Deep Q-Learning With Recurrent Neural Networks

Clare Chen, Vincent Ying, Dillon Laird

Stanford, 2016

Abstract

Deep reinforcement learning models have proven to be successful at learning control policies image inputs. However, they have struggled with learning policies that require longer term information. Recurrent neural network architectures have been 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.

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).
- Deep Q-networks are limited in that they learn a mapping from a single previous state which consist of a small number of game screens.
- We explore the concepts of deep recurrent Q-networks (DRQN), and also a combination of recurrent neural network (RNN) and deep Q-network (DQN).
- In addition to standard RNN architectures, we also examine augmenting the RNN architecture with an attention mechanism.

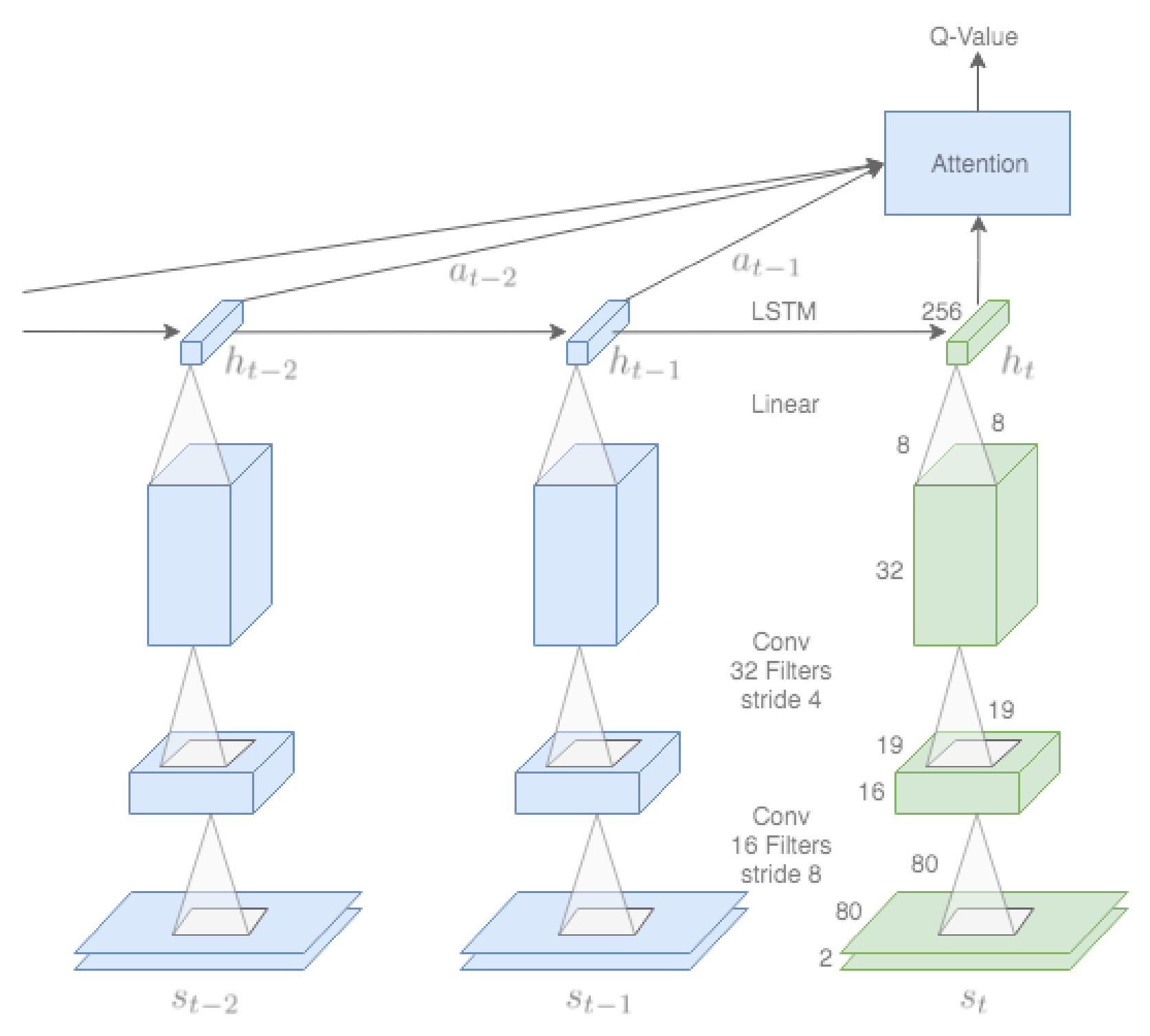
Q*bert, Seaquest, and Ms. Pac-Man

Deep Recurrent Q-Learning

- The architecture of DRQN augments DQN's fully connected layer with a LSTM.
- We accomplish this by looking at the last L states:
$$\{s_{t-(L-1)}, \dots, s_t\}$$
- We feed these into a convolution neural network (CNN) to get intermediate outputs and then send those through the RNN:
$$\text{CNN}(s_{t-i}) = x_{t-i}$$
$$\text{RNN}(x_{t-i}, h_{t-i-1}) = h_{t-i}$$
- The final output is used to predict the Q value.

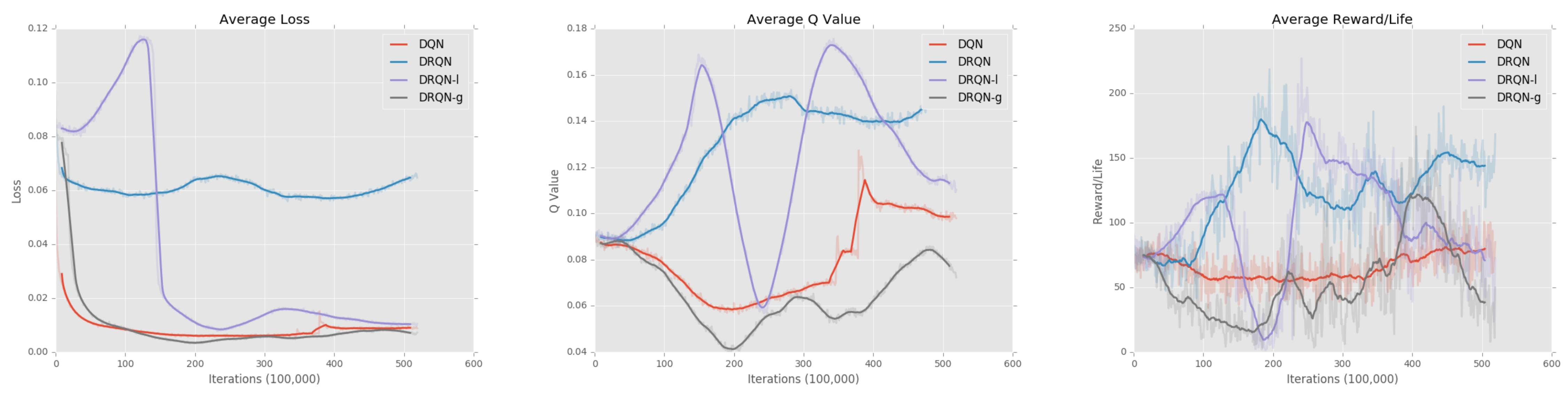
Attention Deep Recurrent Q-Learning

- In **linear attention**, we take the last L hidden states and compute the inner product with learned vector v_a and take a softmax:
$$\{v_a^T h_{t-(L-1)}, \dots, v_a^T h_t\},\ a_{t-i} = \text{softmax}(v_a^T h_{t-i})$$
- Then, we take a weighted sum of hidden states to get a context vector c_t that is used to calculate Q:
$$c_t = \sum_{i=0}^{L-1} a_{t-i} h_{t-i}$$
- In **global attention**, we use the current hidden state, h_t, to calculate the attention scores:
$\{{h_{t-(L-1)}}^T h_t, \dots, {h_{t-1}}^T h_t\}$, $a_{t-i} = \text{softmax}({h_{t-i}}^T h_t)$
- We then compute a context vector similar to above and pass it through a final layer before it is used to compute Q:
$$c_t = \sum_{i=1}^{L-1} a_{t-i} h_{t-i},\ \tilde{h} = \tanh(W_a[h_t; c_t] + b_a)$$



Architecture of the Attention DRQN

Experiments



Graphs for Q*bert Over Iterations

- Images are grayscaled and resized to 80×80
- CNN is 2 layers, first is 16, 19×19 filters with stride 8, second is 32, 8×8 filters with stride of 4.
- DRQN uses $L = 16$ and 2 screens per state.
- DRQN takes up to 6 days to complete 5 million iterations.

Results

*Q*bert Scores*

Algorithms	Scores
Random	150
DQN	700
DRQN	**850**
DRQN-l	700
DRQN-g	550

- DRQN boosts score by 20% while global attention hinders score by 20%.

Discussion

Conclusion

- DRQN clearly adds benefits for training on more difficult games although it takes much longer to train.
- Attention makes it much more difficult to train by adding extra complexity to the architecture and more parameters.
- Attention is not necessary for playing games such as Q*bert where the full game state is visible on the screen. It may be more benificial for games where game states are only partially observed.

Future Work

- Testing DRQN and DRQN with attention on games such as Battlezone where game state is partially observed.
- Trying different augmented recurrent neural networks such as memory networks for handling even longer term dependencies in games.
- Training the algorithms for longer. It is clear the algorithms did not converge and if we had more time we would run them for longer.