Advancements of Deep Q-Networks

Bastian Birkeneder
Department of Computer Science and Mathematics
University of Passau
Passau, Germany
birkeneder@fim.uni-passau.de

Abstract—**Deep Q-Networks first introduced a combination of Reinforcement Learning and Deep Neural Networks at a large scale. These Networks are capable of learning their interactions within an environment in a self-sufficient manor for a wide range of applications. Over the following years, several extensions and improvements have been developed for Deep Q-Networks. In the following paper, we present the most notable developments for Deep Q-Networks, since the initial proposed algorithm in 2013.**

I. Introduction

Reinforcement Learning (RL) and especially Deep Reinforcement Learning (DRL) have experienced steadily increasing media attention over the last recent years. New methods and continuous improvements of existing models helped to develop powerful algorithms which are capable of learning highly complex tasks autonomously.
DRL has several practical uses in different domains like object localization [1], visual navigation [2], resource management [3], and traffic signal timing [4]. The ability to even outperform humans in complex tasks could explain the growing interest in this field of research. DeepMinds AlphaGo surprised the world in 2016 after beating one of the best players in one of the most computationally complex games, Go [5]. Mid 2018, OpenAI Five defeated a professional team in the popular multiplayer game Dota 2 [6]. In both cases the algorithms were trained in an RL setting by playing games against themselves.
It is no surprise that many existing RL studies and benchmarks are closely tied to video games. They offer a closed environment and the encountered settings often resemble problems also faced in real-world applications.

Deep Q-Networks (DQN) [7], [8] can be considered to be one of the most important milestones in DRL, outperforming even human players in complex Atari 2600 games. The success of DQN and their utilization of Neural Networks (NN) inspired similar DRL methods like Deep DPG [9] or A3C [10]. Additionally, various extensions for DQN were introduced like Double DQN [11] or Dueling Networks [12] to further increase the performance.
A large part of the recent success in DRL settings can be credited to NN. Advancements such as improved GPUs and increased availability of large data sets largely explain the rise of deep learning over the last decade, although often using only slightly modified algorithms originally dating back to over two decades. Long short-time memory units [13], for example, used in Recurrent Neural Networks (RNN), produced recent state-of-the-art results in tasks like speech recognition [14] or natural language processing [15].

II. Background

A. Reinforcement Learning

Reinforcement Learning (RL) is considered to be one of three categorical tasks of machine learning, besides supervised learning and unsupervised learning. The conceptual framework involves an agent, which interacts with its environment and incrementally learns new behavior. The agent starts in an initial state S_t and takes an action A_t for each time step t depending on observations extracted from the environment. In the subsequent time step $t+1$ a reward R_{t+1} is received based on the previous action taken and the state of the agent changes, resulting in new observations S_{t+1} [16]. A conceptual model of RL is depicted in Figure 1.

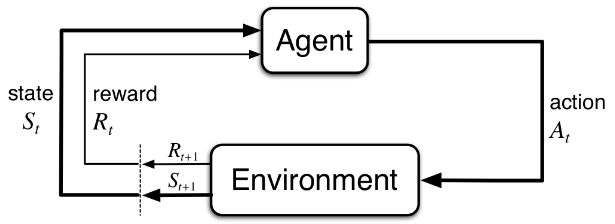

Fig. 1. Interaction between an agent and its environment.

The experience an RL agent acquires can either be learned in an offline setting, where all knowledge about the environment is collected before the actual learning process or in an online setting, where the gathered experience is gradually learned and influences the decision-making of the agent. The resulting predicament of resorting to learned behavior or exploring the environment is also known as exploration-exploitation trade-off/dilemma [17]. The RL agent has to decide whether the environment should be explored to gain new knowledge (exploration) or the existing experience is used to maximize the returned reward (exploitation), based on the extracted observations. The difficulty arises from balancing these two mutually exclusive tasks, since both are important to achieve the goal.

B. *Q-learning*

Q-learning can be considered as a type of model-free reinforcement learning [18]. The goal of the Q-learning algorithm is to learn the value of each state-action combination within a finite Markov Decision Process (FMDP). These values are called Q-values and estimate the received reward by taking a specific action. Intuitively, higher Q-values imply increased chances of getting higher rewards. The Q-value of a state-action pair for a certain time step (S_t, A_t) can be calculated as follows:

$$Q^{new}(S_t, A_t) \leftarrow (1-\alpha)\cdot Q(S_t, A_t) + \alpha \cdot \left(R_t + \gamma \cdot \max_A Q(S_{t+1}, A)\right) \quad (1)$$

where α is the learning rate $(0 < \alpha \leq 1)$ and γ the discount factor for future rewards $(0 \leq \gamma \leq 1)$.

C. *Deep Learning*

One of the advantages of NN over traditional machine learning algorithms is the ability to classify data points which are not linearly separable . This can be achieved by applying a nonlinear activation function to the linear combination of inputs in a neuron and by using at least one hidden layer between the input and output [19]. A stack of these layers, which perform nonlinear input-output mappings would be commonly referred to as deep-learning architecture. [20].
Especially two classes of NN, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), emerged as particularly successful in complex tasks such as image and speech recognition.

1) Convolutional Neural Networks: CNN were first recognized by a broader audience, after achieving impressive results on image recognition while using only limited image preprocessing. A convolutional layer consists of multiple trainable filters or kernels, which stride over the input data like an image or word embeddings. The filters convolve along the input and extract low level features like edges or patterns in case of an image. A pooling layer then allows down sampling of the resulting feature map. A stack of multiple convolutional and pooling layers enables the NN to detect higher level features like parts of objects or shapes. Figure 2 shows the ImageNet architecture as proposed Krizhevsky et al. [21] for image classification. After a series of convolutions and max pooling operations, the output is flattened and fed forward to a fully-connected layer.

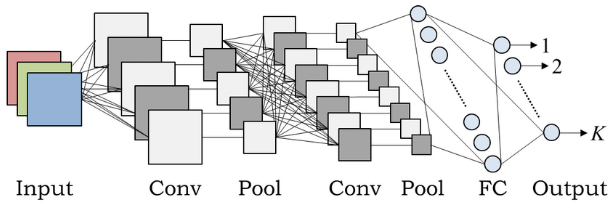

Fig. 2. Architecture of a Convolutional Neural Network.

2) Recurrent Neural Networks: RNN are another class of NN, which are especially suited for sequentially dependent data. An example would be the task to determine whether the sentiment of a product review can be considered as positive or negative. In this setting the semantic information of a word is dependent on all other words in its sentence (”The movie was good” - ”The movie was not good”) [22]. In a recurrent layer the output is dependent on the current input as well as the previous inputs of a sequence. Figure 3 depicts the concept of such an RNN.

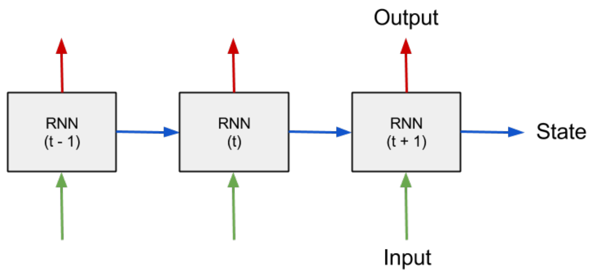

Fig. 3. Architecture of a Recurrent Neural Network.

Long short-term memory (LSTM) units were first described by Hochreiter and Schmidhuber to overcome the vanishing and exploding gradient problem of RNN and enable the ability to encode information from long sequences [13].

III. Deep Q-Networks

One of the main challenges of Q-learning is the convergence of the algorithm if the amount of possible environment states is too large. Although the algorithm converges for a finite set of states, given that every state action pair is updated often enough, the computation of every state-action value would be infeasible for many real word applications.
In the case of autonomously driving cars, for example, the

algorithm needs to work with different sensors and images to determine the current state of a car in its environment, inflating the computational costs for all possible action-values beyond reasonable. Therefore, it is often common to approximate the action-value function instead of calculating its real value to reduce complexity. NN qualify for this task as nonlinear function approximators.

Video games are often used as RL benchmarks, since they can be considered as representative of settings that might be faced in practice [23]. Mnih et al. [7] introduced Deep Q-Networks (DQN) to overcome the problem of high-dimensional observation spaces in Atari 2600 games.

The proposed algorithm utilizes a CNN Q as value approximator and to extract features of RGB images from the Atari environment. A separate target network is used during training to generate the target Y. At every fixed time step C, the weights and biases θ of Q are cloned to obtain the parameters θ^- of the target network. This technique improves the stability of the algorithm and it reduces the risk of divergence [8].

Another problem arises in the setting of online learning, since the gathered samples would be drawn subsequently from the environment. They would be close to each other, which means the training batch of the NN would not be i.i.d. To compensate this, the algorithm uses a memory buffer (or replay memory) to transform an online learning setting into an offline learning setting. The replay memory D stores the last N observations as tuple of states, actions, rewards, and next states (S_t, A_t, R_t, S_{t+1}) from the environment, which are uniformly random sampled $(S, A, R, S') \sim U(D)$ and used as minibatch update for the NN [24, Chapter 6]. The loss function for episode i of the algorithm is calculated as follows:

$$L_i(\theta_i) = \mathbf{E}_{(S,A,R,S')}\left[\left(R + \gamma \cdot \max_{A'} Q(S', A'; \theta_i^-) - Q(S, A; \theta_i)\right)^2\right] \quad (2)$$

Furthermore, the described method clipped all positive rewards to 1 and all negative rewards to -1, which constrains the error derivatives and normalizes the rewards for all games.

The algorithm is run with an ϵ-greedy policy, where $\epsilon = 1$ decreases linearly over a fixed amount of frames until a small enough value is reached (e.g. $\epsilon = 0.05$). Actions are then taken either randomly with the probability of ϵ or as predicted from the target network with the probability $1 - \epsilon$. This promotes exploration at the beginning of the training, which gradually turns towards exploitation of the predicted action values (while still leaving room for exploration).

After the success of DQN to surpass human players in several Atari games, several improvements where proposed to improve the performance of the algorithm.

A. Deep Recurrent Q-Networks

A major challenge RL faces is the issue that single observations taken from the environment may not be sufficient to determine its current state. In the Atari game Pong, for example, a single frame encloses not all relevant information to infer the current state of the game, because it is unclear in which direction the ball is traveling. DQN address this issue in the preprocessing phase, where the last m frames (typically 3 to 5) are stacked to generate an image which encapsulates the missing information.

A problem arises if the agent requires more knowledge than the last m frames to detect the current state of the environment. Hausknecht et al. [25] suggest that most real-world applications of RL can be described as Partially Observable Markov Decision Processes (POMDP), where the agent can only partially recognize the underlying system state. An example of this problem can be seen in figure 4, which shows the Atari game Pong. The given information of a single frame is not sufficient to determine the direction in which the ball is traveling.

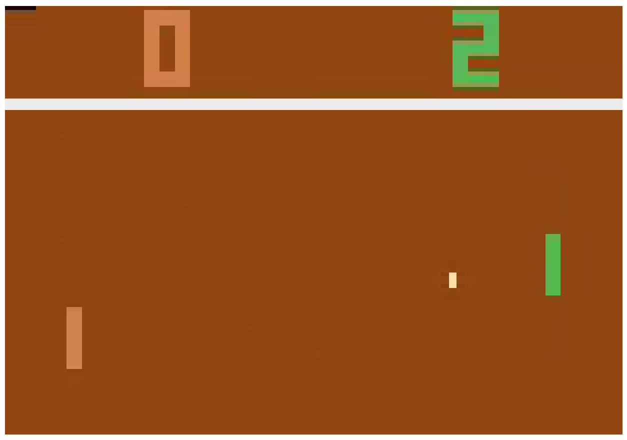

Fig. 4. Pong as an example for a POMDP. Here, the current state of the environment cannot be determined using only a single frame. It is impossible to infer the movement of the ball **(white)** without the context of previous frames.

Their work suggests a modification of the original DQN algorithm, which enables training on single frames by replacing the first fully-connected layer with a recurrent LSTM layer. The samples for training of the NN are selected by randomly choosing a sequence of experiences from the replay memory within an episode. Despite producing similar results as the DQN, the DRQN is able to generalize its policies better than DQN.

B. Double Deep Q-Networks

Hasselt et al. [11] demonstrate that DQN have a tendency to overestimate Q values, which diminishes the training performance and may lead to suboptimal policies [24, Chapter 7]. This is caused by the max operator in Q-learning and DQN, as the same values are used to select and to evaluate an action. Hasselt [26] introduced Double Q-learning, where two estimates are used to determine the greedy policy and its value separately to improve the performance of Q-learning. Based on this idea, actions for the next state are chosen using the trained network but values of Q are selected from the target net.

By changing the DQN target function

$$Y_i^{DQN} = R + \gamma \cdot \max_{A'} Q(S', A'; \theta_i^-) \tag{3}$$

to the proposed Double DQN target function

$$Y_i^{DDQN} = R + \gamma \cdot Q(S', \arg\max_{A'} Q(S', A'; \theta_i); \theta_i^-) \tag{4}$$

the algorithm leads to less overestimation of the Q values and improved stability.

C. Prioritized Experience Replay

Another concept for further improvements was introduced by Schaul et al. [27] in the form of Prioritized Experience Replay (PER). While the NN in the original DQN is trained with uniformly sampled observations from the replay memory, PER liberates agents from regarding transitions with the same frequency as they are experienced. The main idea is to increase the probability of selecting observations from the memory buffer that are more important for the learning progress.

Two different models for PER are suggested, prioritization with the temporal-difference (TD) error, where the priority is calculated using the loss function of the NN and a stochastic prioritization, where a stochastic sampling method interpolates between TD prioritization and uniform random sampling.

A combination of the DDQN architecture and PER further increases the performance and outperforms the improved DDQN.

D. Dueling Networks

Dueling Networks, as introduced by Wang et al. [12], optimize the architecture of the utilized NN. The initial concept is composed of the idea to separate the representation of state values and action advantages. The convolutional layers of the DQN remain unchanged, while the first fully-connected layer is split into two streams of fully-connected layers. These streams represent the value scalar $\tilde{V}(S;\theta;\beta)$ and the advantage $|\mathcal{A}|$-dimensional vector $\tilde{A}(S,A;\theta;\alpha)$ respectively. An aggregation layer connects both streams to estimate the state-action value function $Q(S,A)$. In this case describes θ the weights and biases of the convolutional layer, while α and β are the parameters of the two streams of fully-connected layers. The complete Q-value function estimation is given by

$$Q(S,A;\theta,\alpha,\beta) = \tilde{V}(S;\theta,\beta) + \left(\tilde{A}(S,A;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{A'} \tilde{A}(S,A';\theta,\alpha)\right) \tag{5}$$

Weight updates of both streams are computed only using back-propagation and without any algorithmic modifications of the DQN.

The intuition behind the segregation of value and advantage function is that the extended architecture enables the agent to learn which states are valuable and allows the agent to disregard unimportant action-state pairs. Figure 5 shows the difference between the original DQN architecture and the proposed Dueling Network architecture.

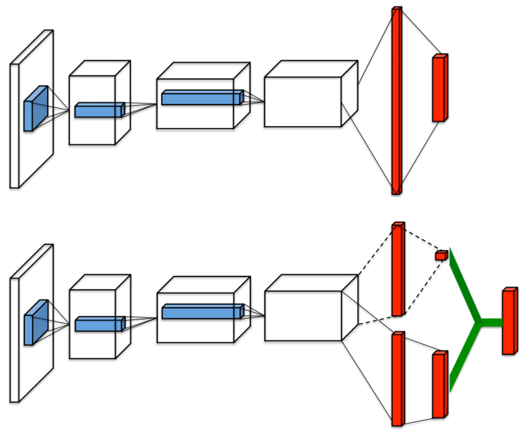

Fig. 5. The Network architecture of the classic DQN **(top)** in comparison to a Dueling Network architecture **(bottom)**, which separates the first fully-connected layer into two streams that represent the value and advantage functions.

E. Noisy Networks

Exploration in DQN is implemented by choosing actions randomly depending on a hyperparameter ϵ. The parameter decreases slowly over time to a small fixed value. This strategy is suitable for many basic environments, but even in simple cases, it requires adjustments to ensure efficient training.

Fortunato et al. [28] propose to add noise to the weights of fully-connected layers of the NN and to adjust the parameters of this noise during training. Two different methods of adding this noise are suggested [24, Chapter 7]:

- *Independent Gaussian noise:* For every weight in a fully-connected layer, a random value is drawn from a normal distribution. The parameters of the noise, μ and σ, are stored inside the layer and get updated through back-propagation.
- *Factorized Gaussian noise:* Two random vectors are used, respectively with the input and output size of the layer, to minimize the sampled amount of random variables.

F. Rainbow

All these recent improvements for DQN naturally raise the question whether these techniques can be combined to a type of ensemble agent. The proposed Rainbow model from Hessel et al. [29] merges several of the most important advancements, which results in the current state-of-the-art architecture for value-based learning methods in the Q-learning family.

Besides the already presented designs for DDQN, PER, Dueling Networks, and Noisy Nets two more extensions are used for Rainbow, namely Multi-step learning and Distributional RL.

Multi-step learning refers to the idea of R. Sutton [30] to unroll the recursive equation of the Bellman update used in Q-learning. Assuming that the action at step $t + 1$ was roughly optimal, the $\max$ operator can be disregarded. This n-times unrolled equation can be used in the weights update of the NN to speed up training.

The Distributional RL extension as suggested by Bellemare et al. [31] addresses the issue which is caused by the expected values in the Bellman update. By trying to predict the expected value for an action, the underlying distribution is disregarded, which can lead to uncertainty for a reliable prediction. This problem can be overcome by further adjustment of the loss function.

Merging all these extensions into one architecture, produces an agent which achieves high score results in many Atari environments.

IV. Limitations of Deep Q-Networks

Despite exceeding in tasks with large observation spaces, all DQN architectures fall short when confronted with large and/or continuous action spaces. This can be attributed to the nature of Q-learning, because it requires finding the action, which maximizes the action-value function. In a continuous setting, actions would need to be discretized to calculate the Q-values. This would be infeasible in most real-world continuous control applications like motion control in robots, where a high degree of freedom combined with a fine discretization of all possible actions would lead to a rapid inflation of the action space.

Policy Gradient (PG) methods like an actor-critic (A2C) agent would be more suitable for this kind of setting. It can be seen as a hybrid between policy based and value based methods, where the agent includes two different component: An actor and a critic. The actor observes the current state and determines the best possible action to take. The critic is used for evaluation by predicting a score which represents the error of the actor [24, Chapter 10].

V. Conclusion

Despite the drawbacks of DQN in large and/or continuous action spaces and better available solutions for such tasks like A2C, DQN continue to prevail in applications with limited action spaces such as or news recommendation [32] or traffic light timing [4]. Due to various extensions, the performance of DQN were further improved, resulting in the Rainbow model which combines several approaches to yield state-of-the-art results for many applications.

References

[1] J. C. Caicedo and S. Lazebnik, “Active object localization with deep reinforcement learning,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2488–2496.

[2] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.

[3] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[4] L. Li, Y. Lv, and F.-Y. Wang, “Traffic signal timing via deep reinforcement learning,” *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, 2016.

[5] “Artificial intelligence: Google’s alphago beats go master lee sedol,” https://www.bbc.com/news/technology-35785875, accessed: 2019-01-17.

[6] OpenAI, “Openai five,” https://blog.openai.com/openai-five/, 2018, accessed: 2019-01-17.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[11] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, vol. 2. Phoenix, AZ, 2016, p. 5.

[12] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] F. Beaufays, "The neural networks behind google voice transcription," *Google Research blog*, 2015.

[15] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2018.

[17] S. B. Thrun, "Efficient exploration in reinforcement learning," 1992.

[18] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] C. C. Aggarwal, *Neural networks and deep learning*. Springer, 2018.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[22] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1422–1432.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[24] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[25] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR, abs/1507.06527*, vol. 7, no. 1, 2015.

[26] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, 2010, pp. 2613–2621.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[28] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[29] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *arXiv preprint arXiv:1710.02298*, 2017.

[30] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[31] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *arXiv preprint arXiv:1707.06887*, 2017.

[32] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 167–176.