Deep Q-Learning with Recurrent Neural Networks

Clare Chen
cchen9@stanford.edu

Vincent Ying
vincenthying@stanford.edu

Dillon Laird
dalaird@cs.stanford.edu

Abstract

Deep reinforcement learning models have proven to be successful at learning control policies image inputs. They have, however, struggled with learning policies that require longer term information. Recurrent neural network architectures have be used in tasks dealing with longer term dependencies between data points. We investigate these architectures to overcome the difficulties arising from learning policies with long term dependencies.

1 Introduction

Recent advances in reinforcement Learning have led to human-level or greater performance on a wide variety of games (e.g. Atari 2600 Games). However, training these networks can take a long time, and the techniques presented in the state of the art [0] perform poorly on several games that require long term planning.

Deep Q-networks are limited in that they learn a mapping from a single previous state which constist of a small number of game screens. In practice, DQN is trained using an input consisting of the last four game screens. Thus, DQN performs poorly at games that require the agent to remember information more than four screens ago. This is evident from the types of games DQN performs poorly at, near or below human-level, in Figure 1.

We explore the concept of a deep recurrent Q-network (DRQN), a combination of a recurrent neural network (RNN) [6] and a deep Q-network (DQN) similar to [5]. The idea being that the RNN will be able to retain information from states further back in time and incorporate that into predicting better Q values and thus performing better on games that require long term planning.

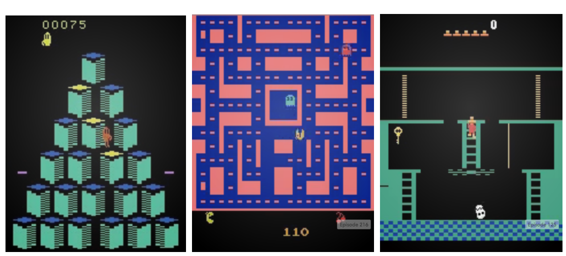

Figure 1: Q*bert, Ms. Pac-Man and Montezuma's Revenge

In addition to vanilla RNN architectures we also examine augmented RNN architectures such as attention RNNs. Recent achievments of attention RNNs in translation tasks [2, 3] have shown promise. The advantage of using attention is that it enables DRQN to focus on particular previous states it deems important for predicting the action in the current state. We investigate augmenting DRQN with attention and evaluate its usefulness.

2 Background

3 Deep Recurrent Q-Learning

We propose three architectures for the Deep Recurrent Q-Network (DRQN). The first is a very basic extension of DQN. The architecture of DRQN augments DQN's fully connected layer with a LSTM. We accomplish this by looking at the last L states, $\{s_{t-(L-1)}, \ldots, s_t\}$ and feed these into a convolutional neural network (CNN) to get intermediate outputs $\text{CNN}(s_{t-i}) = x_{t-i}$. These are then fed into a RNN (we use an LSTM for this but it can be any RNN), $\text{RNN}(x_{t-i}, h_{t-i-1}) = h_{t-i}$, and the final output h_t is used to predict the Q value which is now a function of $Q(\{s_{t-(L-1)}, \ldots, s_t\}, a_t)$.

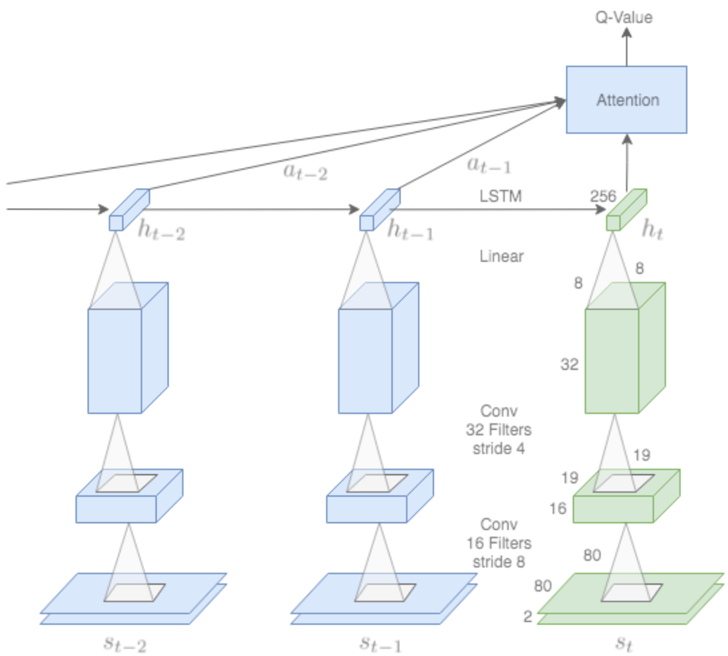

Figure 2: Architecture of the Attention DRQN

The second architecture we used was a version of an attention RNN, depicted in Figure 2, we are calling linear attention. For the linear attention RNN we take the L hidden states outputted by the RNN, $\{h_{t-(L-1)}, \ldots, h_t\}$ and calculate an inner product with v_a, $\{v_a^T h_{t-(L-1)}, \ldots, v_a^T h_t\}$. This allows the model to focus more on nearby hidden states or further away states depending on what values of v_a are learned. We then take a softmax over these values, $a_{t-i} = \text{softmax}(v_a^T h_{t-i})$. We use this softmax to take a weighted sum over the hidden states to get a context vector, $c_t = \sum_{i=0}^{L-1} a_{t-i} h_{t-i}$. This context vector is then used to predict the Q value.

The third architecture we used is global attention, similar to the global attention used in [3]. We treat the current state, s_t as the "decoder" input and the previous $L-2$ states as the "encoder" inputs. We compute the following scores, $\{{h_{t-(L-1)}}^T h_t, \ldots, {h_{t-1}}^T h_t\}$. We then take a softmax over these values, $a_{t-i} = \text{softmax}({h_{t-i}}^T h_t)$. The context vector is computed as a weighted sum over the previous hidden states, $c_t = \sum_{i=1}^{L-1} a_{t-i} h_{t-i}$. Finally the context vector is used to compute $\tilde{h} = \tanh(W_a[h_t; c_t] + b_a)$ which is then used to predict the Q value. This type of attention allows the model to focus on previous states depending on the current state h_t as opposed to a fixed vector such as v_a.

Learning sequences of observations creates some difficulty when sampling experiences and using the following loss function [0]:

$$L_i(\theta_i) = \mathbb{E}_{e_t \sim U(D)}\left[\left(r_t + \gamma \max_{a_t'} Q(s_{t+1}, a_t'; \theta_i^-) - Q(s_t, a_t; \theta_i)\right)^2\right]$$

where $e_t = (s_t, a_t, r_t, s_{t+1})$, $D_t = \{e_1, \ldots, e_t\}$, γ is the discount factor and $e_t \sim U(D)$ are drawn uniformly at random from the pool of stored experiences. We propose a simple solution where we sample $e_t \sim U(D)$ and then take the previous L states, $\{s_{t-(L+1)}, \ldots, s_t\}$ and zero out states from previous games. For example if s_{t-i} was the end of the previous game then we would have states $\{\mathbf{0}, \ldots, \mathbf{0}, s_{t-(i+1)}, \ldots, s_t\}$ and similarly for the next state $\{\mathbf{0}, \ldots, \mathbf{0}, s_{t-(i+2)}, \ldots, s_{t+1}\}$.

4 Experiments

For our experiments we chose to play the game Q*bert. We chose Q*bert because it was a challenging game for DQN which achieved scores only slightly above human-level [0] but it was not so challenging that DQN could not make any progress, such as Montezuma's Revenge [0].

For input the DRQN takes a specified number of screens per state. The screen images are grayscaled and resized to 80×80. The first hidden layer convolves 16, 19×19 filters with stride 8 across the input image and applies a rectified linear unit. The second hidden layer convolves 32, 8×8 filters with stride 4, again followed by a rectified linear unit. Convolutional outputs are fed to an LSTM with 256 hidden units per layer. Finally, a fully connected linear layer outputs a Q-value for each possible action.

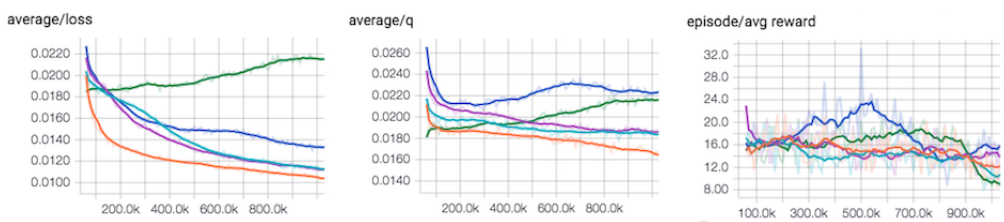

Figure 3: Graphs for Q*bert Over Iterations

In Figure 3, the orange line is a regular DQN architecture with an CNN and 4 frames per state. The light blue line is a DRQN with $L = 4$ and 1 frame per state. The pink line is an DRQN with $L = 4$ and 2 frames per state. The blue line is a DRQN with $L = 8$ and 1 frames per state and the green line is a DRQN with $L = 8$, 2 frames per state and attention.

We can see from the graphs that blue seems to be performing the best in terms of average reward per episode. It also has the highest q value. Green, which is just blue with attention, performs slightly worse than blue but has a very difficult time minimizing loss. This tells us that our basic linear attention does not do much to help the model. A better attention mechanism conditioned on the most recent state might work better. Curiously, green's average reward per episode takes a dip after 900 thousand iterations below all the other models. It is apparent that for smaller L, DRQN performs basically the same as a regular DQN but has a slightly more difficult time minimizing the loss.

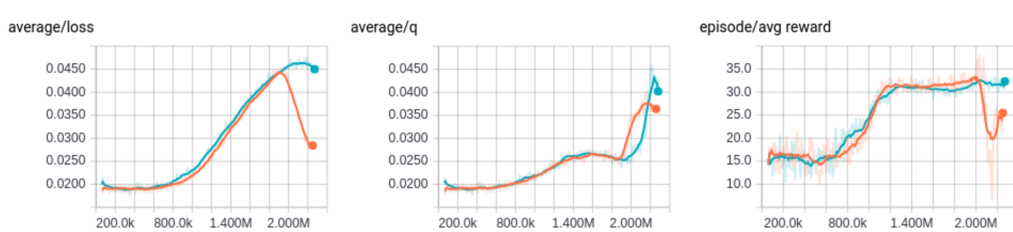

Figure 4: Graphs for Q*bert With 1-layer and 2-layer RNN and Attention

In Figure 4, an experiment has been run using the DRQN architecture with one and two layer RNN component. Both models are DRQN with $L = 8$, with 2 frames per state and attention. The orange line represents the model with 1-layer RNN and the light blue line represents the model with

2-layer RNN. As can be seen, there is not much difference between the models, with the 2-layer RNN model performing slightly better on all the measures except the loss. However, note that the 2-layer RNN has more difficulty minimizing loss than the 1-layer RNN after 1.8 million iterations.

Both DRQN models have similar performance to the earlier models utilizing DQN with CNN and the DRQN at the start. However, they were run twice as long and have a dramatic increase in performance after 1 million iterations. This indicates that future experiments will need a lengthier training period before they can be properly evaluated or utilized. Unfortunately, we did not have enough time to experiment with a large number of parameters and run the models longer. This will be done in the second phase of the project.

References

[0] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529-522, 2015.

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Minh-Thang Luong, Hieu Pham and Chisopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv preprint arXiv:1508.04025*, 2015.

[4] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov and Anastasiia Ignateva. Deep Attention Recurrent Q-Network. *arXiv preprint arXiv:1512.01693*, 2015.

[5] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *arXiv preprint arXiv:1507.06527*, 2015.

[6] Sepp Hochreiter and Jurgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735-1780, 1997.