Human Priors and Deep Reinforcement Learning for Video Games

Sree Lakshmi Addepalli
sla410@nyu.edu

Sree Gowri Addepalli
sga297@nyu.edu

May 15, 2020

This paper analyzes on how having a prior knowledge is helpful for humans in playing video games and compares its game play with that of an Reinforcement Learning Agent. Are Humans better in solving complex video games than an RL trained agent? Does having prior knowledge about the world help humans make better decisions to solve an complex game than an RL agent? We consider the Flappy Bird Video Game, and conduct different experiments to get a quantitative aspect of how important having prior knowledge helps in the performance of humans. We modify the environment on various basis, some being masking visual information or important information needed for efficient game-play and provide a comparison of results between human and an RL agent performance. The source code can be found at: Github.com/Lakshmiaddepalli/DeepReinforcementLearning.

Keywords: *Reinforcement Learning; Human Understanding; Exploration; Priors; Duelling DQN; Convolutional Neural Networks; Semantics; Affordances; Cognitive Modelling; Flappy Bird;*

1 Introduction

According to Mnih, V., Kavukcuoglu, K., Silver [1], Deep Reinforcement Learning (RL) methods have performed extremely well on different video games like Atari etc, but there efficiency in comparison to Human performance is extremely bad. They need millions of action states to solve a simple Atari Game. There have been methods to improve the efficiency of RL Agents but the approach humans take to solve a video game is quite different from an RL Agent. Humans involve prior knowledge of the world, from semantics, affordances and its physics, to solve a complex video game whereas RL Agents build its knowledge from a blank slate to the experiences, it has encountered itself with.

Importance of prior knowledge helps humans solve problems has been looked at in Doshi-Velez, F. and Ghahramani [3]. Psychologists in the paper Carey, S. [4] have been studying the prior knowledge children draw up in learning about the world. Some researchers Kansky, K., Silver, T., Mely [5] Narasimhan, K., Barzilay, R [6] have looked into adding priors into RL agents via object representations or language grounding, but progress is yet to be made as the field is still developing understanding of kinds of prior knowledge humans employ.

We carried out experiments where the human candidate was given an video game to play without any instructions on how to play it and was only notified the terminal state of the game. Secondly, the games were modified removing the semantics and affordances of the game. Again we tried a human candidate to play the modified video game and trained an

RL agent on playing the game. Few of the questions we wanted to answer were:

1. How human beings make decisions in a playing a video game they have never played and do not have knowledge of any of the game rules.
2. Do computational models behave similarly to cognitive models while playing the video game?
3. How do the players decisions differ from the best resembling computational models?

We try to figure out answers to these questions via the experiments and makes some conclusions.

2 The Flappy Bird Game

Flappy Bird was a popular mobile game. In it, the player controls a bird and tries to fly between the pipes without hitting them. It was the hardness and at the same time simplicity of the game that made people go crazy over this game and become a success. The objective is to direct a flying bird, who moves continuously to the right, between sets of pipes. If the player touches the pipes, they lose and a reward of negative five is given. The bird has a single action that is to flap upwards each time that the player taps the screen; and if the screen is not tapped,the bird falls down because of gravity. Navigating between each pair of pipes earns the player a single point. This game is restricted to 30fps as the physics at higher and lower frame rates feels slightly off.
State Representation:

1. Player y position. 2. Players velocity. 3. Next pipe distance to player. 4. Next pipe top y position.

5. Next pipe bottom y position 6. Next next pipe distance to player. 7. Next next pipe top y position. 8. Next next pipe bottom y position.
Here, initially while training the RL agent,we found out that achieving a score of 10 or more was taking around 12 hrs of training on google colab Tesla T4(16G). Also during collecting the data from human players, many were not able to score above a score of 5. Hence human comparisons to RL agent was becoming difficult. As the Flappy bird is a continuous game and so we decided to compare the human and RL agents based on the performance of time taken to reach score of three, number of deaths to reach score of three and total episodes to reach it.
Here, our objective is to find out how human cognition applies while playing a Flappy Bird game using priors and how does it differentiate from training an RL agent on it and able to figure out differences.

3 Concepts

3.1 Reinforcement Learning

These two words have made too much buzz in last few years, a lot of research in ML have now shifted to actively explore the field of Reinforcement Learning. Alpha Go defeated the world's best player in Go using RL, various other platforms like open ai gym have been opened to test the limits of Reinforcement learning, to make computer learn the way we human beings learn things.
Reinforcement learning consist of a transition function T that takes the current state s of the problem and current action a and gives back the rewards r of that action. Our aim in general is to maximize that reward so as to achieve the desired task. For good long-term performance, not only immediate rewards but also future rewards must be taken into account. The total reward for one episode from time step t is

$$R_t = r_t + r_t + 1 + r_t + 2 + \ldots + r_n \tag{1}$$

. The future is uncertain and the further we go in the future, the more future predictions may diverge. Because of that, a discounted future reward is used:

$$R_t = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \ldots + \gamma^{n-t} * r_n = r_t + \gamma * R_{t+1} \tag{2}$$

An agent should choose the action which maximizes the discounted future reward.

3.2 Deep Q Networks

Details of most of the technologies described below can be found in these two papers: [7] Playing Atari with Deep Reinforcement Learning and [8] Rainbow: Combining Improvements in Deep Reinforcement Learning. Q-learning learns the action-value function Q(s, a): how good to take an action at a particular state. In Q-learning, we build a memory table Q[s, a] to store Q-values for all possible combinations of s and a. We find out the reward R (if any) and the new state s′. From the memory table, we determine the next action a′ to take which has the maximum Q(s′, a′). Q-learning is about creating the cheat sheet Q. We can take a single move a and see what reward R can we get. This creates a one-step look ahead. R + Q(s′, a′) becomes the target that we want Q(s, a) to be.

$$target = R(s, a, s') + \gamma max_{a'} Q_k(s', a') \tag{3}$$

As we keep playing, we maintain a running average for Q. The values will get better and with some tricks, the Q values will converge. However, if the combinations of states and actions are too large, the memory and the computation requirement for Q will be too high. To address that, we switch to a deep network Q (DQN) to approximate Q(s, a). The learning algorithm is called Deep Q-learning. With the new approach, we generalize the approximation of the Q-value function rather than remembering the solutions. In reinforcement learning, both the input and the target change constantly during the process and make training unstable. Hence the fundamental problem in training an RL algorithm is trying to learn a mapping f for a constantly changing input and output! To have a more stable input and output to train the network and there are two important concepts to learn:

Experience replay: For instance, we put the last million transitions (or video frames) into a buffer and sample a mini-batch of samples of size 32 from this buffer to train the deep network. This forms an input dataset which is stable enough for training. As we randomly sample from the replay buffer, the data is more independent of each other.

Target network: We create two deep networks $\theta-$ and θ. We use the first one to retrieve Q values while the second one includes all updates in the training. After say 100,000 updates, we synchronize - with . The purpose is to fix the Q-value targets temporarily so we don't have a moving target to chase. In addition, parameter changes do not impact - immediately and therefore even the input may not be 100 percentage independent, it will not incorrectly magnify its effect. This helps in reducing the shift in the θ values by the network. So, the loss function now becomes:

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

Experience replay has the largest performance improvement in DQN. Target network improvement is significant but not as critical as the replay. But it gets more important when the capacity of the network is small.

DQN Algorithm with Experienced Replay

```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ 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
     With probability $\varepsilon$ select a random action $a_t$
     otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
     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 $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
     Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
     network parameters $\theta$
     Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

where Φ preprocesses last 4 image frames to represent the state. To capture motion, we use four frames to represent a state.

4 Proposed Approach

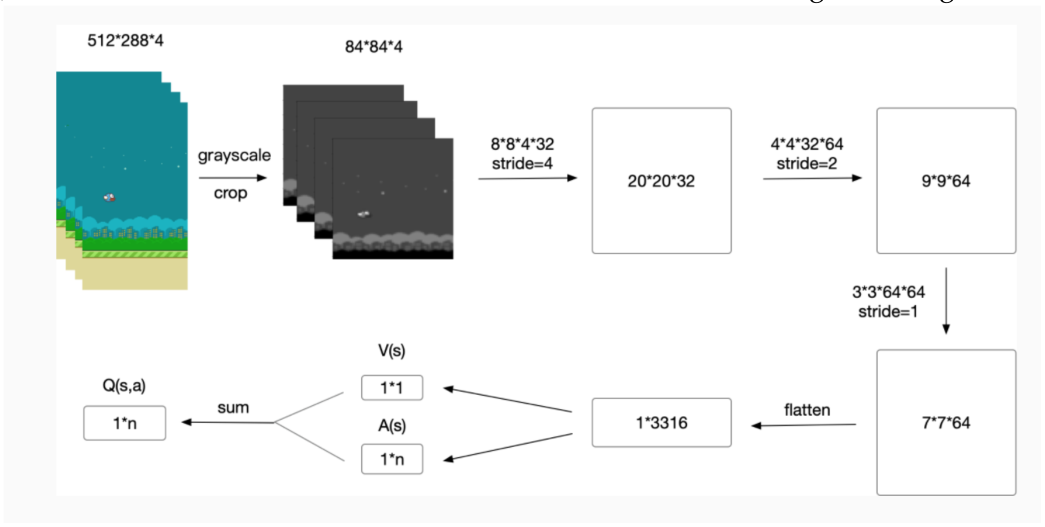
To investigate the aspects of visual information that enable humans to efficiently solve video games we had a local version of flappy bird game built that was similar to the PYGAME LEARNING ENVIRONMENT flappy bird. Similarly, for RL agent based comparison we have used the Flappy Bird environment in the PYGAME LEARNING ENVIRONMENT. The idea is to create different versions of the video game by rerendering various entities such as pipes, platforms using alternate textures. These textures were chosen to mask various forms of prior knowledge that are described in the experiments section. Note that all the games are exactly the same in their underlying structure and reward, thereby ensuring that the change in human performance (if any) is only due to masking of the priors.

We quantified human performance on each version of the game by recruiting 10 participants. Each participant was instructed to finish the game using the arrow keys as controls, but no information about the goals or the reward structure of the game was communicated. For each participant we recorded

1. Time taken to score 3 points
2. Total number of Deaths
3. Total number of Episodes

4.1 Training an RL agent

We trained RL agents on different versions having visual changes of the flappy bird. We here discuss the process of training and the architecture used. According to the paper [8] Rainbow: Combining Improvements in Deep Reinforcement Learning. they have specified various variants of DQN network and hence we went ahead and proceeded with **Dueling DQN** architecture. Here is the model Architecture used for training the RL agent:



The input to the network is an RGB image of size 512*288*4 frames which is cropped to a size of 84*84*4 and converted to grayscale. We use a combination of convolutional layer, batch normalization and ReLU non-linear activation function. All the convolutional layer use a kernel of 8,4,3 and stride of 4,2,1. The number of filters we use are 32; 64; 64; 512. The convolutional layers are followed by two fully connected layer. The first fully connected layer has output of 1 output node and the second fully connected layer has output values equal to number of actions. The final layer is the sum of the two layers minus the mean of the action scores.

```
----------------------------------------------------
        Layer (type)               Output Shape
====================================================
            Conv2d-1           [-1, 32, 20, 20]
       BatchNorm2d-2           [-1, 32, 20, 20]
            Conv2d-3             [-1, 64, 9, 9]
       BatchNorm2d-4             [-1, 64, 9, 9]
            Conv2d-5             [-1, 64, 7, 7]
       BatchNorm2d-6             [-1, 64, 7, 7]
            Linear-7                  [-1, 512]
            Linear-8                  [-1, 512]
            Linear-9                    [-1, 1]
           Linear-10                    [-1, 2]
====================================================
```

The vanilla DQN has the overestimate problem. As the max function will accumulate the noise when training. This leads to converging at suboptimal point.

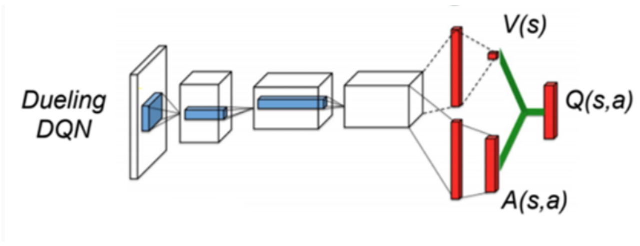
$$Q(s,a) = r + \gamma * max_{a'}[Q(s‘, a‘)] \tag{4}$$

Dueling DQN is solution for this problem. It has two estimator, one estimates the score of current state, another estimates the action score.

$$Q(s,a) = r + \gamma * max_{a'}[A(s‘, a‘) + V(s‘)] \tag{5}$$

In order to distinguish the score of the actions, the return the Q-value will minus the mean action score:

$$x = val + adv - adv.mean(1, keepdim = True) \tag{6}$$



4.2 Techniques Used

1. Image processing: The Images were grayscaled and cropped into size of 84 * 84.

2. Stack frames: The last 4 frames were used as an input. This should help the agent to know the change of environment.

3. Extra Fully Connected before last layer: Added a Fully Connected layer between the image features and the Fully Connected layer to calculate Q-Value.

4. Frame Skipping: Frame-skipping means agent sees and selects actions on every k frames instead of every frame, the last action is repeated on skipped frames. This method will accelerate the training procedure. In this project, I use frame skipping=2, as the more the frame skipping is, the more the bird is likely to hit the pipe. And this method did help the agent to converge faster.

5. Prioritized Experience Replay: According to the paper [9] Prioritized Experience Replay, DQN samples transitions from the replay buffer uniformly. However, we should pay less attention to samples that is already close to the target. We should sample transitions that have a large target gap:

$$|r + \gamma * max_{a'} * Q(s`, a`, \theta^-) - Q(s, a, theta)| \tag{7}$$

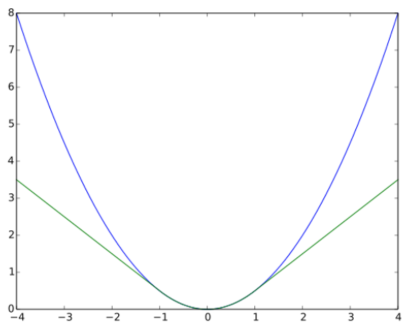
Therefore, we can select transitions from the buffer

1. Based on the error value above (pick transitions with higher error more frequently)

2. Rank them according to the error value and select them by rank (pick the one with higher rank more often).

6. Loss Function: Used Huber loss (green curve) where the loss is quadratic for small values of a, and linear for large values. The introduction of Huber loss allows less dramatic changes which often hurt RL.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \tag{8}$$



Green is the Huber loss and blue is the quadratic loss (Wikipedia)

7. Optimisation: Trained with Adam optimiser

8. Hyperparameter Tuning:

9. Action Selection: The epsilon threshold is set according to the following values:
if LOG DECAY parameter is set true:
epsilon threshold = EPS_{start} + $(EPS_{start} - EPS_{end})$ * exp$(-1. *$ (steps$_{done}$ - OBSERVE)/

BATCH_SIZE = 32
EPS_START = 1
EPS_END = 0.005
FRAME_SKIP = 2
EPS_DECAY = 300000
LOG_DECAY = False
TARGET_UPDATE = 1000
PLOT_INTERVAL = 50
REPLAY_SIZE= 100000
OBSERVE = 20000
LR = 1e-6
USE_PRIORITY_REPLAY = True
N_ACTION = env.action_space.n

EPS_{DECAY})
else if LOG DECAY parameter is set false:
we have variants of epsilon threshold values set, one of which is
epsilon threshold= max(EPS_{start} - (EPS_{start} - EPS_{end})/EPS_{DECAY} *($steps_{done}$ - OBSERVE),EPS_{end})
And if the sample value is greater the threshold then we select the action with maximum value else we select an action randomly out of the available actions.

4.3 Human Player

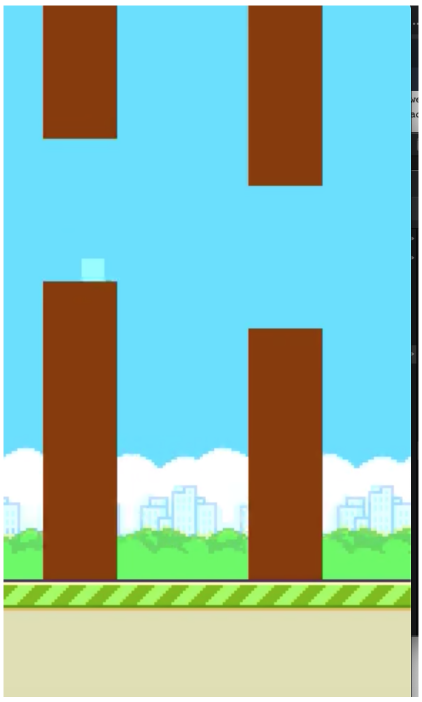
Similar version of the flappy bird game was used with the visual changes made and were given to 10 participants and the parameters of time, number of episodes and deaths were calculated. Hosting a PYGAME ENVIRONMENT was difficult as mostly no support was there and the game had to be converted to javascript based mode. Initially OPEN AI gym provided human playing mode and we found an alternate **repl.it** to host the game, but later we realised that we have to use PYGAME Environment and hence we could take readings from only 10 participants.

5 Experiments

We Experimented on 2 aspects of object priors i.e and compared it with the original game:

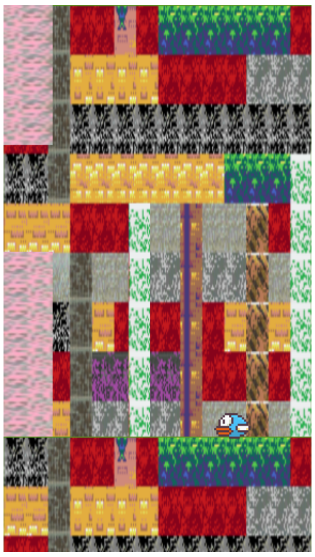
5.1 Semantics

To study the importance of prior knowledge about object semantics, we rendered the bird and pipes with blocks of uniform color so that the player doesnot understand what does it stand for. The visual appearance of objects conveys no information about their semantics. Knowledge of semantics enables humans to infer the latent reward structure of the game. We could have variants of this including reversed semantics.



5.2 Affordances

Till now, we manipulated objects in ways that made inferring the underlying reward structure of the game non-trivial. Even when the birds and the pipes were colored in uniform manner in games the connectivity pattern revealed space between the pipes of the game constitute free space. Here, the bird afford the actions of flying and pipes the action of avoiding to be hit, irrespective of their appearance. In next set of experiments, we manipulated the game to mask the affordances prior. One way to mask affordances is to fill free space with random textures, which are visually similar to textures used for pipes and background. Note that this game manipulation, objects and their semantics are observable.

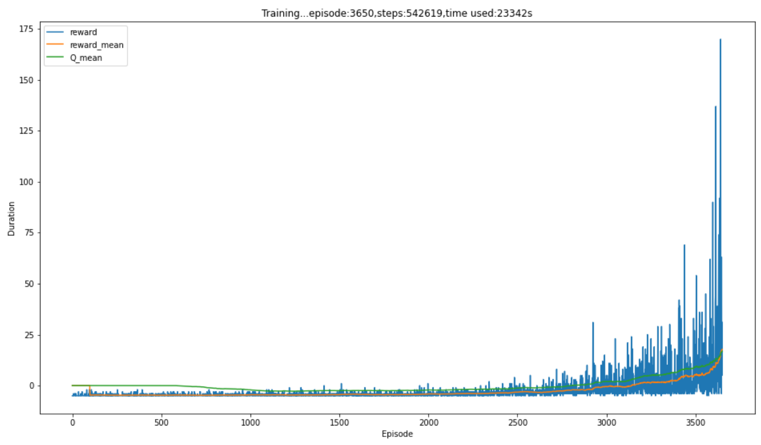


6 Results

Note: This is based on results of 10 participants, out of which 6 could not complete game due to reward structure and time consuming nature and rest 4 data has been averaged.

1. Original

To get score 3:

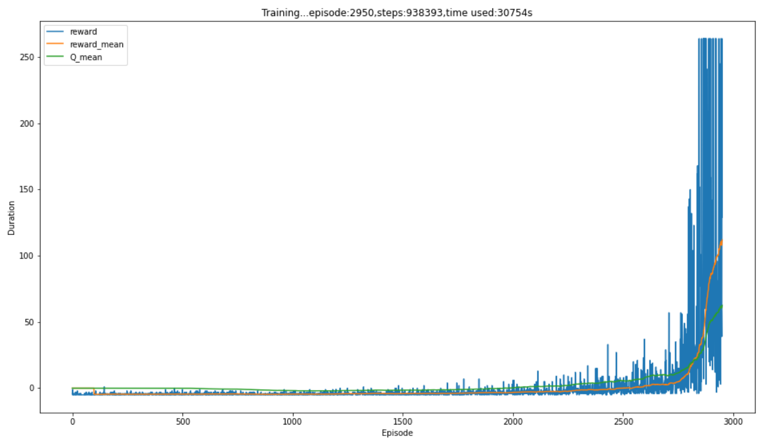


Training of DDQN on Original Flappy Bird Game

	Feature	Reinforcement Learning Agent	Human Player
1.	**TOTAL DEATHS**	2484	243
2.	**TOTAL TIME**	15885.3501 secs	1553.96 secs
3.	**TOTAL EPISODES**	2484	243
4.	**TOTAL ACTION COUNT**	194363	19014

2. No Semantics

To get score 3:

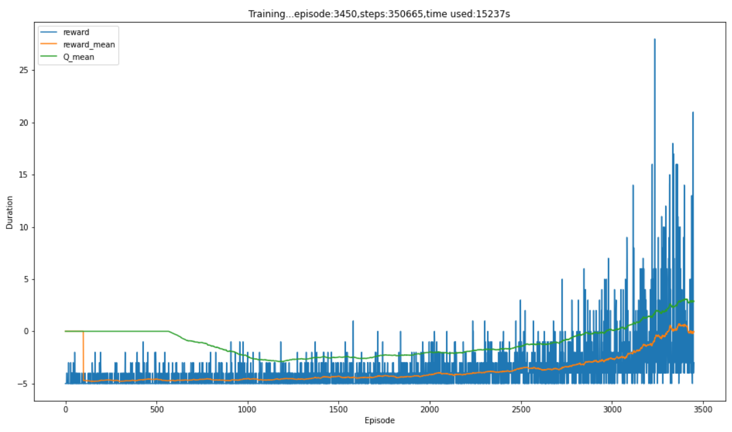


Training of DDQN on No Semantics Flappy Bird Game

	Feature	Reinforcement Learning Agent	Human Player
1.	TOTAL DEATHS	1703	289
2.	TOTAL TIME	17753.9193 secs	3012.84791 secs
3.	TOTAL EPISODES	1703	289
4.	TOTAL ACTION COUNT	140039	23765

3. Affordances

To get score 3:



Training of DDQN on No Affordances Flappy Bird Game

	Feature	Reinforcement Learning Agent	Human Player
1.	TOTAL DEATHS	2499	312
2.	TOTAL TIME	11036.88 secs	3377.95 secs
3.	TOTAL EPISODES	2499	312
4.	TOTAL ACTION COUNT	196648	24552

7 Discussion and Conclusions

1. Humans have a great deal of **prior knowledge** about the world, before making efficient decision making.

2. We find that **removal of some prior knowledge like semantics and affordances causes a degradation in the speed** with which human players solve the game.

3. The results indicate that priors such as the importance of **objects knowledge and visual consistency** are important for game-play for humans.

4. **RL agents takes very high action steps** to solve a simple game than a human player.

5. RL agents attack each problem **tabula rasa**, whereas **humans** come in with a **wealth of prior knowledge** about the world, from **physics to semantics to affordances**.

6. The original game took 25.9 minutes and 19014 action steps for the humans to reach a score of 3, whereas as we remove the semantics and affordances the game play takes nearly

twice the time and more action steps.The games with **no priors is clearly much harder for humans**, likely because it is now more difficult to guess the game structure and goal, as well as to spot obstacles.

7. Unlike humans, **RL did not show much difference between the games**, taking nearly same action inputs to solve each one. This is not surprising. Since the RL agent did not have any prior knowledge about the world, both these games carried roughly the same amount of information from the perspective of the agent.

8. This result provides further evidence that in the absence of semantics, humans are **unable to infer the reward structure and consequently significantly increase their exploration.**

9. Due to difficulty in completing this game, we noticed a **high dropout of human participants before they finished the game.**

10. While improvements in the performance of RL agents with better algorithms and better computational resources is inevitable, our results make a **strong case for developing algorithms that incorporate prior knowledge as a way to improve the performance of artificial agents.**

11. How Humans acquire priors?

11.a Studies in developmental psychology suggest that human infants as young as 2 months old **possess a primitive notion of objects** and expect them to move as connected and bounded wholes that allows them to perceive object boundaries and therefore possibly distinguish them from the background.

11.b At this stage, infants do not reason about object categories. By the **age of 3-5 months, infants start exhibiting categorization behavior based on similarity and familiarity**.

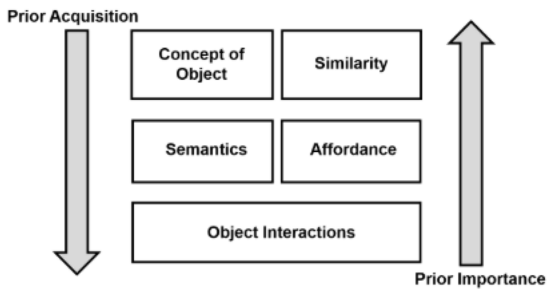
11.c The ability to recognize **individual objects rapidly and accurately emerges comparatively late in development usually by the time babies are 18-24 months old**.

11.d Similarly, while **young infants exhibit some knowledge about affordances early during development**, the ability to distinguish a walkable step from a cliff emerges only when they are 18 months old.

11.e These results in infant development suggest that **starting with a primitive notion of objects, infants gradually learn about visual similarity and eventually about object semantics and affordances.**

11.f It is quite interesting to note that the order in which infants increase their knowledge matches the importance of different object priors such as the **existence of objects as subgoals for exploration, visual similarity, object semantics, and affordances.** Based on these results, we suggest a possible taxonomy and ranking of object priors

11.g **In Figure** below We put 'object interaction' at the bottom as in our problem, knowledge about how to interact with specific objects can be only learned once recognition is performed.



Taxonomy of object priors

12. Finally, **RL agent was unaffected by the removal of semantics, the concept of objects, as well as affordances there is no significant difference between the RL agent on these games when compared to the performance on the original game.** This suggests that the **drop in human performance in these game manipulations is not due to the change in visual complexity, but it is rather due to the masking of the various priors.**

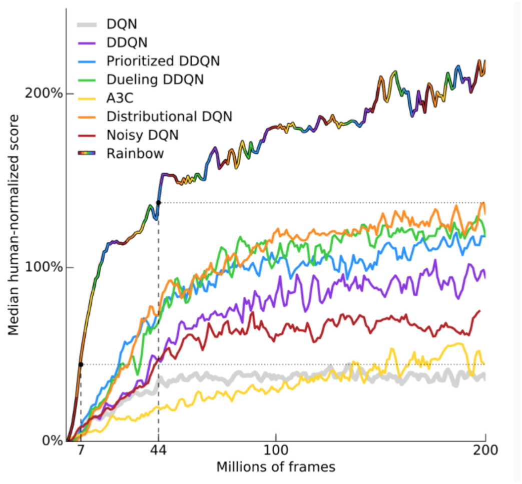
13. While incorporating prior knowledge in RL agents has many potential benefits, future work should also consider **challenges regarding under-constrained exploration in certain kinds of settings.**

14. In addition to developing better optimization methods, we believe that instead of always initializing learning from scratch, either **incorporating prior knowledge directly or constructing mechanisms for condensing experience into reusable knowledge (i.e., learning priors through continual learning) might be critical for building RL agents with human-like efficiency.**

8 Further Work

1. Could look into reverse semantics, masked identity of objects, masked visual similarity, interactions with object, changing gravity and other priors.

2. Could use Rainbow algorithm to train RL Agents as it gives best performance.



9 Issues Faced

1. Setting up the PYGAME ENVIRONMENT was a task in itself.

2. Hosting the pygame version was not possible as it needed to be converted to javascript/web based game.

3. Could not use Prince HPC for training as the video libraries were throwing errors.

4. Google Collab was disconnecting usually after 1 hour of training.

References

1. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., et al. Human-level control through deep reinforcement learning. Nature, 2015.
2. Mnih, V., Badia, A. P., Mirza, M., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In ICML, 2016.
3. Doshi-Velez, F. and Ghahramani, Z. A comparison of human and agent reinforcement learning in partially observable domains.
4. Carey, S. The origin ofconcepts. Oxford University Press, 2009
5. Kansky, K., Silver, T., D. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In International Conference on Machine Learning, pp. 1809–1818, 2017.
6. Narasimhan, K., Barzilay, R., and Jaakkola, T. Deep transfer in reinforcement learning by language grounding. arXiv preprint arXiv:1708.00133, 2017.
7. Playing Atari with Deep Reinforcement Learning
8. Rainbow: Combining Improvements in Deep Reinforcement Learning
9. Tom Schaul, John Quan, Ioannis Antonoglou, David Silver Prioritized Experience Replay
10. Using DDQN to play flappy Bird.
11. github.com/sourabhv/FlapPyBird
12. rach0012.github.io/humanRLwebsite/
13. towardsdatascience.com/the-complete-reinforcement-learning-dictionary
14. hardikbansal.github.io/FlappyDQNBlog/
15. medium.com/@jonathan$_hui/rl-dqn-deep-q-network-e207751f7ae4$
16. pygame-learning-environment.readthedocs.io/en/latest/user/games/flappybird.html