Effect of State Presentation on Deep Q-Learning Performance

Jacob Senecal JACOBSENECAL@YAHOO.COM
Gianforte School of Computing
Montana State University
Bozeman MT, USA

Rohan Khante HAPPYKHANTE@GMAIL.COM
Gianforte School of Computing
Montana State University
Bozeman MT, USA

Greg Hess GREGORY.HESS@MONTANA.EDU
Gianforte School of Computing
Montana State University
Bozeman MT, USA

Abstract

We conduct experiments using difference frames as an alternative to stacked sequential frames for the environment state representation within the context of applying Deep Q-Learning to Atari 2600 games. We show that depending on the complexity and number of objects in a typical game frame, an agent can achieve reasonable performance while using difference frames as a state representation, while reducing computational requirements compared to using stacked sequential frames.

Keywords: Reinforcement Learning, Q-Learning, Atari, Neural Network

1. Introduction

Reinforcement learning is learning what to do, or how to map situations to actions, so as to maximize a numerical reward signal. The agent is not told which actions to take, but instead must discover which actions yield the most reward by trying them (Sutton and Barto, 2011). Sutton and Barto further state that reinforcement learning can be formalized using ideas from dynamical systems theory. Namely, that reinforcement learning is essentially the problem of learning optimal control of incompletely known Markov decision processes.

A Markov decision process is a way of formulating a sequential decision making process, where actions influence subsequent situations and states, as well as immediate and future rewards. Within reinforcement learning Markov decision processes are used to frame the problem of learning from interaction to achieve a goal. The learner and decision maker is called the agent, and the thing it interacts with, comprising everything outside the agent, is called the environment (Sutton and Barto, 2011). The agent and the environment interact during the learning process, giving rise to state transitions, and rewards which depend on the "goodness" of actions. The agent-environment interaction is represented in figure 1.

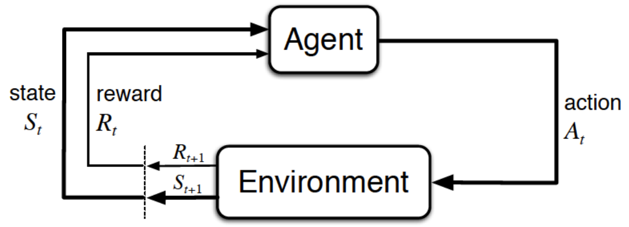

Figure 1: Agent-environment interaction in a MDP (Sutton and Barto, 2011).

Q-learning is a reinforcement learning algorithm proposed in 1989 by Chris Watkins as a method for learning control within the context of a Markov decision process. Q-learning seeks to determine the value or quality of actions that an agent can take, given some input state (Watkins, 1989). The Q-learning update rule is defined as,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha * [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{1}$$

where S_t and A_t are the current state and action, respectively, S_{t+1} is the next state resulting from the chosen action, R is the reward, α is a learning rate between 0 and 1, and γ is the reward discount factor, also a value between 0 and 1.

If the Q-learning policy is optimal then an agent would simply choose the action with the highest Q-value for the current state. The goal of the Q-learning training algorithm is to develop an approximation for the optimal Q-function.

In this study we apply Q-learning to train a reinforcement learning agent to play games in the Atari 2600 domain. In particular, we apply what is known as deep Q-learning. Deep Q-learning uses a convolutional neural network to create a mapping between environment states, and the Q-values associated with the possible actions an agent can take in the environment. In this study we examine the possibility of using a simplified environment state representation compared to previous implementations of deep Q-learning, which can reduce computational requirements, while maintaining adequate agent performance on a subset of Atari 2600 games.

2. Deep Q-Learning

We consider a task in which the reinforcement learning agent is interacting with an environment produced by an Atari emulator. At each time step in the environment the Atari emulator provides the agent with a discrete number of actions to choose from. For example, in the classic game "Pong" the actions include up, down, or no action. A choice of say, up, would result in the paddle (in the case of Pong) moving a fixed distance upward, the distance being constant, and set by the emulator.

As mentioned briefly earlier, Q-learning attempts to learn the value of actions that an agent can take given some input state. This implies that there is some form of mapping or representation that associates Q-values with state action pairs. In early implementations of Q-learning applied to simple gridworld environments, the state and action spaces were small enough that state-action pairs could simply be represented as a 2D array. However, in the case of Atari games the environment state is the current game frame image produced by the emulator. Working directly with raw Atari frames, which are 210 x 160 pixel images with a 128 color palette, represented as a RGB vector, means that there are 210 * 160 * (128 * 128 * 128) $\approx$ 7e10 possible states. It is not feasible to represent this number of states as an array. Furthermore, a single game frame alone is not sufficient to fully represent the Atari environment state. A single game frame cannot represent vital environment information like velocity. So, to fully represent environment state in the Atari domain, stacked sequential emulator frames have been used as the state representation in past studies.

Due to the size and complexity of the state space, in lieu of an array, a deep convolutional neural network is used as a function approximator to process the high dimensional state representation and map a state input to the Q-values of the discrete number of actions that an agent can take within the Atari emulator. In order to train a neural network with weights θ, we must slightly modify equation 1 to produce the loss function at each training iteration t,

$$L_t(\theta_t) = (y_t - Q(s_t, a; \theta_t))^2 \tag{2}$$

$$y_t = r + \gamma \max_a Q(s_{t+1}, a; \theta_t) \tag{3}$$

Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\nabla_{\theta_t} L_t(\theta_t) = (r + \gamma \max_{a'} Q(s_t, a; \theta_{t-1}) - Q(s, a; \theta_t))\nabla_{\theta_t} Q(s, a; \theta_t) \tag{4}$$

which is used to optimize the neural network weights via gradient descent. The original deep Q-learning algorithm is presented in algorithm 1.

Algorithm 1 Deep Q-Learning

```
Initiate replay memory D to capacity N
Initialize action-value function Q with random weights

for episode = 1 to M do
    Initialize frame sequence $s_1 = \{x_1\}$ and preprocessed frame sequence $\phi_1 = \phi(s_1)$
    for t = 1 to T do
        With probability $\epsilon$ select an action $a_t$ otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_t + 1$
        Set $s_t + 1 = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_t + 1)$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 4
    end for
end for
```

3. State Representation

Experience replay is a key aspect of the deep Q-learning algorithm, and something all implementations of deep Q-learning have in common (Beaulieu et al., 2018). In the deep Q-learning algorithm a set of past experiences is maintained. An experience consists of a 4-tuple (s, a, r, s'); state (a sequence of game frames), action, reward, next state (a sequence of game frames) . During training, batches of experiences are randomly sampled from the replay memory to update the neural network.

Experience replay greatly reduces the likelihood of highly correlated states being presented in order as network weights are updated during training. Also, by keeping the experiences we draw random, we prevent the network from only learning about what it is immediately doing in the environment, and allow it to learn from a more varied array of past experiences, preventing what is known as "catastrophic forgetting".

Experience replay is essential to achieve success with deep Q-learning, but it has a high memory footprint. Within the context of the Atari 2600 domain it is common to use experience replay buffers that store $\approx 1,000,000$ experiences. Recall that an experience consists of the tuple (s, a, r, s') where the states are made up of stacked game frames. Even after resizing the stacked game frames to (84 x 84 x 4) and converting them to grayscale values cast to uint8 datatypes (unsigned integer 0 to 255), a buffer that stores 1,000,000 experiences will use $\approx$ 8 gigabytes of memory in our preliminary experiments, using four stacked sequential frames to represent game state.

All of the studies presented in the literature review, and indeed all studies we have read regarding deep Q-learning applied to the Atari 2600 domain, have used stacked sequential frames to represent the game state. Typically four sequential frames are used following the original study by Mnih et al., which stated that the deep Q-learning algorithm was also robust to three or five stacked frames, but there doesn't appear to be any systematic study identifying an optimal number of stacked frames. Hausknecht and Stone even had success stacking 10 frames while applying the deep Q-learning algorithm to partially observable Atari games.

3.1 Difference Frames

Instead of using stacked sequential frames to represent game state, we propose using difference frames. A difference frame is created by subtracting two sequential frames.

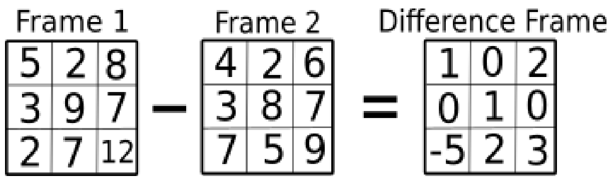

Figure 2: Difference frame creation. Frames are just arrays of pixel values. A difference frame is created through element-wise substraction of an earlier frame from a subsequent frame.

Recall that stacked sequential frames are used to represent game state so that information like velocity can be inferred from the state. We believe that a single difference frame can capture much of the same type of information (e.g. velocity), as four stacked sequential frames. If a single difference frame can indeed capture the same information as four stacked sequential frames this can make it possible to reduce the size of the experience replay buffer, lowering memory requirements. In general reducing computational resource requirements is always useful, but it is easy to imagine specific situations where reduced memory usage would be particularly beneficial, such as in small robotics or embedded systems, as well as hand-held and portable devices.

Additionally, using difference frames to represent game state allows the dimensionality of the neural network to be reduced, as the input to the network is an (84 x 84) array, rather than an (84 x 84 x n) array, n being the number of frames when using stacked sequential frames.

4. Related Work

Mnih et al. published "Human Level Control through Deep Reinforcement Learning" in 2012, describing what they called a "deep Q-network" agent, that could learn successful policies directly from high-dimensional sensory inputs, using end-to-end reinforcement learning. Their agent was tested on 49 different Atari 2600 games. The authors showed that using the same network architecture and hyperparameters, the agent could achieve competency in a variety of games in the Atari 2600 domain.

Subsequent studies produced improved network architectures and performance. Double Q-learning was developed by Van Hasselt et al., in an attempt control overestimation of Q-values under certain conditions. In standard deep Q-learning the max operator in equation 3 uses the same value both to select and evaluate an action. This makes it more likely to select overestimated values, resulting in over-optimistic value estimates (Van Hasselt et al., 2016). The double Q-learning target is then re-written as,

$$y_t = r + \gamma Q(s_{t+1}, \mathrm{argmax}_a Q(s_{t+1}, a; \theta_t); \theta'_t) \tag{5}$$

In equation 5 note the addition of the θ'_t from the original target equation. In double Q-learning we maintain a second neural network that we call the target network. The target network, with parameters θ', is the same as the online network except that its parameters are copied every τ steps from the online network, so that then $\theta'_t = \theta_t$, and kept fixed on all other steps. Doing this provides a more stable target and untangles action selection and evaluation, by using the online network to select the action and the target network to evaluate the action.

Wang et al. developed dueling network Q-learning, which produced more stable training and improved policy evaluation on the Atari 2600 domain. The dueling network architecture essentially separates value and action advantage functions into two output streams from the same network. The authors state that the dueling network automatically produces separate estimates of the state value function and advantage function, without any extra supervision. The authors state that the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state. This is particularly useful in states where its actions do not affect the environment in any relevant way.

The previously mentioned studies assumed fully observable Markov decision processes. Hausknecht and Stone advanced an adaptation of the deep Q-learning algorithm to handle partially observable Markov decision processes (POMDP). In previous deep Q-learning research the entirety of the current state information was available to the agent. For the Atari 2600 domain, the game state is typically represented as four sequential game frames stacked together to allow an agent to infer information such as velocity in the game. In the study by Hausknecht and Stone which also focused on the Atari 2600 domain, the authors modified the games so that at each timestep, the screen is either fully revealed or fully obscured with probability $p = 0.5$. Obscuring frames in this manner probabilistically induces an incomplete memory of observations needed for games to become a POMDP. To overcome the partially observable state the authors used what is known as a long short term memory network (LSTM) stacked on top of the original convolutional neural net architecture developed by (Mnih et al., 2015). LSTM's preserve some information between training inputs, which Hausknecht and Stone leveraged to produce an agent that could function with partially observed states. This paper also studied stacking up to 10 frames to represent the game state, in an additional attempt to overcome the lack of information arising from incomplete observations. The authors had some success with this method, though it did not achieve as high of performance as the LSTM augmented network.

5. Experiments

We conducted experiments to assess if a single difference frame could represent as much state information as multiple stacked sequential frames. If this is the case we would expect an agent trained using difference frames to better maintain performance, as measured by game score, as the number of frames available in the experience replay buffer is reduced.

If difference frames provide a more efficient state representation we expected to see a reduction in training time in terms of number of training epochs to convergence. Since the use of difference frames also allowed us to reduce the number of network parameters by a factor of four compared to using stacked sequential frames as a state representation, we expected to see a reduction in wall clock runtime, although this can be an inconsistent measure due to varying loads on a machine, etc.

5.1 Hypotheses

With the experiments we are testing two primary hypotheses.

1. Using difference frames to represent environment state will result in an agent capable of achieving a mean game score equal to or higher than an agent trained using stacked sequential frames.

2. Using difference frames to represent environment state will provide a more efficient state representation, as measured by number of training epochs to convergence.

We also track wall clock training time as we expect using difference frames will result in lower computational requirements, although wall clock time can be an inconsistent measure.

5.2 Experimental Design

We performed three sets of experiments varying the experience replay buffer capacity across three sizes, 1,000,000, 250,000, and 50,000 frames. At each buffer size we trained the agent on two games, Pong and Space Invaders, since we lacked the time to run experiments on a comprehensive set of Atari 2600 games, as has been done in past studies. We chose these games due to the diverse game elements they provide. Pong requires an agent to precisely hit a moving object, while Space Invaders requires an agent to shoot accurately at "aliens", maneuver underneath obstacles, and avoid lasers from the aliens.

We used an Atari emulator provided by the organization OpenAI that is designed specifically for reinforcement learning research (Brockman et al., 2016). The emulator provides a Python API as an interface to retrieve game frames and scores. We added additional preprocessing steps to resize game frames to 84 x 84 pixels, and converted frames to grayscale.

For each buffer size and game we ran five experiments, and tracked training time in terms of training epochs (an implementation independent measure) and wall clock time. A training epoch consists of 2500 minibatch weight updates. For our performance measure we tracked game score at the end of each episode.

Initial parameter tuning was conducted using a limited grid search where the range of parameter values tested was loosely based on parameter values used in past studies. We avoided an exhaustive parameter search due to the training time required for each experimental run. Also, as we are comparing relative performance of using stacked sequential frames vs. difference frames, extensive tuning of parameters to achieve the maximum score possible is not necessary. After initial parameter tuning we kept hyperparameters constant across all experimental runs, including when an agent was trained on a different game or with a new experience replay buffer size.

5.2.1 Network Architecture

Our network architecture is identical to the original architecture used in (Mnih et al., 2015). As described by Mnih et al. the input to the neural network consists of an $84 \times 84 \times n$ image produced by the preprocessing step, where n is the number of stacked frames. The first hidden layer convolves 16 8×8 filters with stride 4 with the input image and applies a nonlinear rectified unit (ReLU) (Nair and Hinton, 2010). The second hidden layer convolves 32 4×4 filters with stride 2, again followed by a rectifier nonlinearity. The final hidden layer is fully-connected and consists of 256 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action that can be taken by an agent within the emulator.

5.3 Results

In every case, using stacked sequential frames to represent game state resulted in higher maximum game scores compared to using difference frames as a state representation. The difference in game score in Pong wasn't particularly large, as using stacked sequential frames resulted in an average score differential of 6.4 points vs using difference frames, across the three experience replay buffer sizes. For Space Invaders, however the performance gap was significant. Figure 3 displays learning curves with a buffer size of 1,000,000 frames. Figure

4 shows learning curves with a buffer size of 250,000 frames, while figure 5 displays learning curves with a buffer size of 50,000 frames.

In figure 5, we see an interesting development in the graph depicting the agent's performance on Pong. We observe that the sequential frame learning curve appears to converge, but then performance drops off significantly, falling below the game score achieved using difference frames.

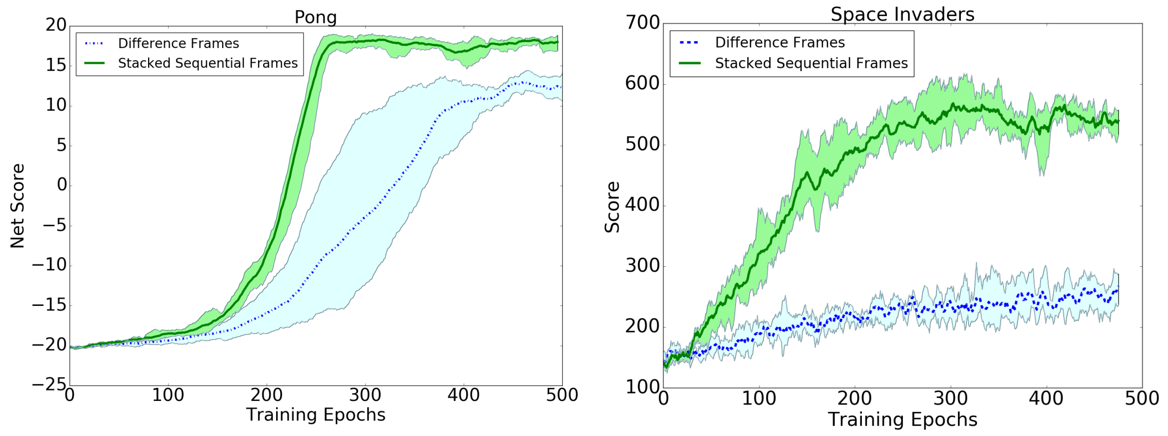

Figure 3: Agent learning curve with an experience replay buffer of 1,000,000 frames. The lighter green and blue bands indicate the range of maximum and minimum scores, while the solid line is the mean score.

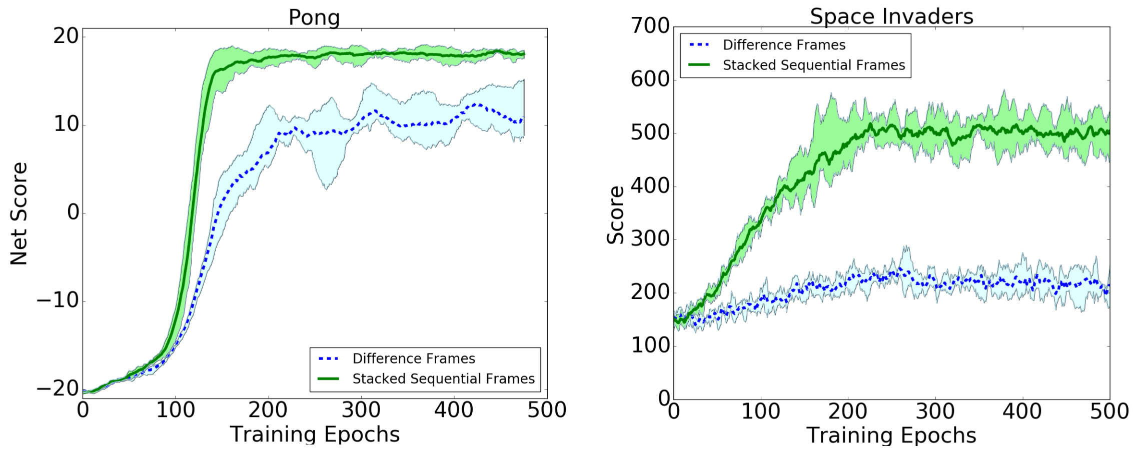

Figure 4: Agent learning curve with an experience replay buffer of 250,000 frames. The lighter green and blue bands indicate the range of maximum and minimum scores, while the solid line is the mean score.

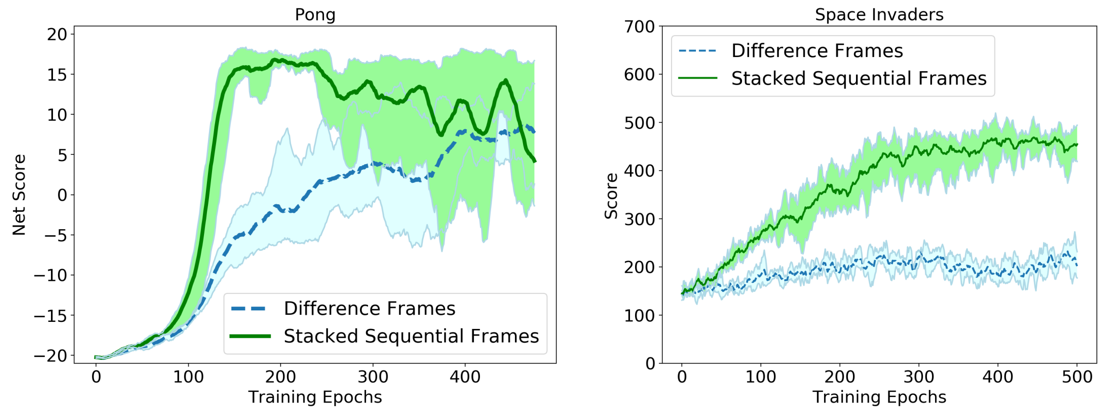

Figure 5: Agent learning curve with an experience replay buffer of 50,000 frames. The lighter green and blue bands indicate the range of maximum and minimum scores, while the solid line is the mean score.

In figures 6 and 7 we present the reductions in maximum game score achieved by the agent as the experience replay buffer is reduced. Recall that if it was true that a single difference frame is able to represent as much information as multiple sequential frames, then we expected reductions in agent performance when the experience replay buffer size is reduced to be more severe when using sequential frames, as more sequential frames would be required to store the same amount of "memory" as a smaller number of difference frames. We observed mixed results in these experiments. Our Space Invader tests showed that using difference frames does result in smaller reductions in maximum score as the experience replay buffer size is reduced, but on Pong using stacked sequential frames allowed the agent to maintain better performance.

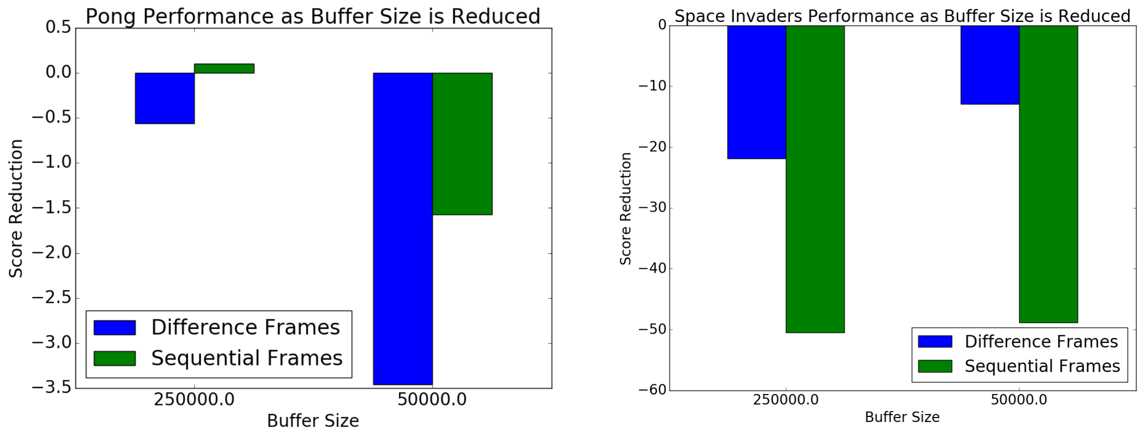

Figure 6: Change in score for different experience replay buffer sizes. Change in score is calculated using the score achieved with a buffer of 1,000,000 frames as a baseline.

As expected due to the reduced network complexity required to process difference frames compared to stacked sequential frames we observed reductions in wall clock runtime for both Pong and Space Invaders. All tests were ran on a GTX 1080TI GPU. Training times were calculated for 5,000,000 algorithm iterations in the case of Pong, and 10,000,000 algorithm iterations in the case of Space Invaders.

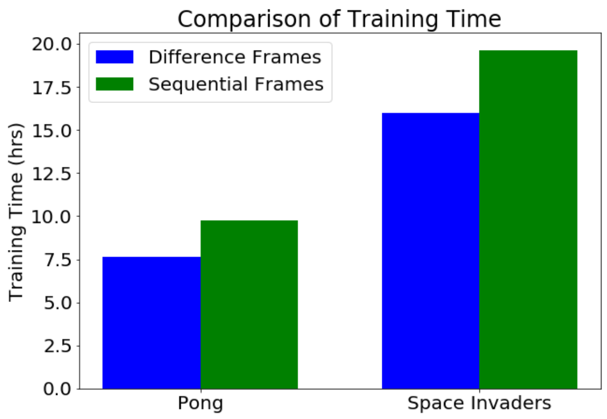

Figure 7: Comparison of training times, showing difference frames in blue as the left bar, and sequential frames are green as the right bar.

6. Discussion

In general our agent performed worse when trained using difference frames to represent the environment state. Clearly, there is information present when using stacked sequential game frames to represent environment state that is lost when converting to difference frames. However, we also observed that using difference frames worked reasonably well on Pong, while the results were much poorer on Space Invaders. We believe that the disparity in the two games is due to greater complexity and number of elements in Space Invaders. Figure 8 shows an example of a still frame from Pong and Space Invaders.

To understand what is lacking when using using difference frames for Pong vs. Space Invaders we need to consider how difference frames are created. In the case of Pong, when we create difference frames by taking the subtraction of two sequential frames, the background will be zeroed out (this also happens in Space Invaders), and there will be negative regions representing where the ball and paddles were previously, and positive regions where the ball and paddles are currently at. In Space Invaders the aliens are lined up in vertical columns, and after looking at the difference frames it appears that as the aliens track sideways across the screen there are regions were aliens will overlap and essentially cancel the image out at times. We believe that this problem with overlapping is the primary reason that using difference frames results in poor performance on Space Invaders.

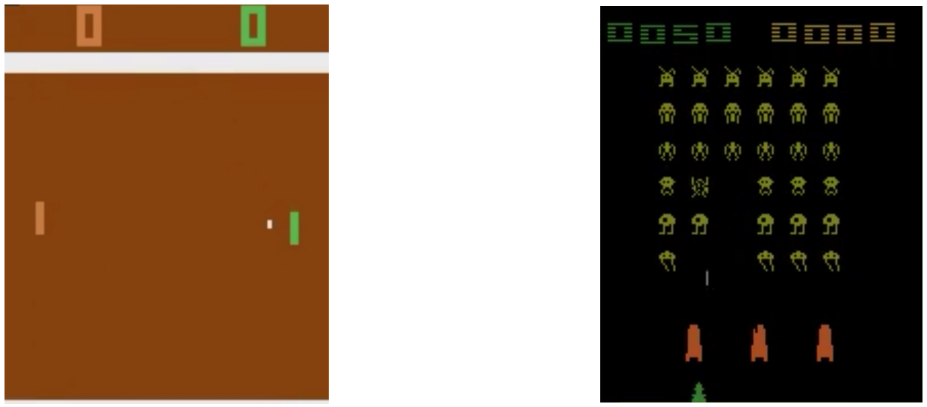

Figure 8: Examples of still frames from Pong (left) and Space Invaders (right).

We also hypothesized that a single difference frame could contain as much game relevant information as multiple sequential frames, and that if this was the case then when using difference frames, reductions in the experience replay buffer size would not be as detrimental compared to when using stacked sequential frames. Our results were mixed in this regard. When using difference frames we did observe much smaller reductions in max score on Space Invaders, but the result was reversed for Pong, with stacked sequential frames maintaining better performance. Perhaps the reason we observed small reductions in score as the experience replay buffer was reduced when using difference frames on Space Invaders was that the agent was already performing poorly, so it simply did not have as much room to fall, so to speak.

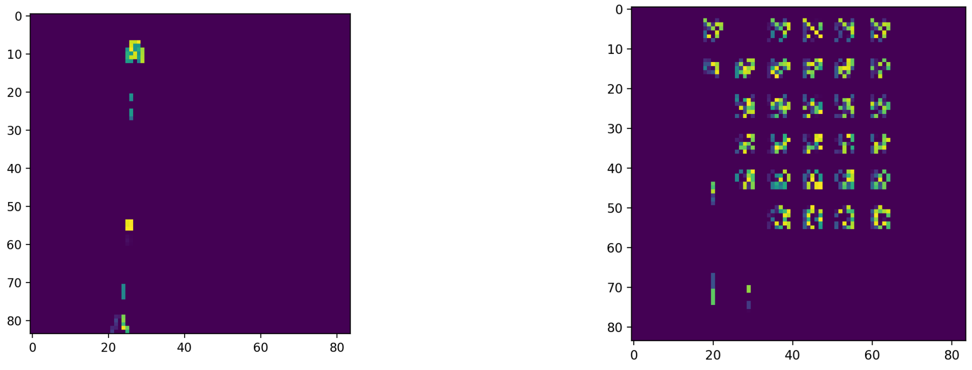

Figure 9: Examples of difference frames from Space Invaders. In the image on the left much of the objects in the scene have been canceled out, because there were only small changes in position. In the scene on the right many more objects are visible as position changes from a previous frame to the next must have been more significant.

The clearest benefit from using difference frames is the reduced runtime in terms of wall clock time. Since using difference frames reduces the number of network parameters by a factor of four (compared to using four stacked sequential frames), we expected network training to be quicker, but we weren't sure what the magnitude of the training time reduction would be. For our experiments the reduction was on the order of 2 to 3 hours for a single training run, which isn't insignificant when you are running multiple sets of experiments.

An interesting ancillary result not pertaining to our main hypotheses, was that performance did not drop as drastically as we expected when reducing the experience replay buffer size. We have not seen studies evaluating the significance of the size of the experience replay buffer. It seems that the common size of 1,000,000 frames is simply based off what was used in the original study by Mnih et al..

7. Conclusion

Based on our set of experiments it appears that using difference frames as an environment state representation can be a viable alternative to stacked sequential frames in situations where there are typically few objects in a frame. Otherwise objects tend to overlap in subsequent frames often producing muddled images when difference frames are created. In the type of situations where difference frames tend to perform well, the use of difference frames will typically result in reduced computation time on the order of hours, with the benefit growing as more iterations of the agent training algorithm (in our case Q-learning) are required.

The use of difference frames will need to be tested on a greater number of environments beyond the two we tested here, to get a better sense of general agent performance when using difference frames. Again, we expect that games with few objects, such as the Atari game "Breakout", will work well with difference frames.

An intriguing area for future research would be to integrate the use of difference frames and stacked sequential frames. As mentioned previously, we believe that the major problem with difference frames is that objects are sometimes removed from the scene. It may be possible to use a single difference frame stacked with a single unmodified frame as a state representation. The difference frame would provide a sense of motion and velocity that is required to a avoid a partially observed Markov decision process, and the unmodified frame would provide the current positions of objects, but the size of the state representation and the size of the neural network required to process the state input would still be reduced.

References

Shawn LE Beaulieu, Sam Kriegman, and Josh C Bongard. Combating catastrophic forgetting with developmental compression. *arXiv preprint arXiv:1804.04286*, 2018.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, King's College, Cambridge, 1989.