Validating ADRQN for POMDPs

A-
good

XueGuang Lu
Northeastern University
360 Huntington Ave.
Boston, Massachusetts 02115
xueguang@ccs.neu.edu

Abstract

Recent works on deep reinforcement learning (DRL) have devised many techniques which achieved human-level performances on sequential decision making tasks given fully-observable high dimensional environment [Mnih et al., 2015] such as the game of Go. Following the work of extending DRL to partially observable environments, we present additional experiment on Action-specific Deep Recurrent Q-Network (ADRQN), proposed by Zhu et al. [Zhu et al., 2017], on the Atari 2600 game "Space Invader" in addition to the experiments in the original article on "Pong", "Chopper Command" and "Asteroids" which all out-performed predecessor model DRQN. In this paper, we validate these results and show the effectiveness of the model through experimentation on similar tasks, while also demonstrating its limitations in terms of suitable tasks.

Introduction

Research in deep reinforcement learning has combined the power of deep learning and reinforcement learning to create end-to-end, general and high-performance learning agents. Thanks to the Arcade Learning Environment [Bellemare, Marc G., et al. 2013] which provides unified input and output dimensions for drastically different environments (Atari 2600 games), enables researchers to test and validate the generality of agents, Mnih et al. proposed DQN (Deep Q-Network) [Mnih et al., 2015] which achieved human-level control on 49 games, whose results were soon to be improved by several proceeding articles, such as Double Q-Learning [Van Hasselt et al., 2016], Dueling network architectures [Wang et al., 2015] and Asynchronous actor-learner models [Mnih et al., 2016]. The Arcade Learning Environment provides $210 \times 160 \times 3$ pixel values as output, and 6 simple controller inputs. While action space is small and simple, the state is large and, in the original DQN architecture, each state is combined using a sliding window of 4 stacked frames to preserve time series information, which makes the state space 414720 dimensions. The core idea of DQN is to use convolutional neural network to approximate the Q function in a MDP problem formulation for a given state and action by first extracting useful visual features from high dimensional input. Subsequent work on Deep Recurrent Q-Network (DRQN) [Hausknecht and Stone, 2015] relaxes fully-observable state assumption, hence making it a POMDP formulation, by introducing a LSTM Layer into the model after the convolutional layers. When making partial-observable assumption, while real states remains Markovian, we usually maintain a underlying probability distributing over actual states. In this case, maintaining actual states is, like aforementioned, is computationally unlikely; therefore, adding recurrent layer into the model simulates an arbitrary number of sequence of state history, which, in best case, consists of most information needed to infer the actual state distributions using theoretical Bayesian methods.

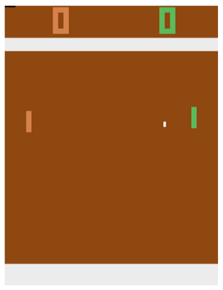
Figure 1: Game of Pang

Figure 2: Game of Space Invaders

The DRQN out-performed DQN on various Atari games, by a large margin under a test condition called flickering, where artificial noise is added to the input pixels to create imperfect information. Upon attempt to improve the model further, Zhu et al. points out that state history alone is theoretically insufficient information to infer the underlying state distribution. Zhu et al., while acknowledging similar work on DDRQN (deep distributed recurrent Q-networks) which includes action history series as inputs [Foerster et al., 2016], proposed a model which takes state action pairs as called Action-based Deep Recurrent Q-Network (ADRQN).

This paper documents further experiments and validations on ADRQN and highlights differences in performances on different game environments, from "Pong" to "Space Invaders". Some of the experiment results are brought from the Zhu et al.'s paper without implementation.

Background

In this section, we provide a brief review of Markov Decision Process (MDP), Deep Q-Network (DQN), Partially Observable Markov Decision Process (POMDP), Deep Recurrent Q-Network (DRQN) and Action-specific Deep Recurrent Q-Network (ADRQN).

Markov Decision Process and Q-Learning

We can formulate the sequential decision making problem as a Markov Decision Process $< S, A, T, R >$ where S is a finite state space, A is a finite action space, $T(s, a, s')$ is the probability of transitioning from state $s \in S$ to $s' \in S$ by taking action $a \in A$, and $R(s, a, s')$ is the immediate reward for such transition. The Markovian assumption is implied by the transition probability. Under thus formulation, the problem is to find a optimal policy $\pi^\star$ which maximize sum of rewards.

One popular model-free method of learning the optimal policy is Q-Learning [Watkins and Dayan, 1992], which iterate on a set of Q values or approximators to approach optimal Q values or estimates, where $Q(s, a)$ is the expected maximum sum of rewards achievable in the future given in state s and taking action a.

$$Q^\star(s,a) = \max_\pi E(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a, \pi) \tag{1}$$

Traditional Q-Learning methods often guarantees to convergence on discrete set of states and actions. The update of tabular approach is shown below.

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)) \tag{2}$$

Where $\alpha \in (0, 1]$ is learning rate at which the Q value updates, and $\gamma \in (0, 1]$ is the discount factor applied to future reward estimations.

Deep Q-Network

Mnih et el.'s proposed DQN model develops on a common practice where a function approximator is used for estimating Q values by parameterize Q function $Q(s, a, \theta)$ using parameter θ using deep convolutional neural network. In particular, θ is the weights of the network. DQN uses experience replay [Lin, 1993] where each transition (s_t, a_t, r_t, s_{t+1}) is stored into a large fix-sized experience buffer $D_t = \{(s_1, a_1, r_1, s_2), ..., (s_t, a_t, r_t, s_{t+1})\}$ from which all training batches for the network is uniformly sampled. The update of the network follows the following loss function:

$$L_i(\theta_i) = E_{s,a,r,s' \sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{3}$$

where θ_i^- is the parameters for target network at iteration i, the target network is an identical network whose parameters are never directly updated but copied from the main network every C steps in order to maintain value stability.

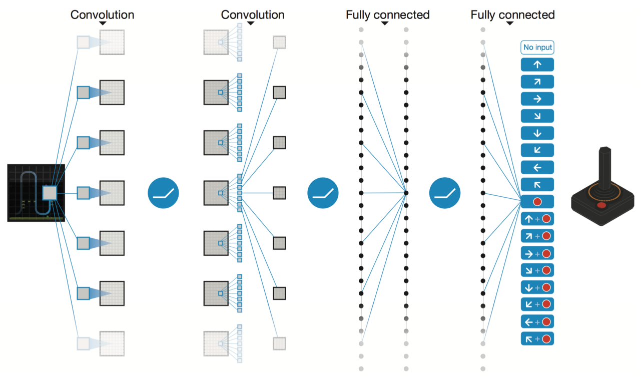

Figure 3: DQN Architecture

Partially Observable Markov Decision Process

DQN makes fully observable assumption and proved effectiveness in games like Go or most Atari games, but suffers inaccuracy in tasks with imperfect information for some Atari games with hidden states and often real-world tasks. POMDP extends MDP by defining states to be not directly observable, formally defined as $< S, A, Z, T, O, R >$; where S a finite state space, A is a finite action space, Z is a finite set of observations, $T(s, a, s')$ is the probability of transitioning from state $s \in S$ to $s' \in S$ by taking action $a \in A$, $O(s, a, z)$ is the probability of observing $z \in Z$ given state s and action a, $R(s, a)$ is the immediate reward for such transition.

Although we do not have the true state, we can maintain a belief distribution over our state space, denoted as $b = (b(s_1), b(s_2), ..., b(s_{|S|}))$ where $s_i \in S$, $b(s_i) \geq 0$ and $\sum_{s_i \in S} b(s_i) = 1$. The believe update when given current belief b^t, action a and observation z, $b^{t+1} = SE(b^t, a, z)$ can be obtained from the following:

$$b^{t+1}(s_j) = \frac{O(s_j, a, z) \sum_{s_i \in S} T(s_i, a, s_j) b^t(s_i)}{P(z|a, b^t)} \tag{4}$$

$$P(z|a, b^t) = \sum_{s_j \in S} O(s_j, a, z) \sum_{s_i \in S} T(s_i, a, s_j) b^t(s_i) \tag{5}$$

good

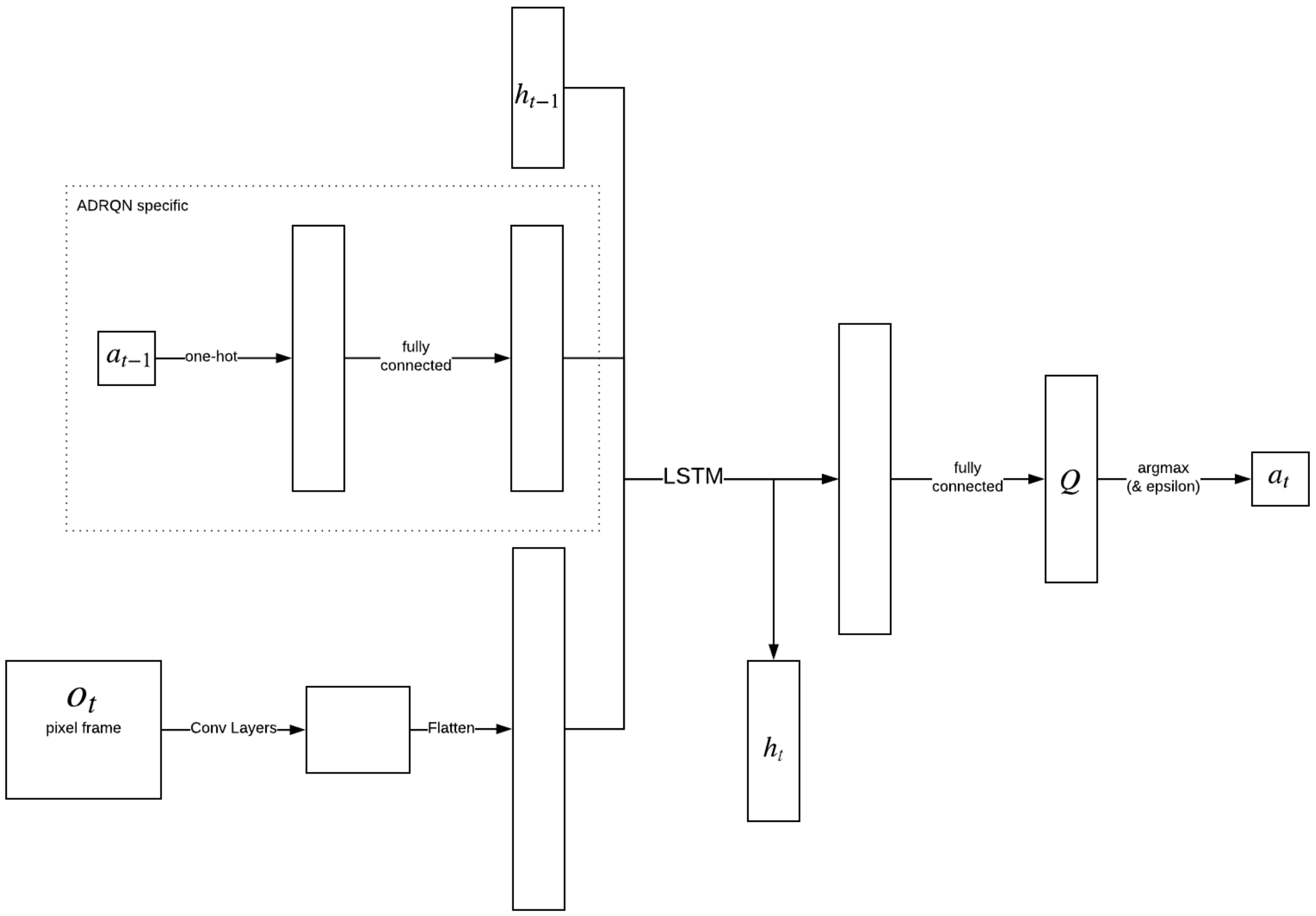

Figure 4: Architecture of DRQN and ADRQN

The expected immediate reward in state s, taking action a, can be obtained as $p(b,a) = \sum_{s_i \in S} b(s_i)R(s_i,a)$. The transition function among beliefs becomes $\tau(b,a,b') = \sum_{z \in Z} p(b'|b,a,z)P(z|b,a)$ where $p(b'|b,a,z) = 1$ if $b' = SE(b,a,z)$, and 0 otherwise. The optimal policy $\pi^\star$ can be computed by value iteration:

$$V(b) = \max_a [p(b,a) + \gamma \sum_{b'} \tau(b,a,b')V(b')] \qquad (6)$$

where γ is the discount factor applied to past rewards.

Deep Recurrent Q-Network

As we already mentioned that DQN assumes each observation represents the full state of the environment (MDP); in the case of Atari games, 4 frames are stacked together to form one observation in order to preserve time series information, such as the velocity of objects. This work-around is, as Mnih et al. points out, is owing to the face that these tasks are partially observed. As one would expect, DQN performs insignificantly on complex Atari games. [Hausknecht and Stone, 2015] also showed most Atari games is in fact partially observable despite their simplicity.

To extend the model to accommodate truly partially observable tasks, [Hausknecht and Stone, 2015] proposed a model called Deep Recurrent Q-Network (DRQN), where a recurrent (LSTM) layer was used to replace the first post-convolutional fully-connected layer of DQN. Hausknecht and Stone argues that the recurrent layer is able to integrate an arbitrarily long history which can be used to infer the underlying state, and the frame-stacking method is hence being replaced by integrating hidden states. The model out-performed DQN on partially observable games and is on par with DQN on fully-observable games. Formally, the network estimates $Q(o_t, a_t, h_{t-1}|\theta)$ (instead of $Q(s_t, a_t, \theta)$), where θ is the parameters of the network, and $h_{t-1} = LSTM(h_{t-2}, o_t)$ is the hidden state generated from the LSTM layer from $t-1$ time step. Although the model arguably appears to conform more closely neuroscientifically , [Hausknecht and Stone, 2015] nonetheless points out that DRQN confers no systematic benefit compared to DQN where frames are stack together on the Atari tasks.

Action-specific Deep Recurrent Q-Network

Built on top of DRQN, shown in Figure 4, one major differnece is that ADRQN uses the action from previous time

More Details

step a_{t-1} as input to the LSTM layer. Notice that the action, when inputting, goes through a one-hot encoder and a fully connected layer. Argued by [Zhu et al., 2017], the state, action pairs are crucial for believe updates as shown in Eq. 5. The new architecture is able to remember past actions, thus the performance of the model is improved in all three of the games shown in the original paper.

Theoretically, the optimal method of estimating current state without explicitly estimating transition and emission probability is to use an array of all previous observations as input. However, this approach is not practical for Atari games since they usually have hundreds if not thousands of transitions and LSTM layer need to be unrolled for too many transitions. Therefore, to maintain computational tractability, the LSTM layer is unrolled for fixed small number of transitions (10 transitions in the original paper).

Related Work

There are several techniques that made training feasible and stable. In this section we will talk about four of them: Doubling, Dueling, Prioritized Experience Reply, and Flickering.

Double network

Double DQN is a model proposed by [Van Hasselt et al., 2016] to address the issue where value estimations in DQN is usually higher than optimal values by a random margin, an issue Van Hasselt et al. brought to attention in the same paper. This ”overoptimism”, as Van Hasselt et al. called it, is successfully reduce by Doubling: where instead of taking the max over Q-values when computing the target Q value for our training step, we use our primary network to chose an action, and our target network to generate the target Q value for that action.

Dueling network

Dueling DQN is a model proposed by [Wang et al., 2015] following the DQN by Mnih et al., where the last layer is decoupled into two vectors of what is called action advantages $A(s, a)$ and state value $V(s)$, thus the Q values can be re-obtained as $Q(s, a) = V(s) + A(s, a)$. The Dueling method increased robustness of the model and dramatically improved performance on most Atari games.

Prioritized Experience Reply

Instead of uniformly sample from experience buffer, Prioritized Experience Reply [Schaul et al., 2016] increases the replay probability of transitions which incurred a high absolute TD-error. This technique offers faster convergence and better performance across most Atari games.

Flickering

Flickering is a technique from which one is able to create POMDP tasks using MDP tasks presented. The idea of Flickering is to artificially introduce noise into the pixel frames, more specifically, [Hausknecht and Stone, 2015] in their Flickering Pong environment, they either reveal or obscure the frame entirely with a probability of 0.5. This setup allows researchers to test their models on POMDP environments, and to compare with their non-flickering counterparts. The same technique is used by [Zhu et al., 2017] on their work of ADRQN.

Model Description

As partially highlighted in Figure 4 that our model consists of three convolutional layers, one LSTM layer, two fully connected layers (one for up-scaling one-hot action vector, another for processing LSTM output).

Network Units and Parameters

The one-hot encoded action vector is 18-D, which becomes 512-D after up-scaling through the fully connected layer. Each observation o_i is re-scaled to $84 \times 84 \times 3$ instead of $216 \times 160 \times 3$ as to conform with all previous works. The first convolutional layer has 32 8×8 filters with stride $= 4$, 64 4×4 filters with stride $= 2$, and 64 3×3 filters with stride $= 1$. The tensor is then flattened before feeding into LSTM layer. Finally the output of the LSTM layer is sliced into Advantage vector and Value vector, both 512-D and have one more of their own fully connected layer to produce Advantage vector (18-D) and Value scalar respectively.

Experience Buffer and modifications

Since for each time step, also batch training, we need to look up the previous action, we define each entry in the experience buffer to be $(\{a_{t-1}, o_t\}, a_t, r_t, o_{t+1})$ (instead of (s_t, a_t, r_t, s_{t+1}) in the case of DQN and DRQN) in order to conveniently retrieve the action a_{t-1}. The Experience Buffer length used is a double-ended queue, whose maximum length is maintained to be smaller than or equal to 1 million transitions. Notice that the transitions are not stored linearly in the buffer; instead, the buffer consists of episodes (array of transitions), and when we sample, we sample the episodes (instead of transitions) and slice from each sampled episodes to obtain arrays of transitions for batch training, this technique, called random updates, will be elaborated in the next section.

Experiments

We set out to test and validate the performance for ADRQN using Space Invader as our task, since it is a popular game and was not used in evaluation in the original work on ADRQN.

Experiment setup

Frame skip Scheme We utilize the frame skip technique [Bellemare et al., 2013], a technique that is commonly used in all previous works of Deep reinforcement learning on Atari tasks. The idea is that the agent select a task a_i for $k + 1$ consecutive frames and define the effect of the action to be the first frame f_0 to frame f_{K+1}, effectively skipping frames for increased time efficiency while suffering minimal performance loss. It is common that k is set to 4.

should also try without this

Flickering and non-Flickering Space Invaders As to achieve comparability with original work on ADRQN [Zhu et al., 2017] and DRQN [Hausknecht and Stone, 2015], we adopt the same Flickering technique with probability 0.5 where entire frame of pixels where set to zeros. This presents a strong POMDP task.

Simulator Environment Atari games are simulated through Arcade Learning Environment (ALE) [Bellemare et at., 2013], a framework which provides raw pixels output and raw controller input for supported Atari 2600 games.

Random Updates Each sampled episode is further sliced into *traces*, a short array of transitions of length d. [Hausknecht and Stone, 2015] points out that Random Update allows for faster training while maintain same performance compared to preform updates using the entire episode conventionally. Notice that each transition includes additional action a_{t-1}; and a_0, although does not exist, is treated as 0. d is set to 10 for the ADRQN experiments and to 8 for the DRQN experiments.

ϵ-greedy During training, actions performed are selected by ϵ-greedy with ϵ annealed linearly from 1.0 initially to 0.1 at 1 million time step and stay at 0.1 ever since. The evaluation experiments have ϵ set to 0.05 for consistency with previous works.

Training

We evaluate DQN, DRQN and ADRQN on non-flickering *Space Invaders*, additionally, DRQN and ADRQN on flickering *Space Invaders* with observation probability $op = 0.5$.

For non-flickering setting, Figure 5 shows how evaluation score (expected sum of rewards during evaluation episodes) rises with increasing time step. From Table 1, we observe that, unlike previously evaluated games, in non-flickering setting, the performance of ADRQN is not convincingly higher than DRQN on *Space Invaders*; although that of *Pong*, *Chopper Command* and *Asteroids* does not suggest ADRQN has a strong advantage over ADRQN anyways. One factor we should also consider is that *Space Invader* has a more limiting pool of reward, and it is not unusual for the agent to collect all (or most) of the rewards possible in a given episode (defeating all or most invaders).

Task	ADRQN($\pm std$)	DRQN($\pm std$)
Pong	15.7($\pm$2.5)	13.3($\pm$2.2)
Chopper Command	1190($\pm$662.4)	1150($\pm$397.9)
Asteroids	1014($\pm$374.6)	960($\pm$300.5)
Space Invaders	1049($\pm$117.76)	1040($\pm$110.5)

Table 1: Testing results of *Pong*, *Chopper Command*, *Asteroids* are brought from [Zhu et al., 2017] instead of recreated; *Space Invader* result is what we add to the table.

For flickering setting, Figure 5 shows that a significant drop of performance compared to perfect information setting. It is trivial to infer the reason: half of the information is lost for flickering setting. On the other hand, we notice from testing results from Table 2 that ADRQN achieved relatively higher performance than DRQN. The difference in performance suggests that action information provides more accurate state belief estimation than observation alone (see Figure 4 for key differences between the two models).

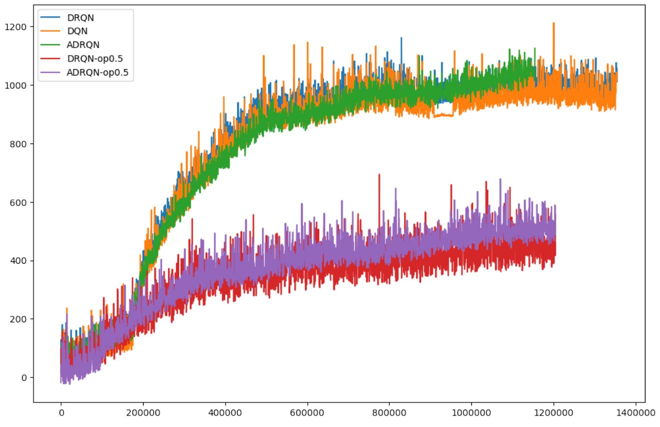

Figure 5: Training results for *Space Invaders*

Task	ADRQN($\pm std$)	DRQN($\pm std$)
Pong	4.6($\pm$6.1)	-8.3($\pm$6.8)
Chopper Command	1060($\pm$189.7)	850($\pm$259.3)
Asteroids	1034($\pm$300.9)	986($\pm$372.3)
Space Invaders	**514.6**($\pm$180.0)	428.3($\pm$155.1)

Table 2: Testing results with flickering setting where observation probability is 0.5 and ϵ set to 0.05. Same with Table 1, only *Space Invader* result is what we add to the table.

Conclusion

In this paper, we validate the performance increase of Action-specific Deep Recurrent Q-Network (ADRQN) on partially observable task compared to that of Deep Recurrent Q-Network (DRQN). The convolutional observations as well as encoded actions both contribute to latent variable estimator LSTM. We showed through experimentation that under partially observable setting, ADRQN is the more effective approach in solving flickering Atari games and possibly other high dimensional POMDP tasks.

References

[Bellemare et al., 2013] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. J. *Artif. Intell. Res. (JAIR)*, 47:253279, 2013.

[Bellemare et at., 2013] Bellemare, Marc G., et al. ”The Arcade Learning Environment: An evaluation platform for general agents.” J. *Artif. Intell. Res.(JAIR) 47* (2013): 253-279.

[Zhu et al., 2017] Zhu, Pengfei, Xin Li, and Pascal Poupart. ”On Improving Deep Reinforcement Learning for

More experiments and analysis

POMDPs.” *arXiv preprint arXiv:1704.07978* 2017.

[Mnih et al., 2015] Mnih, Volodymyr, et al. ”Human-level control through deep reinforcement learning.” *Nature 518.7540* 2015: 529.

[Mnih et al., 2016] Mnih, Volodymyr, et al. ”Asynchronous methods for deep reinforcement learning.” *International Conference on Machine Learning.* 2016.

[Van Hasselt et al., 2016] Van Hasselt, Hado, Arthur Guez, and David Silver. ”Deep Reinforcement Learning with Double Q-Learning.” *AAAI.* Vol. 16. 2016.

[Wang et al., 2015] Wang, Ziyu, et al. ”Dueling network architectures for deep reinforcement learning.” arXiv preprint arXiv:1511.06581 (2015).

[Hausknecht and Stone, 2015] Hausknecht, Matthew, and Peter Stone. ”Deep recurrent q-learning for partially observable mdps.” CoRR, abs/1507.06527 (2015).

[Foerster et al., 2016] Foerster, Jakob N., et al. ”Learning to communicate to solve riddles with deep distributed recurrent q-networks.” *arXiv preprint arXiv:1602.02672* (2016).

[Watkins and Dayan, 1992] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Machine Learning*, 8:279292, 1992.

[Lin, 1993] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[Wang et al., 2015] Wang, Ziyu, et al. ”Dueling network architectures for deep reinforcement learning.” arXiv preprint arXiv:1511.06581 (2015).

[Schaul et al., 2016] Schaul, Tom, et al. ”Prioritized experience replay.” *arXiv preprint arXiv:1511.05952* (2015).