe that this will help since state transitions when \(a = -1\) are very rare. By boosting their occurrence with the aggregated replay memory \(D_2\) and weighting their effect on model training appropriately, we are achieving a similar effect to (Graves et al., 2016) and (Ba et al., 2016) in learning from rare events.

### 4.6 Self-supervised Dueling DQN Architecture

Using these loss functions and additional streams, figure 2 shows the final Self-supervised Dueling DQN architecture:

![Figure 2: Final Self-supervised Dueling DQN architecture with additional streams after the convolutional layers. \(M_r\) is the stream that predicts reward \(\hat{r}\) and \(M_a\) is the stream that predicts one step lookahead best action \(\hat{a}'\). The original streams that predict \(V(s)\) and \(A(s, a)\) remain.](image-url)
4.7 Loss Function

Suppose we represent \( Q(s, a) \) with a Dueling DQN \( \mathcal{M}_{DQN} \) with parameters \( \theta \). The final loss function is defined as follows:

\[
y_{i}^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^{-})
\]

\[
L_{i}(\theta_{i}) = \mathbb{E}_{s,a,r,s'} \left[ (y_{i}^{DQN} - Q(s, a; \theta_{i}))^2 \right] + \lambda_{\{r,a,f\}} \mathcal{L}_{\{r,a,f\}}
\]

where \( \theta^{-} \) represents the parameters of a fixed and separate target network. \( \mathcal{L}_{\{r,a,f\}} \) are the regularization loss functions for reward prediction, best action prediction and next frame regularization as specified above. \( \lambda_{\{r,a,f\}} \) are regularization hyperparameters.

4.8 Training Details

We follow the original training procedure for Dueling DQN. The loss terms \( \mathcal{L}_{\{r,a\}} \) are easily added based on the outputs of the Self-supervised Dueling DQN’s additional streams \( \mathcal{M}_r \) and \( \mathcal{M}_a \) respectively.

When we perform next frame regularization \( \mathcal{L}_f \), we store 2 replay memories. During the sampling phase, we sample episodes \( s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T \) from our model’s policy \( \pi(\cdot|\cdot, \theta) \) and add this to the standard replay memory \( D_1 \). We also sample episodes \( s_0, a_0 = -1, r_1, s_1, a_1 = -1, r_2, \ldots, s_{T-1}, a_{T-1} = -1, r_T \) and add this to the aggregated replay memory \( D_2 \).

5 Experiments

5.1 Datasets

We performed experiments Breakout and SpaceInvader games in the Atari Learning Environment.

5.2 Results

Here are the results using a dueling deep convolutional network with target fixing and with experience replay. The experimental setup followed (Mnih et al., 2013) in terms of data preprocessing, replay memory and optimization. We followed (Wang et al., 2015) as closely as possible for the Dueling DQN architecture.

Hyperparameters for our aggregated replay memory \( D_2 \):

1. Num Burn-In = 50000
2. Replay memory size = 500000

Architecture hyperparameters for our additional model stream for reward prediction (on top of the last convolutional layer of Dueling DQN):

1. A dense layer with 512 output units and ReLU activation.
2. A dense layer with 64 output units and ReLU activation.
3. A dense layer with 1 output unit and no activation.

Architecture hyperparameters for our additional model stream for one step lookahead best action prediction (on top of the last convolutional layer of Dueling DQN):

1. A dense layer with 512 output units and ReLU activation.
2. A dense layer with the output dimension equal to the number of actions and softmax activation.

For the full set of standard hyperparameters chosen, please refer to the Appendix, section 8.2.

The first experiment involves implementing reward prediction and best action prediction on top of the baseline dueling network model. The results are shown in figure 5.
From this we can see that reward prediction improves the agent’s learning. We observe consistent improvement in average reward by including these self-supervised signals. We also note that the values of \( \lambda_{\{r,a\}} \) should be chosen as small values since the main objective of the model has to be trained with the standard loss function to predict the best \( q \)-values. When larger values of \( \lambda_{\{r,a\}} \) were used, the performance decreased as compared to small values of \( \lambda_{\{r,a\}} \). Nevertheless, we found this to be an easy way to improve model performance without any increase in model complexity and training time.

The second experiment involves implementing next frame regularization on top of the baseline dueling network model. Results are shown in figure 4.

Once again, we observe slight improvement in average reward by including this self-supervised signal. We had to ensure that the only regularized next frame samples were drawn from \((s,a=-1, r=\epsilon, s', t) \in D_2 \) where \( r = \epsilon \) was important. In addition, it was important to add the multiplicative discount factor \( \gamma \) into the loss term \( L_f \) during training. Otherwise, the model was not able to obtain rewards that were better than the baseline.

Unfortunately, the improvement in agent training and reward obtained was not as significant as we expected. One possible reason is that the added regularization term is not very significant, since our regularization term is:

\[
L_f = \|Q(s, a = -1) - \gamma \max_{a'} Q(s', a')\|_2^2
\]

while the standard loss function can be rewritten as:

\[
L = \|Q(s, a) - r - \gamma \max_{a'} Q(s', a')\|_2^2
\]

In particular when \( r = 0 \), the 2 losses are equivalent. Therefore, the difference lies when \( r > 0 \) and when we simulate \( a = -1 \), which is almost never chosen in the standard DQN models.
For future improvements, it will be interesting to explore how many state inputs we want to constraint to have similar $q$-value functions when no action is taken. As of now we are using two ($s$ and $s'$) since we can easily sample $(s, a = -1, r = \epsilon, s', t)$ from the aggregated replay memory $D_2$. However it will be interesting to experiment with different temporal ranges. This will be similar to an $k$-step DQN rather than one-step DQN. Providing stronger regularization will likely have a greater impact on performance, probably at the expense of training time since we will have to evaluate and regularize $k$ successive states rather than just 2.

6 Conclusion

In this paper, we tackled the problem of sparse rewards in the field of reinforcement learning by examining the use of self-supervised learning as auxiliary reward signals. In order to better model the environment dynamics, we proposed a 2 stream architecture: the controllable stream represents state features that our agent can interact with and modify while the uncontrollable stream represents the state features from the environment that our agent has no control of.

To learn about the uncontrollable stream, we proposed replay memory aggregation, a simple method that explicitly adds agent-independent environment simulations into the replay memory. Finally, our self-supervised signals include reward prediction, one-step lookahead best action prediction and next frame regularization. These simple and intuitive regularization terms can be easily added to the loss function and is independent of the model chosen. We experimented with several environments in the Atari Learning Environment and found that such self-supervised rewards helps to improve the convergence speed and agent performance.

7 Discussion and Future Work

We found that adding auxiliary rewards indeed improves performance, agreeing with (Shelhamer et al., 2016) and (Jaderberg et al., 2016) although we used different methods. However, when auxiliary rewards are included, it is still important to weight the standard DQN loss term much higher than the regularization loss terms.

One limitation of our work is that it requires the ability to simulate the environment when the agent does not take any actions. There are only a few games in the OpenAI gym environment that supports this. As an extension, we could train and test on a wider variety of environments, especially the ones tested on in (Shelhamer et al., 2016) and (Jaderberg et al., 2016) since these are the only papers that incorporate self-supervised auxiliary rewards into deep reinforcement learning agents. Note that some of these environments may not be part of the Atari Learning Environment.

Furthermore, we have not fully leveraged the temporal dynamics of the individual frames in the Atari Environment simulations. We could try an idea similar to (Misra et al., 2016), where we leverage on the temporal dynamics between frames as a form of self-supervision. Experimenting with different temporal ranges (mentioned in section 5.2) could also help improve performance by providing a stronger form of regularization across the next $k$ states rather than just 2 states.

Finally, an interesting idea cropped up when we were visualizing agent-independent environment simulations in the standard environments. In reference to figure 1 in the Pong environment, we see that the movement of the opponent (in orange) is also part of the agent-independent environment dynamics. By focusing on these dynamics in the uncontrollable stream, the model could possibly treat the opponent as an expert and learn the opponent’s movement relative to the ball. It could be promising to model the opponent explicitly and train the agent in 2 ways: train it with a DQN and train it by observing the opponent’s movements in the aggregated replay memory $D_2$ and learn from the opponent (expert).
References


8 Appendix

8.1 Replay Memory Aggregation

Figure 5 and figure 6 are more examples of agent-independent environment simulations that we add to the aggregated replay memory $D_2$. We can see the movement of the ball in Breakout as well as how it bounces off walls, and the movement of the bullets in SpaceInvaders. These are part of the uncontrollable stream and helps the model to learn agent-independent environment dynamics.

![Figure 5: Breakout environment simulation with no action taken.](image1)

![Figure 6: SpaceInvaders environment simulation with no action taken.](image2)

8.2 Training Hyperparameters

Here is the complete list of standard training hyperparameters that were present in the Dueling DQN paper (Wang et al., 2015), so they are not unique to our paper. The majority of these hyperparameters are set as the default in the paper except for scaling it down by a factor due to time and computational constraints.

Optimization:

1. Optimizer = Adam
2. Learning Rate = 0.00025
3. Decay = 0.00002
4. Gradient clipping = [-1.0,1.0]
5. Num Iterations = 1000000
6. Batch size = 32

Memory:

1. Num Burn-In = 50000
2. Replay memory size = 500000

Environment:

1. $\gamma = 0.99$
2. Testing $\epsilon = 0.05$
3. Training initial $\epsilon = 1.0$
4. Training final $\epsilon = 0.1$
5. Training $\epsilon$ decay length = 1000000

Training:
1. Training frequency = 4
2. Target update frequency = 10000
3. Target update method = hard

Evaluation:
1. Evaluate frequency = 10000
2. Num eval episodes = 20
3. Max episode length = 1000000