Reversal Generative Reinforcement Learning

Po-Lung Wang[1]

[1]Taipei, Taiwan

October 11, 2024

Abstract

We introduce a simple method for enabling pure deep neural networks to engage in reinforcement learning. The approach treats state and action as input data and reward as output data. First, when neural nets observe state, the neural nets initialize and update actions using error backpropagation, with desired reward as target. Second, the agent (neural nets) executes the updated actions and learns the ensuing actual reward. The above learning process is a continuous iterative process until the agent's learned parameters approximates the real environment.

This method is applicable across diverse digital environments. Furthermore, it circumvents the need for handcrafted functions such as the Bellman function for propagating future reward to action, further leveraging the flexibility and capacity of deep neural networks. We substantiate the approach with experimental results and provide GitHub repository[1] in this paper.

1 Introduction

Deep reinforcement learning (Deep Q-Learning) [15] is one of the most popular algorithms in reinforcement learning. By leveraging the flexibility of deep neural network and the reward mechanism derived from the Bellman equation, it is widely used in areas such as robotics, gaming, and autonomous systems.

Deep reinforcement learning utilizes a deep neural network to map state (input) to Q-values for each possible action (output). During training, the Bellman equation is used to propagate future rewards to these Q-values, helping the network learn which actions are more valuable in the present state. Namely, the update rule for the Q-value of a state-action pair (s, a) is given by: $Q(s, a) \leftarrow Q(s, a) + \alpha\left(r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)\right)$ where α is the learning rate and γ is the discount factor representing how much future rewards are considered (see also Volodymyr et al. [14]).

[1] See: https://github.com/Brownwang0426/RGRL.

After training, an agent can determine the optimal action by selecting the action with the highest Q-value in the current state. From this perspective, the agent's action is directly derived from the output of the deep neural network, which is a straightforward and immediate process.

While theoretically sound, the Bellman equation is based on a handcrafted formula that assumes a perfect model of the environment's dynamics (i.e., how present state and actions affect future reward and how future rewards are distributed to actions). This assumption could be limiting in complex environments where modeling these dynamics explicitly is difficult, which more or less results in the fact that the choice of hyperparameter such as discount factor γ might notably affects the over performance of an agent (as shown by Henderson et al. [7]). Therefore, multiple works are proposed to introduce more flexibility.

For example, Policy Gradient Methods (by Sutton et al. [21]) were proposed to offer more flexibility compared to methods based on the Bellman equation. In these methods, the gradient is derived mathematically from the objective of maximizing the expected cumulative reward. However, the expected cumulative reward itself, along with the discount factor, relies on human expert design.

There is also work proposing RL frameworks that do not rely on the Bellman equation. For example, "Model-Free Episodic Control (MFEC)" by Blundell et al. [1] proposes an alternative to value function-based methods. MFEC utilizes a non-parametric memory (a table of previous experiences) of state-action-reward tuples, where each entry stores information from individual episodes. This allows the agent to make decisions based on direct look-ups of similar past experiences instead of relying on learned value functions.

There are also studies focusing on decision-making by agents solely through deep neural networks, without the use of explicit reward functions, discount factors, or look-up tables. Since the Bellman equation and other human-designed functions typically average, assign, or predict future rewards based on present states and actions, these methods could be replaced by deep neural network, modeled as $f(s, a) = r$, where s and a represent the input state and action, and r denotes the output reward. For example, a work by Yize et al. [2] employs gradient-based optimization through backpropagation in a recurrent neural network to adjust a time sequence of input data (e.g., energy usage) to minimize overall energy consumption in a building. Also, Miguel et al. [20] utilize similar gradient method to optimize wireless network configurations. Wang [22] extends these approaches by borrowing concept from Deep Dream (Mordvintsev et al. [16]) where the author iteratively refines input actions a via forward passes and backpropagation while freezing the neural network weights before executing actions a. Additionally, the author introduces a approach called 'Multi-Weight-Matrices Stochastic Gradient Descent (MWM-SGD),' wherein, unlike in Deep Dream, an agent leverages an ensemble of trained neural networks. The input data undergoes stochastic gradient descent across the ensemble to avoid local minima, thereby enhancing decision-making performance. However, the aforementioned decision-making works are conducted within a supervised learning framework, with limited exploration in reinforcement learning.

Aside from the previous works, there are also medical and neuroscientific studies that support the idea that human actions are not the result of spontaneous, instantaneous events in the brain but rather emerge from ongoing updates or gradual build-up of activity in certain brain areas, like the motor cortex, before an action is executed.

For example, Kornhuber et al. [11] founded Readiness Potentials (RPs), which are slow electrical brain signals that occur before intentional motor actions. RPs involve a gradual, slow increase in electrical activity that reflects the brain's preparatory processes leading up to the execution of a intentional action, such as lifting a hand or pressing a button (see also Deecke [4]).

Libet et al. [12] and Haggard et al. [6] highlight two distinct phases of RPs. In the early stage, the brain initiate general motor preparation. In the second stage of the RPs, the brain fine-tunes the actual motor commands in the primary motor cortex before executing it.

Schurger et al. [18] also present a model where neural activity builds up gradually prior to the initiation of intentional movements. It describes how motor actions are prepared and updated over time before execution.

Desmurget et al. [5] further discuss how motor control involves both efferent signals (motor command) and reafferent signals (motor feedback), illustrating how the brain uses feedback to adjust and refine motor actions using both signals iteratively before motor execution (see also Wolpert et al. [23]).

From this perspective, Deep Q-Learning's use of a deep neural network is a good approach. However, its viewing action as output of a neural network seems to conflict with the finding of the above medical and neuroscientific studies. Since, in Deep Q-Learning, after training or learning, the motor actions of an agent are the direct forward-feeding result of a deep neural net, it is more spontaneous and instantaneous than an ongoing update. Furthermore, it involves little iterative interaction between efferent signals (motor command signals) and reafferent signals (motor feedback signals) for fine-tuning motor actions before execution.

Given the limitations of the Bellman function, and the missing alignment with biological decision-making processes, this paper seeks to explore an alternative and biological possible approach to bridge the gap between reinforcement learning models and phenomenon observed in decision-making neural systems.

2 Algorithm

We first notice that the brain has many pathways from later layers back to earlier ones, and it could use these pathways in many ways to convey the information required for learning or error-backpropagation (as in Hinton [8]). This suggests potential parallels between the mechanisms of forward propagation and error backpropagation in deep neural networks and the reafferent (motor feedback) and efferent (motor command) signals found in neuroscience.

If an agent's actions were to be influenced by these two signals, the origin of actions should likely reside somewhere other than the output layer, allowing sufficient layers for forward propagation and error backpropagation to emulate these signals. To explore this hypothesis, we propose a radical approach: postulating that the source of actions might lie at the input layer, rather than the output.

Consequently, we model the agent as a deep neural network, where states and actions are treated as input data and rewards as the output. However, updating actions via backpropagation in this manner is prone to encountering local minima, as the agent's actions are factually performing gradient descent on an error surface created by a single deep neural network. To address this issue, we incorporate MWM-SGD, proposed by Wang [22], into the reinforcement learning framework.

We treat an agent as a neural ensemble $\mathbb{W}$ where $\mathbb{W}$ comprises m homogeneous neural networks and $\mathbb{W} = \{W_1, W_2, \dots, W_m\}$. We view the state s as an input vector, the actions a as a sequence of input vectors, and the reward r as an output vector[2].

For each interval, when the neural ensemble observes state s, the neural ensemble initializes actions a where $a = \{a_1, a_2, ..., a_t\}$. Then the neural ensemble iteratively and randomly selects $W_i \sim \mathbb{W}$ and updates a by error backprop:

$$a \leftarrow a - \beta \frac{\partial}{\partial a} E\Big(\acute{r}, f\big(W_i, (s, a)\big)\Big) \tag{1}$$

where β is the updating rate, $\acute{r}$ is the desired reward, E is the error or loss function, $\leftarrow$ is error backprop, imitating efferent signals (motor command signals), and $f(\cdot)$ is the forward function outputting temporal reward, imitating reafferent signals (motor feedback signals).

Then, the neural ensemble executes actions a, imitating the final stage in Readiness Potentials (RPs), and observes actual reward r, after which s, a and r are stored into long term experience replay buffer $\mathbb{D}$:

$$\mathbb{D} \leftarrow \mathbb{D} \cup \{(s, a, r)\} \tag{2}$$

Upon sufficient $\mathbb{D}$, for each W_i, the neural ensemble iteratively and randomly selects $s, a, r \sim \mathbb{D}$ and updates W_i by error backprop:

$$W_i \leftarrow W_i - \alpha \frac{\partial}{\partial W_i} E\Big(r, f\big(W_i, (s, a)\big)\Big) \tag{3}$$

where α is the learning rate. In offline learning, Statement 3 is performed after each episode[3], and updated $\mathbb{W}$ will be used in the next episode, forming a circulation.

The above method is based on the following numerical reason: upon observing state s, an agent's actions a are performing stochastic gradient descent

[2] For demonstration purpose, we shape state, actions and reward into vectors.
[3] Or after a predetermined number of episodes, known as batch offline learning.

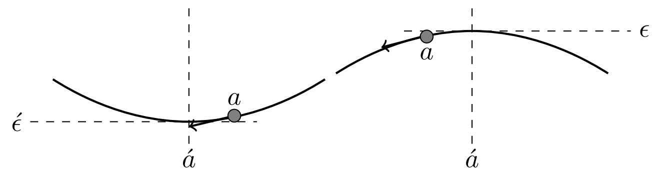

Figure 1: **Left**: Suppose the neural nets are initialized as $\inf_{s,a} E\big(\acute{r}, f(s,a)\big) \geq \acute{\epsilon}$ where $\acute{\epsilon}$ is positive, the actions a of the agent will roll into the low point $\acute{a}$ where the agent prematurely thinks $\acute{a} = \text{argmin}_a E\big(\acute{r}, f(s,a)\big)$ and $f(s,\acute{a}) \approx \acute{r}$. **Right**: However, after learning $f(s,\acute{a}) = r$ where $r \neq \acute{r}$ and $E\big(\acute{r}, f(s,\acute{a})\big) = \epsilon > \acute{\epsilon}$, under similar s, the actions a of the agent will roll away from the high point $\acute{a}$, encouraging the agent to find or explore alternative path to desired reward in a numerical way.

among error surfaces[4] created by neural ensemble $\mathbb{W}$ for $f(s,a)$ to approximate desired reward $\acute{r}$, as shown in Figure 1 (left). However, when the environment later reveals the actual reward $r \neq \acute{r}$, $\mathbb{W}$ will adjust by learning s, a, r so that $f(s,a)$ will approximate actual reward r where $r \neq \acute{r}$ and level up the previous supposed minimum. After learning, upon observing state similar to s, the agent's a will be performing stochastic gradient descent among updated error surfaces created by updated neural ensemble $\mathbb{W}$ for $f(s,a)$ to approximate desired reward $\acute{r}$. However, since the error surfaces are updated, the agent's gradient of a is also updated and will roll away form the previous supposed minimum as shown in Figure 1 (right).

Upon this trial and error circulation, $\mathbb{W}$ are constantly changing as well as the error surfaces upon which actions a are updated. The learning process will continue until supposed minima converge to actual minima and actions a constantly reach both supposed and actual minimum, leaving little error to be propagated for the neural nets to learn.

To clarify this intuitively: upon observing a particular state (s), an agent initializes, updates and executes actions (a) based on its learned parameters, represented by the neural network weights ($\mathbb{W}$), with the goal of achieving a desired reward (i.e., making $f(s,a)$ approximate $\acute{r}$ under $\mathbb{W}$). However, when the environment subsequently indicates that the initial assumption was incorrect, the agent observes the actual reward (r) and adaptively updates its parameters (utilizing s, a and r to train $\mathbb{W}$). This adaptive learning encourages the agent to select different actions in similar future states. This iterative process continues until the agent's parameters accurately aligns with the real environment.

Upon this view, the initialization of the agent's actions a can be viewed as analogous to ϵ-greedy method for balancing exploration and exploitation. Specifically, when $a \sim N(0, \sigma^2)$ (i.i.d.), the variance σ^2 functions as the ϵ in ϵ-greedy approach. A larger σ^2 induces greater variability in the agent's actions, even after fine-tuning, thereby facilitating exploration beyond local min-

[4] Or non-convex high-dimensional error surfaces.

ima. Conversely, a smaller σ^2 reduces action variability, thereby encouraging exploitation around known local minima.

Since the actions of an agent are iteratively generated from input layer of deep neural networks rather than output, we refer to this model as Reversal Generative Reinforcement Learning (RGRL).

We present the full algorithm in offline learning in Algorithm 1:

Algorithm 1 RGRL with offline learning

```
1: Initialize homogenious neural ensemble $\mathbb{W}$ where $\mathbb{W} = \{W_1, W_2, \ldots, W_m\}$
2: Initialize desired reward $\acute{r}$
3: Initialize long term experience replay buffer $\mathbb{D}$
4: for each episode do
5:     Initialize short term experience replay buffer $D$
6:     Initialize environment
7:     while not done do
8:         Observe state $s$
9:         Initialize actions $a$ where $a = \{a_1, a_2, ..., a_T\}$
10:        for each iteration do
11:            Select $W_i \sim \mathbb{W}$
12:            Perform back-prop $a \leftarrow a - \beta \frac{\partial}{\partial a} E\Big(\acute{r}, f\big(W_i, (s, a)\big)\Big)$
13:        end for
14:        Execute action $a_1$ where $a_1 \in a$
15:        Observe actual reward $r$
16:        Store $s, a_1, r$ to $D$
17:    end while
18:    Return agent performance to human for inspection
19:    Sequentialize $D$ to $\mathbb{D}$
20:    for $W_i$ in $\mathbb{W}$ do
21:        for each iteration do
22:            Select $s, a, r \sim \mathbb{D}$ where $a = \{a_1, a_2, ..., a_t\}$ and $t < T$
23:            Perform back-prop $W_i \leftarrow W_i - \alpha \frac{\partial}{\partial W_i} E\Big(r, f\big(W_i, (s, a)\big)\Big)$
24:        end for
25:    end for
26: end for
```

where sequentializing D to $\mathbb{D}$ is defined as: in D, for s in time step τ and r in time step $\tau + t$, $a = \{a_{\tau+1}, a_{\tau+2}, ..., a_{\tau+t}\}$, and we store s, a, r to $\mathbb{D}$ for each T under fixed t. Namely, we use fixed-size rolling window to subtract pairs of s, a, r from D into $\mathbb{D}$.

Furthermore, there are a few important additions to the actual implementation of the algorithm that are not reflected in Algorithm 1 for the purpose of clarity:

1. **Prioritized Experience Replay**: In practice, since it is impractical to train neural networks on the entire dataset $\mathbb{D}$ or to set the number of iterations greater than the size of $\mathbb{D}$, as indicated in line 21 in Algorithm 1. We implement "Prioritized Experience Replay (PER)" introduced by Schaul et

al. [17] and train neural nets on part of $\mathbb{D}$ by setting the number of iterations smaller than the size of $\mathbb{D}$. More concretely, in each iteration, for a sample k in $\mathbb{D}$ in line 22 in Algorithm 1, its probability of being selected by W_i is:

$$P_i(k) = \frac{(|\delta_{ik}| + \epsilon)^\gamma}{\sum_\zeta (|\delta_{i\zeta}| + \epsilon)^\gamma} \tag{4}$$

where $|\delta_{ik}|$ represents the sum of the absolute differences between the actual reward and the forwarded reward for W_i on sample k, ϵ is a small positive constant preventing edge cases where a sample will not be selected once its differences become zero, and γ is the exponent (as in Schaul et al. [17]). Consequently, surprising experiences are prioritized by assigning them greater chance being selected during the stochastic gradient descent learning process[5].

2. **Elastic Weight Control**: While PER helps select important samples during training, it still leaves some samples unselected, which means the risk of catastrophic forgetting persists, especially in continuous learning scenarios. To mitigate this risk, we further implement "Elastic Weight Control (EWC)" introduced by Kirkpatrick et al. [10]. Namely, the error function in learning process in line 23 in Algorithm 1 is modified as:

$$E\Big(r, f\big(W_i, (s,a)\big)\Big) = E\Big(r, f\big(W_i, (s,a)\big)\Big) + \sum_j \frac{\lambda}{2} F_{ij}\big(W_{ij} - W^*_{ij}\big)^2 \tag{5}$$

where W_{ij} represent the j^{th} weight parameter in the i^{th} neural net in the ensemble, W^*_{ij} is the last learned weight parameters, F is the the diagonal Fisher information matrix for W^*_{ij} on the entire dataset $\mathbb{D}$ treated as a single batch, and λ is the elastic rate[6] where $\lambda \in \mathbb{R}$.

3. **Drop-out**: To further increase the effect of MWM-SGD, we implement "Dropout"[7] introduced by Srivastava et al. [19] in each iteration in line 12 and 23 in Algorithm 1. Since applying Dropout to a neural network amounts to sampling a "thinned" network from it (as in Srivastava et al. [19]), applying Dropout to every neural net in the neural ensemble amounts to implicitly increasing the size of the ensemble without actually adding any neural network, allowing actions a to escape local minima more easily during stochastic gradient descent.

[5] We set ϵ to 0.1^6, exponent γ to 1 and sample with replacement.
[6] We set λ to 1.
[7] We set drop rate to 0.1^3.

3 Experimental Results

To validate the above methodology, given that the process exclusively involves neural networks, a wide variety of architectures can be employed as substitutes for $f(s,a)$, provided that error propagation can be effectively traced back to the input layer. For the sake of simplicity and demonstration purpose, in this work, we evaluate the approach using Recurrent Neural Networks (specifically GRU) by Cho et al. [3] and LSTM by Hochreiter et al. [9] independently[8]

We present the experimental results concerning the the impact of the neural ensemble—specifically, the size of m in $\mathbb{W}$ where $\mathbb{W} = \{W_1, W_2, \ldots, W_m\}$—on GRU and LSTM.

We slightly adjust our algorithm in Algorithm 1 to allow batch offline learning. Namely, we implement a strategy where the agent begins the learning process only after completing a predetermined number of episodes[9], allowing the agent adequate time to accumulate sufficient data derived from its own erroneous judgments and to identify flaws in the current experience.

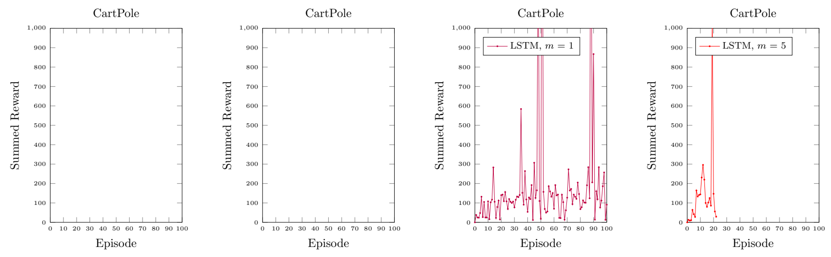

Figure 2: Performance comparison under different neural types and size of m

Figure 2 shows the impact of the size of m. By increasing the size of m, namely the size of the neural ensemble, the peak value of the reward obtained by an agent surges.

The process of assigning or predicting future rewards based on the current state and actions can be modeled by deep neural networks, thereby mitigating the reliance on the Bellman equation or other expert-designed functions.

4 Future Works

In this work, we introduce a basic frame work purely for deep neural networks to engage in reinforcement learning. However, in the present introductory work,

[8] To enhance the model's ability to capture the causal dynamics of the environment, the hidden input is treated as the present state, and the input corresponds to the present action while the output represents the present reward, and the hidden output serves as the next state. And the update of GRU and LSTM in line 23 in Algorithm 1 becomes: $W_i \leftarrow W_i - \alpha \frac{\partial}{\partial W_i} E\Big((r, \acute{s}), f\big(W_i, (s, a)\big)\Big)$ where $r = \{r_1, r_2, ..., r_t\}$, $\acute{s} = \{s_2, s_3, ..., s_{t+1}\}$, $s = \{s_1\}$ and $a = \{a_1, a_2, ..., a_t\}$.

[9] We set the interval to be 10 episodes.

we present only offline learning. Furthermore, due to limitation in hardware resource, we take vectorized state as input rather than raw image. Also, the neural ensemble shares identical desired reward $\acute{r}$ and comprises homogeneous neural net, which might not entirely true for human. Finally, We postulate that, by deploying the same agent in multiple identical digital environments simultaneously or by deploying multiple agents (with identical $\mathbb{W}$) in the same physical environment concurrently[10], the data gathering process can be accelerated.

5 Conclusion

In this paper, we present a reinforcement learning model that is both model-free and value-function-free, replacing the traditional Bellman equation purely with deep neural networks. We try to find an alternative approach to mainstream perspectives: first, we demonstrate that deep neural network is capable of deriving information from both their output and input layers. Second, we demonstrate that actions of an agent can be derived from input layer of neural networks using iterative forward and backward propagation. Third, we show that by iteratively training the networks using self-generated data from its own input and its interaction with environment, the networks can dynamically adapt to changing environmental conditions. Fourth, we demonstrate that the proposed method can be seamlessly integrated with other state-of-the-art architectures or methodologies.

Acknowledgments

We thank reviewers for their insightful discussions and feedback.

Conflict of interest statement

No funding was received to assist with the preparation of this manuscript.

References

[1] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv:1606.04460v1*, 2016.

[10] This approach is a little different from federated learning by McMahan et al. [13] in that it is the global model doing training and local models doing data gathering. However, we don't rule out the possibility of incorporating original federated learning such as the global model of this paper becomes the local model of McMahan et al. [13], enabling cross border data gathering and training.

[2] Yize Chen, Yuanyuan Shi, and Baosen Zhang. Modeling and optimization of complex building energy systems with deep neural networks. *51st Asilomar Conference on Signals, Systems, and Computers*, 2017.

[3] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[4] L. Deecke. Planning, preparation, execution, and imagery of volitional action. *Cognitive Brain Research, 3(2), 59-64*, 1996.

[5] Michel Desmurget and Scott Grafton. Forward modeling allows feedback control for fast reaching movements. *Trends in Cognitive Sciences – Vol. 4, No. 11*, 2000.

[6] P Haggard and M Eimer. On the relation between brain potentials and the awareness of voluntary movements. *Experimental Brain Research, 126(1), 128-133*, 1999.

[7] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv:1709.06560v3*, 2019.

[8] G.E. Hinton. How neural networks learn from experience. *Scientific American*, pages 145–151, 1992.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation, Volume 9, Issue 8*, 1997.

[10] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Venessa, Guillaume Desjardinsa, Andrei A. Rusua, Kieran Milana, John Quana, Tiago Ramalhoa, Agnieszka Grabska-Barwinska, Demis Hassabisa, Claudia Clopathb, Dharshan Kumarana, and Raia Hadsella. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 2017.

[11] H.H. Kornhuber and L. Deecke. Brain potential changes in voluntary and passive movementsin humans: readiness potential and reafferent potentials. *Pflügers Archiv'S Historical Articles*, 2016.

[12] B Libet, C A Gleason, E W Wright, and D K Pearl. Time of conscious intention to act in relation to onset of cerebral activity (readiness-potential). the unconscious initiation of a freely voluntary act. *Brain, 106(3), 623-642*, 1983.

[13] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*, 2013.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature, 518(7540):529–533*, 2015.

[16] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. In *Archived from the original on 2015-07-03*. Google Research, 2015.

[17] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations (ICLR)*, 2016.

[18] Aaron Schurger, Jacobo D Sitt, and Stanislas Dehaene. An accumulator model for spontaneous neural activity prior to self-initiated movement. *Proceedings of the National Academy of Sciences, 109(42), E2904-E2913*, 2012.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[20] Miguel Suau, Alexandros Agapitos, David Lynch, Derek Farrell, Mingqi Zhou, and Aleksandar Milenovic. Offline contextual bandits for wireless network optimization. *arXiv:2111.08587v1*, 2021.

[21] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Conference on Neural Information Processing Systems (NIPS)*, 2000.

[22] Brown Wang. Deducing decision by error propagation. *Proceedings of Proc. of the Adaptive and Learning Agents Workshop (ALA)*, 2022.

[23] D.M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks Volume 11, Issues 7–8*, 1998.