Human VS Deep Reinforcement Learning Agent in Visual Input Task

Xueqing Liu[1] Yannan Chen[2] and Xinhui Li[3]

***Abstract*— Artificial neural networks such as convolutional neural networks, though developed independently from biological neural networks, have been compared the structure and function similarities with human brains. However, little research is about the comparison of the reinforcement learning mechanism and performance between an artificial agent and a human player in a game task. In this study, we use visual input to train a deep reinforcement learning agent to perform a real-time error process task aiming to match an individual's performance in a similar task. It is the first time that an agent is trained with visual observation instead of vector observation in a 3D balancing task. We also visualize the "attention" of the artificial agent for subsequent comparisons with eye-tracking and EEG data collected from human subjects in our previous study.**

I. INTRODUCTION

Artificial Intelligence (AI) has demonstrated its ability and potential in multiple applications such as image processing and decision making. Compared with traditional machine learning algorithms, deep artificial neural network has been proved its superiority in both tasks. Although convolutional neural networks (CNN) and deep reinforcement learning algorithms are developed independently of the detailed neural mechanisms, the success and opacity of these artificial neural networks inspire the exploration of computational neural science to find the connection between a biological neural network and an artificial neural network. U. Guclu and M. A. van Gerven [1], [2] built models to predict the blood oxygen level-dependent responses to natural images and movie stimuli with the layers of CNN. In line with the work by Eickenberg et al. [3], their findings showed that the hierarchical feature representation of the CNN is in agreement with the representation organization in both the dorsal and ventral visual pathways of the human brain. Electroencephalography (EEG), magnetoencephalography (MEG) and functional Magnetic Resonance Imaging (fMRI) are functional neuroimaging modalities to study the dynamics of neural activities and interactions. To integrate the information from these measurements, several studies used representational similarity analysis (RSA) [4] to compare the representational similarity across these modalities and computational models. For example, Cichy et al. [5] showed the spatial and temporal hierarchical correspondence between the human brain and CNN in visual object categorization after comparing results of MEG and fMRI with those of CNN. Subsequently, they further investigated the representation in scene recognition between MEG and CNN, specifically indicating the temporal dynamics in the MEG is consistent with multi-stage scene processing in the CNN [6]. Recently, Kheradpisheh et al. [7] studied the viewpoint invariance in object recognition via a comparison of the representational similarity between human behavioral measures and neural network models. Correlations between different layers of a convolutional neural network to corresponding visual cortex areas are presented by all the of the studies above, but there are very limited studies of the comparison of the reinforcement learning mechanism between individuals and artificial deep reinforcement learning agents.

The study of human reinforcement leaning mechanism has a long history. Modern brain imaging techniques such as EEG and fMRI add a few more pieces to the puzzle and provides a general picture [8]: "A mesencephalic dopamine system for reinforcement learning and a 'generic' error-processing system associated with the anterior cingulate cortex. The existence of the error-processing system has been inferred from the error-related negativity (ERN), a component of the event-related brain potential elicited when human participants commit errors in reaction-time tasks."

EEG as a widely used brain imaging technique has achieved great development. It is well-known for its high temporal resolution since it directly records the electrical activity generated by neurons. With the assist of physiological information such as pupil radius, eye gaze, electrocardiogram, electrodermal activity and respiration, we can investigate further into the dynamic error-processing system of human brain.

In our previous study [9], human subjects get involved with a brain computer interaction research which their EEG data are recorded during a boundary avoidance task (BAT). The BAT is an aerial navigation task implemented in the Unity environment which creates cognitive conditions that escalate arousal and quickly leads to task failure if missing or crashing into the boundary. The information extracted from the EEG is used to generate a real-time neural feedback signal that dynamically adjusts the arousal state of a subject.

Our ultimate plan is to train a reinforcement learning agent with the same task layout and compare its performance with that of the human subject. This requires us to master the following tasks:

1) Learn to design and modify a game in the Unity environment

*This work was not supported by any organizations

[1]Xueqing is PhD candidate with the Department of Biomedical Engineering, Columbia University, New York, NY 10027, USA xl2556 at columbia.edu

[2]Yannan Chen is with the Department of Biomedical Engineering, Columbia University, New York, NY 10027, USA yc3338 at columbia.edu

[3]Xinhui Li is with the Department of Biomedical Engineering, Columbia University, New York, NY 10027, USA xl2679 at columbia.edu

2) Connect machine learning tools and resources in the Unity platform, such as Unity machine learning agent toolkit (ml-agent), with high-level neural networks APIs such as Keras, TensorFlow etc.
3) Implement deep reinforcement learning algorithms to train an artificial agent
4) Compare the performance of the artificial agent with that of the human player, and investigate the error process mechanisms of human beings

In this paper, due to the limit of time and level of relevance to the course, we focus on the second, the third and part of the fourth task of the project.

II. Method

A. Task Overview and Modification

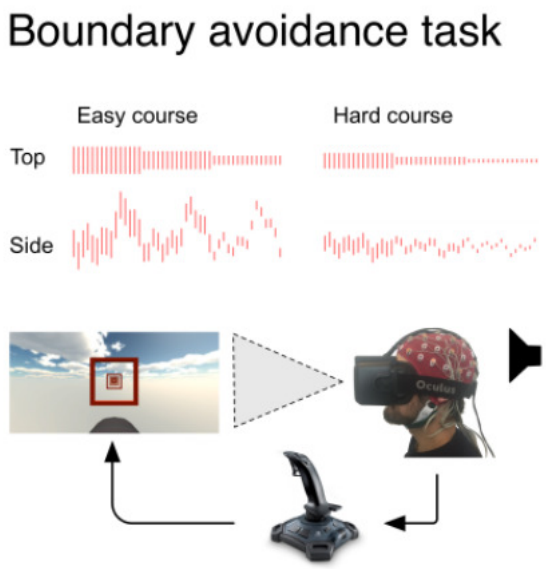

Fig. 1. Boundary avoidance task experiment setup: study subjects alternately guided a virtual aircraft through an easy or hard course of red rectangular boundaries. Both courses were a maximum of 90s long and increased in difficulty over time as ring sizes decreased. Missing or crashing into the boundary ended the flight trial immediately.

Twenty healthy adults perform a BAT in a virtual reality environment, where they navigate a plane through courses of red rectangular boundaries. Flight attempts are alternately performed in an easy and hard course (Fig.1, top). Every course is a maximum of 90 seconds long but end immediately whenever the pilot misses or crashes into the boundary. The task difficulty increases as the size of the rings decreased every 30 seconds.

After trials for one week, we suspended the attempt of connecting the BAT game to Unity ml-agent toolbox. Although we are able to modify the game to make it run with the ml-agent, it doesn't connect to several modules such as "brain" and "agent" of the ml-agent. The reason lies in that our previous task is written in JavaScript and Unity has stopped supporting JavaScript and transformed to C# completely. To make it compatible with unity ml-agent, the only solution is to rewrite the game completely in C# following the game structure in Unity ml-agent toolkit. Because of the limit of time of class project, we choose to use an available Unity task to continue with the rest of the project and turn back to rewriting the game subsequently.

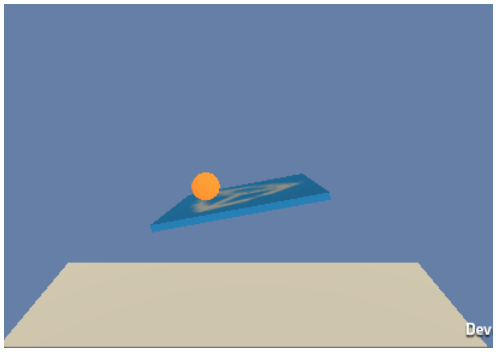

Fig. 2. Temporary substitute task: learn to balance a ball on a plate

The temporary substitute task is a 3D ball balance task built in Unity environment. Despite this task display looks very different from the BAT, it has one of the most important features that we are studying in the BAT – It's a continuous task involves over compensation and error process.

3D ball balance task is usually trained with vector input: ball position, plate position and ball speed. This training process is very different from how a human subject plays this game – based on visual input only. To make the reinforcement learning agent performs in a similar way as a human being does, we modified the Unity task to provide visual observation instead. We set up the Unity environment to adjust the ball, plate and camera position of the original game to acquire the optimal view of the task.

B. Task Environment Setup

The Unity ml-agent toolbox only offers Proximal Policy Optimization (PPO) algorithm as an example to train a reinforcement learning agent in TensorFlow. To apply more algorithms and take advantage of neural network visualization toolbox in Keras in future research, instead of using the PPO algorithm provided, we refer to OpenAI's gym, generate a gym-like environment based on the Unity 3D ball balance task prototype, and apply reinforcement learning algorithms in Keras in the new environment.

C. Reinforcement Learning Agents

We implement Deep Q Network (DQN) and Deep Deterministic Policy Gradients (DDPG) agents for both vector input and visual input in the experiment.

Vector input (Fig.3a) is of length 8, including 2 rotation elements for the plates, 3 relative position elements between the plate and the ball and 3 velocity elements of the ball. Visual input at each time step is a preprocessed version of current frame. Preprocessing consisted of gray-scaling, cropping the game space to an 80×80 square size and normalizing the RGB color values to [0,1]. Experiments on vector input are used to test the performance of our DQN and DDPG agents.

Generally, we experiment with visual input with several different models, including:

- Use one frame as state input to convolutional layers (Fig.3b)
- Use stacked multiple frames as state input to convolutional layers (Fig.3c)
- Use multiple frames as state input to convolutional layers followed by long short-term memory (LSTM) units (Fig.3d)

1) Deep Q-learning [10]: The full DQN algorithm for is presented in Algorithm 1.

Algorithm 1: deep Q-learning with experience replay

Initialize replay memory D
Initialize action-value function Q with random weights θ
Initialize target action-value function Q' with weights $\theta' = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1, T **do**
    Select action $a_t = \underset{a}{\text{argmax}}\, Q(\phi(s_t), a;\ \theta)$ according to the policy and exploration noise
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_i = \begin{cases} r_j, if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max_{a'} Q'(\phi_{j+1}, a';\ \theta'), otherwise \end{cases}$
    Perform a gradient descent step on $\left(y_j - \left(\phi_j, a_j;\ \theta\right)\right)^2$ with respect to θ
    Update the target networks $Q' = Q$
  End For
End For

Our agent is given the vector input and the visual input respectively. It interacts with an environment through a sequence of observations, actions and rewards and aims to select actions that maximizes cumulative future rewards, i.e. survived frames in this task. A deep neural networks are used to approximate the optimal action-value (also known as Q) function as in Equation (1).

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a, \pi] \quad (1)$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation s and taking an action a.

In order to minimize the instability when a neural network is used to represent the Q function, we use a novel Q-learning with experience replay. We use an approximate value function $Q(s,a;\theta_i)$ in which θ_i are the parameters of the Q network at iteration i. To perform experience replay we store the agent's experience (s_t, a_t, r_t, s_{t+1}) at each time step t in a memory dataset D. The Q-learning update is applied on samples of experience and uses the loss function as in Equation (2) at iteration i.

$$L_i(\theta_i) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \quad (2)$$

For the vector input, Q network is a sequential neural network model with 3 dense layers (Fig.3a). For the visual input, Q network model contains 3 convolutional layers, followed by flatten layers and dense layers (Fig.3b,c). Another LSTM network (Fig3.d) is also implemented to train the DQN agent with visual input.

2) Deep Deterministic Policy Gradients [11]: The full DDPG algorithm is presented in Algorithm 2.

Algorithm 2: deep deterministic policy gradients

Initialize replay memory D
Initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s, a|\theta^\mu)$ with random weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights
$$\theta^{Q'} = \theta^Q, \theta^{\mu'} = \theta^\mu$$
For episode = 1, M **do**
  Initialize a random process N for action exploration
  Receive initial observation state s_1
  For t = 1, T **do**
    Select action $a_t = \mu(s, a|\theta^\mu) + N_t$ according to the policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from D
    Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}\right)$
    Update critic by minimizing the loss:
    $$L = \frac{1}{N}\sum_i \left(y_j - Q(s_i, a_i;\ \theta^Q)\right)^2$$
    Update the actor policy using the sampled policy gradient:
    $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)}\, \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
    Update the target networks:
    $$\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}, \theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
  End For
End For

Considering the continuous action space in our task, we implement DDPG algorithm, an actor-critic approach using neural network function approximators based on the deterministic policy gradients (DPG) algorithm.

The DPG algorithm uses a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s,a)$ is learned using the Bellman equation. The actor is updated by the expected return from the start distribution J with respect to the actor parameters:

$$\nabla_{\theta^\mu} J = \mathbb{E}[\nabla_a Q(s, a|\theta^Q)\ |_{s=s_t, a=\mu(s_t)}\ \nabla_{\theta^\mu}\mu(s|\theta^\mu)\ |_{s=s_t}] \quad (3)$$

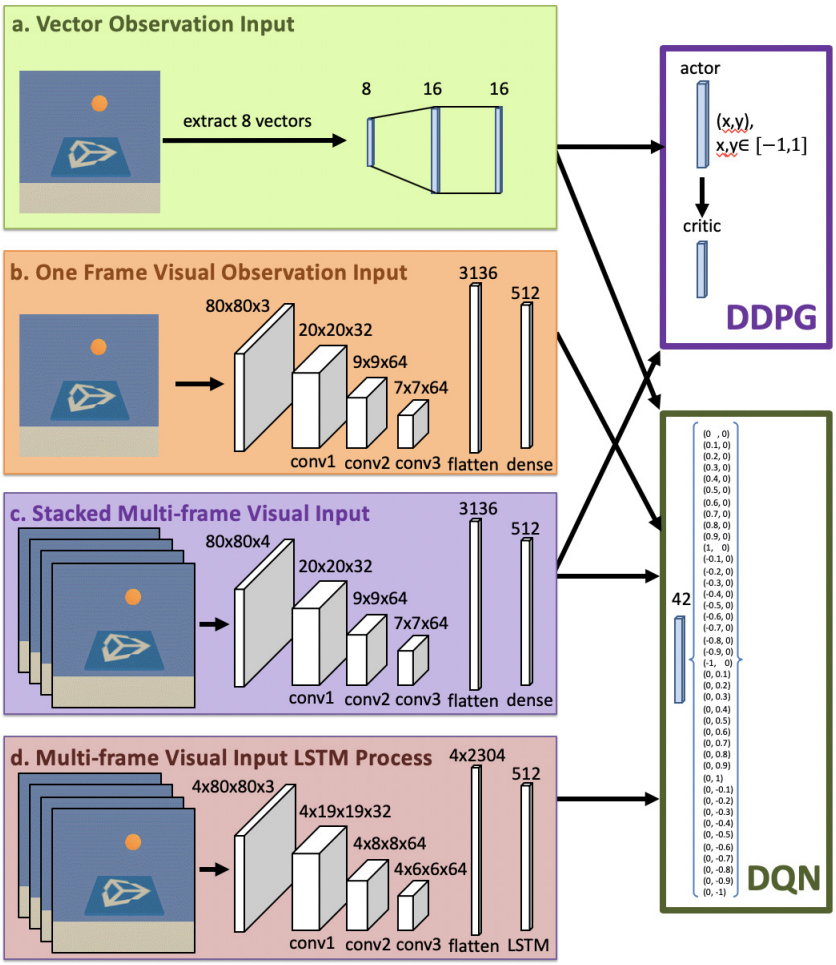

Fig. 3. Algorithm Frameworks

In order to overcome the unstable issues, a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$, are used to compute the target values. The weights of these target networks are updated by slowly tracking the learned networks, which improves the stability of learning.

For the vector input, there are 4 dense layers in both actor and critic model. For the visual input, both actor and critic model have 3 convolutional layers followed by flatten and dense layers to process visual input as the network in DQN. We only experiment with stacked multi-frame input for visual input (Fig.3c).

All different combinations of the inputs and reinforcement learning methods we conduct experiments with are shown in Fig.3.

D. Visualization

Two neural network visualization methods from Keras visualization toolkit, saliency map and attention map, are applied to the promising convergent experimental designs with visual input. Important features in convolutional layers for the agent to make the action decision are shown in corresponding highlighted image pixels. These visualization maps help to explain how the agent selects a particular action given visual input.

1) Saliency Map: Saliency map computes the gradient of output category with respect to input image.

$$\frac{\partial Output}{\partial Input} \tag{4}$$

The output value changes with respect to a small change in the inputs. These gradients are used to highlight input regions that cause the most change in the output. Intuitively it should highlight salient image regions that most contribute towards the output.

2) Attention Map: Attention map generates an input image that maximizes the filter output activations. i.e., we compute

$$\frac{\partial ActivationMaximizationLoss}{\partial Input} \tag{5}$$

and the estimate is used to update the input. Activation maximization loss simply outputs small values for large filter activations. This allows us to understand a particular filter is activated by what kind of input patterns.

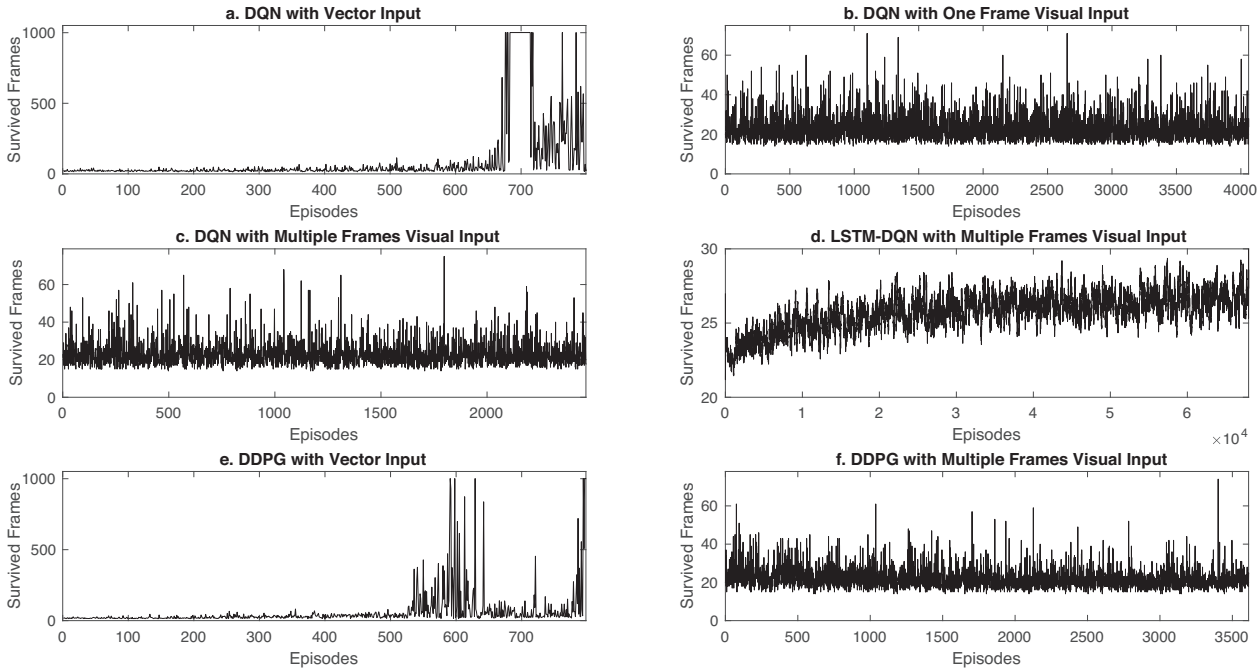

Fig. 4. Learning Curves for Different Settings

III. Results

A. Agent Performance

The performance of different input settings are shown in Fig.4. It is obvious to see that DQN with vector input converges fast (Fig.4a). It only takes about 680 episodes to converge to the best performance with 1000 survived frames per episode. Similar to the result of DQN with vector input, the number of survived frames of DDPG with vector input (Fig.4e) also increases over time and reaches the top of 1000 frames around the episode 600. However, both of them suffer from instability – there is a drop after the first top performance and then the number of survived frames begins going up again. The implementation of Q-learning with neural networks has been proved to be unstable in many cases. Both trained DQN and DDPG models perform well and achieve the top performance, i.e. 1000 survived frames, for every episode on the test set.

Besides, we experimented DQN with one frame visual input. The learning curve is shown in Fig.4b. The curve keeps flat and doesn't show any signs of learning over 4000 episodes, so we decided to stop the training process.

It is easy to explain why this one frame visual input architecture doesn't work in this 3D ball balance task. To balance a ball, we need to know not only the position, but also the moving speed and the moving direction of the ball. One frame input only provides the position information, so the information is not enough to train a well-performed reinforcement learning agent.

To overcome this problem of one frame visual input, we implemented DQN and DDPG with stacked multi-frame input. The results are shown in Fig.4c and Fig.4f respectively. However, the learning curves in both methods don't improve over time. The results indicate the difficulty of training the task with visual input only.

Considering the time-related property of the task, we tried to implement LSTM, which performs well on time series data, in our DQN model. Our LSTM-DQN model with multi-frame visual input eventually shows promising results. From Fig.4d, we find that the learning curve keeps increasing through training episodes. Although it doesn't converge to the maximum 1000 survival steps, it is able to survive almost 30 frames after training for above 60000 episodes and the agent shows obvious attempt to balance the ball.

B. Visualization

Two visualization methods, saliency map and attention map, are applied in the LSTM-DQN with multiple frames architecture to highlight the more heavily weighted regions used by the model for action selection. Shown in both Fig.5 and Fig.6, the images in the first row represent original clip frames, the images in the second row are corresponding to the visualization results of saliency map while the plots in the third row are the results from attention map. In all of the frames, the agent focuses on the upper center of the frames, where most important information is provided. The attention map shows some interesting features, such as the edge of the ball and the plate. It indicates that the agent is able to distinguish important objects inside each frame to make an action, which attempts to balance the ball.

The region that the agent focuses on varies little among different frames and different clips, but the strength of the attention varies in each frame. This is an interesting finding, which shows that different frames have different levels of importance in the agent's decision making process. We can compare this result with the eye-tracking data of human

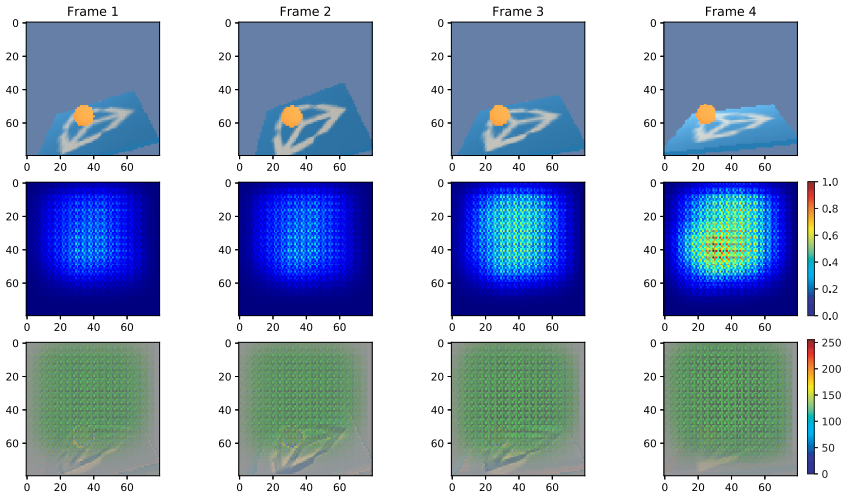

Fig. 5. Visualization Results Clip1

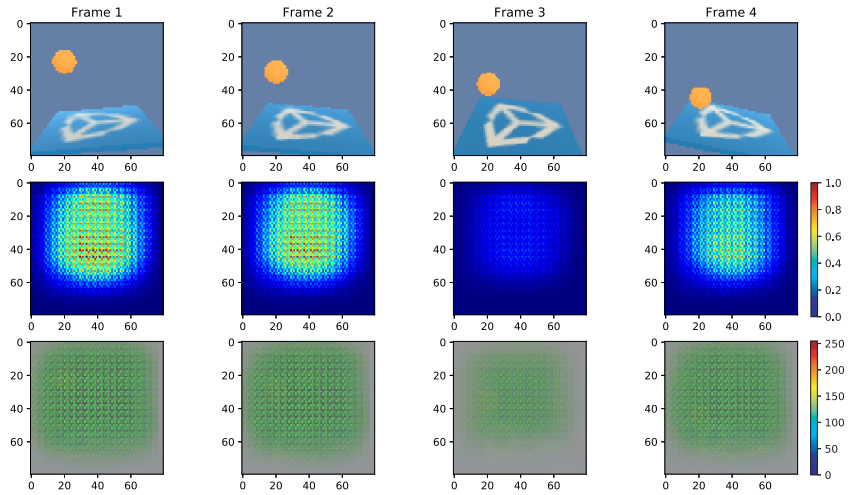

Fig. 6. Visualization Results Clip2

players, as pupil size is relevant to the attention level of human subjects.

IV. Discussion

We proposed to compare reinforcement learning agent with human performance on error process and over compensation tasks. In this report, we explore a pipeline to study this problem with reinforcement learning tools. Due to the script language conversion difficulty of the boundary avoidance task, we use a temporary substitute task with similar error process and over compensation mechanism to setup the pipeline for the study.

To make the task more similar to boundary avoidance task on human, we choose to train the model base on visual input only. We also explored DQN and DDPG's ability to solve this problem by training the agents with vector input. The learning curves of both DQN and DDPG algorithms converge very fast but also expose the unstable problem soon. The instability of Q-learning agent with neural networks is also reported by many researchers. To overcome this problem in the future, we need professional knowledge to optimize the algorithms and high-performance computing support to tune various parameters. In spite of the training instability, the trained DQN and DDPG models show perfect performance on the test set.

Visual input of different settings are tested: one frame, stacked multi-frame, multi-frame with LSTM. Our results demonstrate that it is easy for a reinforcement learning agent to play a real-time error process task such as balancing a ball given vector input. However, the key difficulty is to train the agent with only visual input. Most of the settings failed, only multi-frame with LSTM-DQN achieved limited success. There are several reasons for this:

1) Unlike the 2D environment in Atari games, Unity provides a 3D game environment. The agent needs to figure out not only the positions of the objects but also their velocities and moving directions, which has

always been a difficult problem in the computer vision field.
2) For a task like 3D ball balancing, the agent needs to be sensitive to the position and speed of the objects. However, due to the limit of computational load, we can't use high resolution images for training. This limits the agent's ability to extract information from the visual input.
3) The task by itself is difficult even for human beings with visual observation only. Most of us are able to balance a ball on a plate given a real plate and a real ball. However, that's under the condition we can touch the objects, and their position, speed and location will pass on to us through feelings. However, when we try to play this game on the screen based on visual input only, we often fail pretty badly even after a self-training process.

The visualization results show the potential of the method to compare with human eye tracking results. The intensity of the attention changes in different frames, which means some of the frames are of more importance in decision making. If we have a reinforcement learning agent who can play the game to a perfect score, we can consider that as a "ground truth" for the variation of attention. As attention can be reflected by a human being's pupil size, level of attention can be extracted from eye-tracking data.

For our project, since the ball and plate always stay in the same position, the agent learns a pattern of focusing on that location. In the boundary avoidance task, as the virtual airplane moves, the position of the rectangular boundary is also going to change on the screen. The agent will need to learn to find the location of the boundary in order to survive through the game or focus on the most important features in the game. A "perfect" reinforcement learning agent will learn to focus on the most important information on the frame and make a wise decision. This can also be compared with a human being's gaze location data collected with eye-tracking equipment. The onset of reinforcement learning agent's attention can also be analyzed together with event-related potential in EEG data.

Our future work will focus on trying to implement the boundary avoidance task with C# to make it compatible with Unity ml-agent toolkit, and compare the reinforcement learning results with our human subject data.

V. Project Contributions

1) Setup boundary avoidance task and modify 3D ball balance task.(Xueqing, Yannan, Xinhui)
2) Implement DQN with vector input. (Xueqing, Yannan)
3) Implement DQN with one frame visual input. (Xueqing, Yannan)
4) Implement DQN with stacked multi-frame input. (Xueqing, Yannan)
5) Implement DQN with multi-frame LSTM processed input. (Xueqing)
6) Implement DDPG with vector input. (Xueqing, Xinhui)
7) Implement DDPG with stacked Multi-frame input. (Xinhui)
8) Visualize DQN with multi-frame LSTM processed input. (Yannan, Xinhui)
9) Generate result plots. (Yannan, Xinhui)
10) Write report. (Xinhui, Xueqing)

References

[1] U. Güçlü and M. A. van Gerven, "Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream," *Journal of Neuroscience*, vol. 35, no. 27, pp. 10 005–10 014, 2015.

[2] U. *Güçlü* and M. A. van Gerven, "Increasingly complex representations of natural movies across the dorsal stream are shared between subjects," *NeuroImage*, vol. 145, pp. 329–336, 2017.

[3] M. Eickenberg, A. Gramfort, G. Varoquaux, and B. Thirion, "Seeing it all: Convolutional network layers map the function of the human visual system," *NeuroImage*, vol. 152, pp. 184–194, 2017.

[4] N. Kriegeskorte, M. Mur, and P. A. Bandettini, "Representational similarity analysis-connecting the branches of systems neuroscience," *Frontiers in systems neuroscience*, vol. 2, p. 4, 2008.

[5] R. M. Cichy, A. Khosla, D. Pantazis, A. Torralba, and A. Oliva, "Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence," *Scientific reports*, vol. 6, p. 27755, 2016.

[6] R. M. Cichy, A. Khosla, D. Pantazis, and A. Oliva, "Dynamics of scene representations in the human brain revealed by magnetoencephalography and deep neural networks," *Neuroimage*, vol. 153, pp. 346–358, 2017.

[7] S. R. Kheradpisheh, M. Ghodrati, M. Ganjtabesh, and T. Masquelier, "Deep networks can resemble human feed-forward vision in invariant object recognition," *Scientific reports*, vol. 6, p. 32672, 2016.

[8] C. B. Holroyd and M. G. Coles, "The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity." *Psychological review*, vol. 109, no. 4, p. 679, 2002.

[9] J. Faller, J. Cummings, S. Saproo, and P. Sajda, "Regulation of arousal via on-line neurofeedback improves human performance in a demanding sensory-motor task," 09 2018.

[10] 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.

[11] 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.