Using fNIRS to Improve RL Agent Learning from Demonstrations

Saber Bahranifard, Ellery Buntel

Abstract

With recent advancements in using Neural Networks as nonlinear function approximators, Deep Reinforcement Learning (DRL) has been successfully applied to many complex stochastic zero-sum games[1–3]. One approach to boost agent's learning is to use human experience to train the agent [1]. However, there human demonstrations may come in different quality depending on the condition of the human player. That is, if the human player is paying less attention, this may disqualify the undergoing experience to be used for the RL agent's training. In this work, we are investigating this hypothesis that using fNIRS signals may shed a light on screening human experience to meaningfully help RL agent learn more efficiently.

For the purpose of this test, we are using 'MS-PacMan' game which is, also, a stochastic zero-sum game. It requires an agent to explore an unexpectedly changing environment in order to make progress and improve score.

A Double Deep Q Network (DDQN) is trained using demonstration data to teach the agent how to navigate in a randomly-generated world, eat pellets and avoid ghosts to maximize the return.

1 Introduction

For a given Reinforcement Learning (RL) problem, if the model of the environment, including the distributions of all transitions, was available, then finding an optimal policy at each state could be a trivial task. For example, a simple soft-greedy algorithm could be used to find the best action. However, in most cases, models are not known in advance. Thus, developing an algorithm that can train the agent to find an optimal path to the terminal state is a non-trivial task in the field of Reinforcement Learning. This problem becomes more complicated where high-dimensional sensory input such as vision need to be used. As an example, consider a video game in which the player as an agent needs to ingest raw pixels of the video frames of the game as input.

Reinforcement Learning algorithms developed to control agents in such environments with high-dimensional inputs need a lot of field experience to extract features from the sensory inputs. Also, solving Reinforcement Learning in dynamic and non-deterministic environments such as video games poses a big challenge to Reinforcement Learning algorithms. Deep-Q Learning has been used to train an agent to deal with high-dimensional sensory data in the dynamic and non-deterministic environment presented by Atari games[4].

However, RL agents have a tendency to struggle when they start training as they often have little to no information about their environment. In situations like this it can be helpful to give the RL agent a human demonstration of the task to learn from. Doing so dramatically increases the training speed of RL systems by bypassing much of the randomness of the initial training phase. But not every demonstration is created equal. The usefulness of each demonstration is dependent, of course, on the person giving the demonstration. If the demonstrator isn't paying attention, or is feeling overwhelmed then they are unlikely to provide a useful demonstration to the agent. In which case, it would be beneficial to be able to distinguish between good and bad demonstrations as you could prioritize better demonstrations over worse ones.

One way this could be accomplished is by deciding which demonstration is the best using additional physiological signals from the demonstrator. One potential source of such signals is brain imaging. Brain imaging devices like fMRI or fNIRS shine light through the outer layers of the brain in order to measure the levels of oxygenated blood present in different regions of the brain. These levels of oxygenated blood can then be used to infer information about the current state of the brain. In particular, researchers have gotten quite good at discerning between states of high workload and low workload. So, the question is, is high/ low workload a useful feature for deciding which demonstration is the best? The goal of this project is to answer this question by examining the impact of including mental workload as an input to a training RL agent.

Among many different types of games the ones with stochastic actions, which fall in the category of zero-sum games, are in the interest of this research to prove our hypothesis. These games are non-deterministic because at each time step, the agent's action heavily depends on an unexpected change in the game environment or unexpected move of other players. Also, zero-sum games are the ones where the agent's gain is directly equivalent to its opponent's loss and vice versa. A single variable value, which one team or agent tries to maximize and the opposing team or agent tries to minimize, is summed up as a score. Atari 2600 games are considered in the class of probabilistic zero-sum games [4]. For the sake of this project, we run our agent in the MS-PacMan environment. In this game, the agent tries to maximize the game score by eating pellets quickly and efficiently while ghosts try to minimize it by eating the agent first.

In this paper, section 2 provides a brief background of this work and methods used. Section 3 defines the framework of the reinforcement learning problem to be solved and details our approach while the experimental setup is explained in Section 4. The results, including a comparison with benchmark approaches, will be presented in Section 5.

Disclaimer: *This report is, in part, adopted from Ellery Buntel's report for CS-138 course, as well as Saber Bahranifard's report for the same course. We are collaborating in the same project with the same topic. Thus, the parts talking about fNIRS data is used from Ellery's report.*

2 Background and Related Work

Deep Q-learning has been used in conjunction with convolutional neural networks (CNNs) for environments with images as part of the observation since at least 2013 [4]. The approach is similar to traditional Q-learning, but as the state-space is far too large to tabulate, an approximation is learned; the CNN provides a better interpretation of an image, in this case the player's point of view in-game, and should be able to pick out the salient features of the input after training. The general update for Q-learning is given by

$$Q_\pi^{(i)}(s_t, a_t) \leftarrow Q_\pi^{(i-1)}(s_t, a_t) + \alpha\left(r_t + \gamma . \max_{a_{t+1}} Q_\pi^{(i)}(s_{t+1}, a_{t+1}) - Q_\pi^{(i-1)}(s_t, a_t)\right)$$

where we seek to optimize $Q_\pi(s_{t+1}, a_t)$. For deep Q-learning, the value approximations are given by an approximation function that learns a series of weights through a neural network. We further use a variant called Double Q-Learning, where two networks are trained at once, a behavioral network and a target network. The separation of the network used to choose an action and the network used to evaluate the action has been shown to reduce maximization bias and improve performance for DQNs [5].

Replay buffers provide the agent more flexibility in learning, as past transitions are revisited, potentially out of order. In this way, rare transitions that are nonetheless valuable learning opportunities can be learned from to a greater extent than a traditional agent. Prioritized replay buffers further improve performance by biasing the selection from the buffer towards transitions that resulted in the largest TD error [6]. Our use case further leverages the buffer as a repository for the demonstration data, which is then revisited continually as the agent improves.

In brain computer interfaces (BCI) research, fNIRS is a commonly used non-invasive brain imaging tool that detects activation in the brain [7, 8]. Put another way, fNIRS measures the level of oxygenated and de-oxygenated haemoglobin in the brain, and these levels can be used to determine how active the brain is[7]. In a previous work, these signals were used to design a learning system for playing piano, to measure mental workload, and as an input in many other interfaces[9]. It is found that despite the noise and complexity of brain signals they can be effectively utilized as an input to other systems. Recent work in BCI has also shown that fNIRS can be used to train a classifier that reliably distinguishes between states of low mental workload and states of high mental workload[10]. These classifiers will be a core part of this project.

Reinforcement learning has proved to be a very effective method for learning high complexity tasks such as video game playing[11]. However, as stated above, RL agents often struggle initially as most RL agents start off with very little information about their environment. To help alleviate this problem, human input is commonly used in RL models to increase training speed [12, 13]. In particular, human demonstrations have been shown to be effective at increasing the learning speed of RL agents [11].

BCIs have been used as an input to RL agent training [14]. This work was done using EEG, which is another non-invasive brain imaging system that has higher temporal resolution than fNIRS, but lower spatial resolution [15]. It also incorporates human feedback, not human demonstrations, into its RL system. Another works on connection between RL and BCI in which PacMan has been used as a domain are reported in [8, 11]. So far, our literature reviews show that there is no work showing connection between RL and fNIRS data and our work is a forefront of this research.

3 Technical Approach

Most of the time, in real-world applications, it is costly to let the agent learn from the scratch by solely interacting with the environment. In such situations, an alternative path is to pre-train agents on demonstration data gathered from an expert's experience, if accurate simulators are not available. However, training agents using behavior cloning in a supervised learning fashion could result in a biased model in which agent greedily imitates expert's trajectories, giving rise to an accumulated error which gradually derails agent from learning demonstration states (co-variate shift) [16].

In search for an optimal policy by imitating a demonstrator, a modification to the behavior cloning approach is to pre-train agents using a batch of uniformly sampled examples from expert's demonstration and guide the exploration policy by adapting an occupancy measure that closely matches demonstration policy [16]. To achieve this goal, a Prioritized Replay Buffer can be used along with a value-function learning algorithm [6].

The main goal of this project is to evaluate fNIRS' usefulness as a potential input to an RL agent which is learning from demonstrations. The human demonstrations are weighted by a classifier on the expert's neural data. States of high mental workload often induce mistakes in participants[10] so we are making the assumption that demonstrations given in that state will be less useful. The hypothesis is that the experiences in which the expert is entering a state of high mental workload must be given a lower priority to improve training speed of the agent.

3.1 Agent

In this project, we use a Q-learning algorithm to motivate the agent to learn from demonstration and satisfy Bellman equation:

$$Q^*(s,a) = \mathbb{E}\Bigg[\Bigg(r(s,a) + \gamma . \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')\Bigg]$$

in which $Q^*(s,a)$ and $r(s,a)$ are the optimal value function and assigned reward to state s and given action a, respectively. $P(s'|s,a)$ is the transition distribution used to select the next state based on current state and selected action. The optimal policy $\pi(s)$ is acquired by $\pi(s) = \arg\max_{a \in A} Q^*(s,a)$. In order to approximate the value function, we use a Deep Neural Network as the function estimator. Called Deep Q-Network (DQN) [4], this network updates value function through each training iteration:

$$Q_\pi^{(i)}(s_t, a_t; \theta) \leftarrow Q_\pi^{(i-1)}(s_t, a_t; \theta) + \alpha\bigg(\widehat{Q}_\pi^{(i-1)}(s_t, a_t) - Q_\pi^{(i-1)}(s_t, a_t; \theta)\bigg)$$

in which $Q_\pi(s_t, a_t; \theta)$, called the behavior function, is approximated by the DQN from the underlying pattern of training samples, and $\widehat{Q}_\pi(s_t, a_t)$ is the target value. The parameters of the behavior network are updated on each iteration of the training. $\widehat{Q}_\pi(s_t, a_t)$ is typically calculated using a Temporal Difference (TD) value of the current Q-values.

DQN models suffer from model bias problems because the same approximated Q function is used for action selection and value prediction. A Double DQN model (DDQN) [1] uses a separate Q-Network to approximate target values, $\widehat{Q_\pi}(s_t, a_t; \widehat{\theta})$. Parameters of the target network are updated with frequency of τ with values learned for parameters of behavior network.

The agent samples demonstration trajectories from the replay buffer during the pre-training phase and then updates the network parameters with TD loss and a supervised loss. The TD loss helps the agent learn from interaction by the environment and satisfies the Bellman equation. This loss is implemented as sum of a 1-step TD loss, $J_{DQ}(Q)$, and an n-step TD loss, $J_n(Q)$:

$$J_{DQ}(Q) = \mathbb{E}\Bigg[\bigg(r_t + \gamma. \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}; \widehat{\theta}) - Q_\pi(s_t, a_t; \theta_{i-1})\bigg)^2\Bigg]$$

$$J_n(Q) = \mathbb{E}\Bigg[\bigg(\sum_{i=0}^{n} \gamma^i r_{t+i} + \gamma^n. \max_{a_{t+n}} Q_\pi(s_{t+n}, a_{t+n}; \widehat{\theta}) - Q_\pi(s_t, a_t; \theta_{i-1})\bigg)^2\Bigg]$$

The supervised loss, $J_E(Q)$, helps the agent learn the policy from demonstrator and is formulated as a margin classification loss:

$$J_E(Q) = \max_{a_t}\Bigg[Q_\pi(s_t, a_t; \widehat{\theta}) - l(a_t, a_{E,t})\Bigg] - \widehat{Q_\pi}(s_t, a_{E,t}; \widehat{\theta_{i-1}})$$

The total loss, $J(Q)$, is regularized with an L2 regularizer to prevent behavior network parameters to overfit to demonstration data:

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)$$

After pre-training, which solely samples target values from demonstration data, the agent enters the training phase in which it interacts with environments and adds self-collected trajectories to the replay buffer until the buffer overflows. Then it overwrites the data on top of the older samples, but needs write-protect demonstration data stored in the buffer.

To sample the data from the replay buffer during the training phase, a prioritized replay buffer [6] is used to sample from both demonstration and exploration data proportional to ϵ_d and ϵ_a, respectively. In the training phase, the supervised loss is turned off, but all the other losses are applied to both demonstration and exploration data with the priority proportion[1].

3.2 Data Collection

As mentioned in earlier sections, there is a wide range of visual game options that can be considered as the main task on which we can evaluate the learning performance of an agent. However, to successfully evaluate our hypothesis, in this project, the task assigned to must be satisfy the following minimum requirements:

1. Must engage in both the RL agent and human player to a reasonable extent (e.g., a non-deterministic game)

2. Mush take long enough for the changes in the participants' brain state be detected by the relatively poor temporal resolution of fNIRS.

3. Must cause increase in mental workload.

It is difficult to accurately quantify the last requirement but should keep in mind as a rule of thumb that a task that is not stimulating enough, will not produce interesting data. Given the above requirements, most of zero-sum stochastic Atari Arcade games are good to choose. Among them, MS-PacMan is chosen for the purpose of this experiment. It fulfills all the above conditions according to [8].

The neural data required for this experiment is collected from subjects using an fNIRS machine while the subjects are playing game of PacMan. The fNIRS machine has four channels that rest on the forehead of the subject and measure haemoglobin levels in their frontal lobe of brain. We can then send this data to a binary mental workload classifier that distinguishes between low and high workload recently developed by [10]. The output from that classifier is then used to prioritize some demonstrations over others based on the mental workload.

One difficulty to this approach is that the classifier in question has to train on the participant's data in order to be reliably accurate. Meaning that we need labelled neural data, in addition to the PacMan data, from each of the participants in the study. The brain waves we record while they play PacMan won't be labelled so we will either need to gather some n-back[10] neural data in addition to the PacMan data, or we will have to use participants of a previous study whose neural data we already have [10] which significantly simplifies data collection.

Ideally, we would try using the raw fNIRS signal as an input rather than the output of the mental workload classifier. The raw fNIRS signal is complex so it would be more difficult for an agent to use it than the output of the classifier, but it may still improve the network's performance. As a future goal, we will plan to integrate the demonstration weighting algorithm with agent network to concurrently learn the weights, as well as train the agent.

The neural classifier that we will use to assign a priority value to each demonstration is a deep neural network that is trained in three stages for each participant. The first stage involves training the network on generated synthetic neural data. The second stage involves training the network on all participants other than the target participant. Finally, in the third stage, we train the classifier on the target participant's data. This method has proven effective at distinguishing between high and low mental workload[10]. Using this method we can learn a probability value that represents the likelihood that the subject is in a state of high mental workload. This value can then be used to rank or weight the value of each demonstration.

4 Experimental Setup

In this section, we detail the PacMan environment that we used to train our agent on, `MsPacman-v0` along with the implementation details.

4.1 Environment

As discussed earlier, PacMan game not only fits into the *Markov Decision Process* (MDP) framework, but also satisfies the fNIRS data collection requirements. To this end, we use `MsPacman-v0` environments frin *OpenAI gym*:

1. *observation* is an RGB picture of size $210 \times 160 \times 3$. Each frame represents:
    - the position of the PacMan and ghosts
    - the position of rewards and goals
2. *reward* is a scalar float value which represents the score gained by the agent
3. *action* is a scalar integer with only 9 possible values:
    - 0 — "no move"
    - 1 — "move right"
    - 2 — "move right"
    - 3 — "move left"
    - 4 — "move down"
    - 5 — "move upright"
    - 6 — "move upleft"
    - 7 — "move downright"
    - 8 — "move downleft"

Like other Atari games, feeding the raw Atari frames to the Deep-Q Network (DQN) mopdel can be computationally expensive. Thus, a basic pre-processing step is required to reduce the dimensionality of input data [4]. First, we convert RGB colors to gray-scale, down-sample the frames, and eventually crop them to reach to an 84×84 image. Also, we stack every 4 frames to let the agent have a sense of speed and direction of each ghost. Thus, The input to the CNN layer of the DQN consists is an $84 \times 84 \times 4$ image. The Convolutional Layers are designed as per [4].

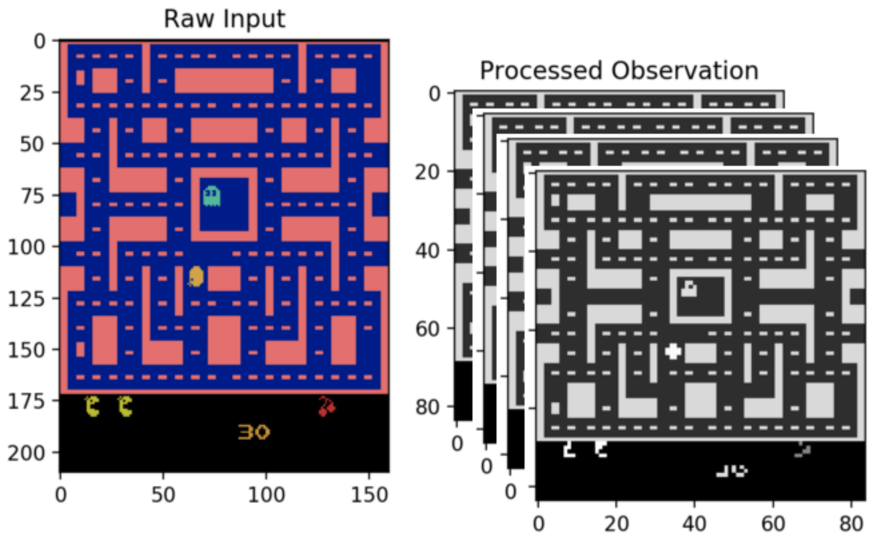

Figure 1: Pre-processing of raw PacMan frames

4.2 Training the RL Agent

We used Deep Q-Learning to approximate value functions directly from raw pixels. Using a Convolutional Neural Network (Figure 2), the $210 \times 160 \times 3$ frames of screen plays are converted into a dense state vector and linear layers for processing other features such as ghost locations. We will feed both outputs into another linear layer to estimate Q-values for each of the 9 separate actions.

The batch of screen plays are sampled either from demonstration data or self-play experiments. In either case, they are sampled from the replay buffer based on the priority discussed in the 3 section. Calculated Q-values at the output of both behavior and target networks are then used to compute the loss

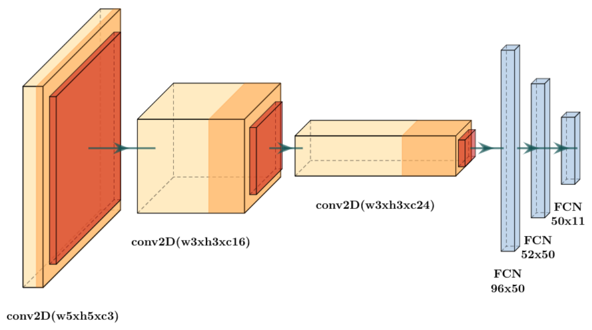

Figure 2: The Deep ConvNet used to approximate value functions

function in both pre-training and training processes (Figure 3). On every τ iterations, the target network parameters are updated from parameter values calculated for the behavior network.

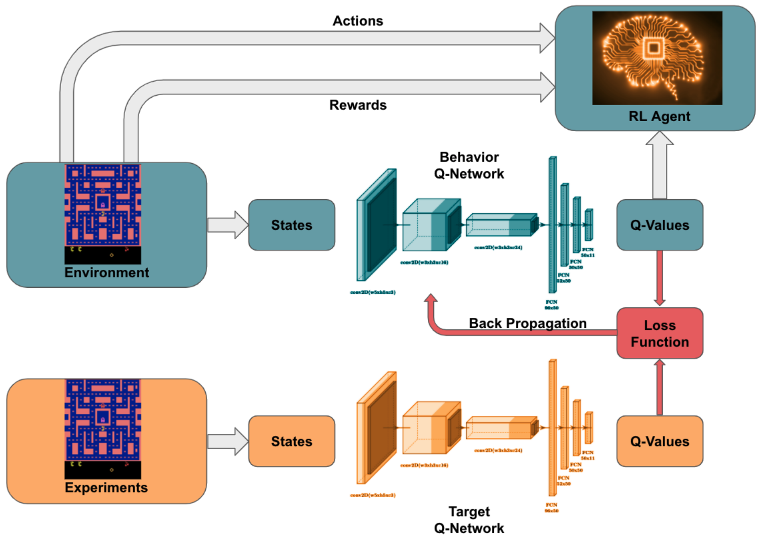

Figure 3: The full training loop of the Dual Deep Q-Network

We are using TensorFlow Agents [1] API as the framework for this project. This framework is very well integrated and makes RL agents developments easy. TF-Agent is also well integrated with Reverb [2] replay buffer from DeepMind. In this work, We use Reverb in prioritized mode to apply the classified mental work load weights to stored human experiences.

[1]`https://www.tensorflow.org/agents`
[2]`https://deepmind.com/research/open-source/Reverb`

5 Experimental Results

We will evaluate the results of this project by comparing the training rates and performances of DQfD networks that take fNIRS as an input, and networks that don't. In particular, we will look at which of the networks trains more quickly at the beginning of the training phase. If the DQfD network with fNIRS learns faster than the one without, then we can conclude that the fNIRS input is useful for the RL agent.

Since, at the time of writing the first report of this project, the fNIRS data is not ready to apply, yet, we are just showing the baseline performance of the DDQN model on MS-PacMan game. The average return goes up after first few thousands iterations rises but will not pick till $150,000$ iterations are passed.

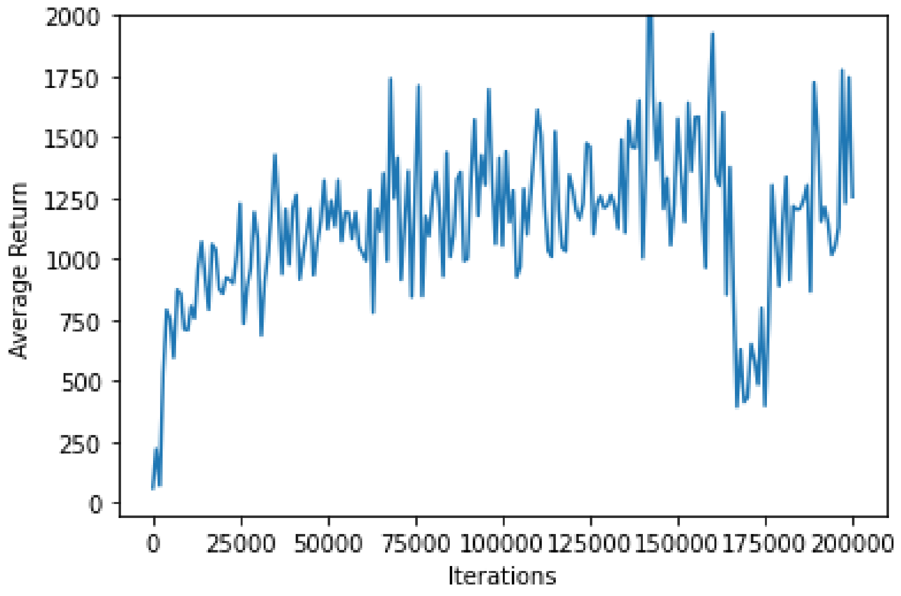

Figure 4: Average Returns over iterations.

6 Conclusion

In this project we presented a framework to increase the speed of learning for an RL agent using a combination of expert data and data generated by an agent. This framework will be used to import human experience which is weighted using a mental workload classifier algorithm to guide the agent to use more appropriate trajectories.

Given the complexity of the project and the need for data collection, we could not finish the experiment over the course of the current semester but we keep working on this project through the next semester to get the results and see if fNIRS data can help the RL agent learn faster, or our hypothesis will be rejected. For this project, all the codes can be found under this colab notebook [3]. Also, videos of the RL agent playing PacMan game before [4] and after[5] training are posted online.

7 Future Work

Below is a list of actions that we need to take to conduct the research in the future:

1. Full data collection

   After we are finished with the IRB application process we can start formally gathering data. The target number of participants can be decided later, but roughly we will need between 7 and 15 participants. Due to individual differences in head shape and hair, fNIRS data collection occasionally results in unusable data so we will have to plan to collect reasonably more data. The whole process should take a few weeks.

2. Analysis of full data set

   After collecting the full data set we can analyze the data, and more formally infer whether or not fNIRS is a useful signal for the RL agent. At this point we will also be better able to decide whether or not adding raw fNIRS data as an input to the agent is also worth trying.

3. Experiment with raw fNIRS data

   If the analysis of the first set of data suggests that fNIRS was a useful signal for the RL agent, then we might try giving the participants' raw neural data to the agent rather than the output of our mental workload classifier. It would be interesting to see whether or not the agent could learn to use the more complex raw fNIRS data as an input.

[3]https://colab.research.google.com/drive/1PoKqydDG9hfRJzpMqm6EYQbsJDSmXwyb#scrollTo=NxtL1mbOYCVO

[4]https://drive.google.com/file/d/1iZrw_aC2MC7J4X-xgkhPSRx6yCGfGObl/view?usp=sharing

[5]https://drive.google.com/file/d/1J5s1KJ6hPAWqxng6G4JigLsnKj2lmB76/view?usp=sharing

References

[1] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[3] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

[4] 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.

[5] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[6] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[7] Valentina Quaresima and Marco Ferrari. Functional near-infrared spectroscopy (fnirs) for assessing cerebral cortex function during human behavior in natural/social situations: a concise review. *Organizational Research Methods*, 22(1):46–68, 2019.

[8] Audrey Girouard, Erin Treacy Solovey, Leanne M Hirshfield, Krysta Chauncey, Angelo Sassaroli, Sergio Fantini, and Robert JK Jacob. Distinguishing difficulty levels with non-invasive brain activity measurements. In *IFIP Conference on Human-Computer Interaction*, pages 440–452. Springer, 2009.

[9] Alexa Bosworth, Matthew Russell, and Robert JK Jacob. Update of fnirs as an input to brain–computer interfaces: a review of research from the tufts human–computer interaction laboratory. In *Photonics*, volume 6, page 90. Multidisciplinary Digital Publishing Institute, 2019.

[10] Liang Wang, Zhe Huang, Ziyu Zhou, Devon McKeon, Giles Blaney, Michael C Hughes, and Robert JK Jacob. Taming fnirs-based bci input for better calibration and broader use. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 179–197, 2021.

[11] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John

Agapiou, et al. Learning from demonstrations for real world reinforcement learning. *CoRR*, 2017.

[12] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16, 2009.

[13] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *NIPS*, 2013.

[14] Tian-jian Luo, Ya-chao Fan, Ji-tu Lv, et al. Deep reinforcement learning from error-related potentials via an eeg-based brain-computer interface. In *2018 IEEE international conference on bioinformatics and biomedicine (BIBM)*, pages 697–701. IEEE, 2018.

[15] Alisa Berger, Fabian Horst, Sophia Müller, Fabian Steinberg, and Michael Doppelmayr. Current state and future prospects of eeg and fnirs in robot-assisted gait rehabilitation: a brief review. *Frontiers in human neuroscience*, 13:172, 2019.

[16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.