"You'll get an Ice-Cream!": Implementing motivation based reward strategies using Reinforcement Learning

Nolan Cardozo
s1034065
Radboud University
Nolan.Cardozo@student.ru.nl

Ajinkya Indulkar
s1034517
Radboud University
Ajinkya.Indulkar@student.ru.nl

Aghil Karadathodi Prasad
s1036614
Radboud University
Aghil.Karadathodiprasad@student.ru.nl

Puja Prakash
s1039329
Radboud University
Puja.Prakash@student.ru.nl

Abstract

According to the Incentive Theory of Motivation, a human is more likely to complete a task in a more optimized manner when it is provided an internal motivation compared to a human who hasn't been provided with any motivation. To exploit this theory, we have proposed and implemented size-based and time-based reward engineering strategies to learn the optimal policy. In this paper, we employ a basic DQN [3] model architecture to train an agent to learn the optimal policy for winning in the NeuroSmash environment. We found that giving optimal median reward scores gives the best performance. Further, we noticed that giving a large negative reward, forces the agent to lose as soon as the episode starts, so as to get a non-negative reward. Also it is seen that, a basic DQN agent employs a defensive strategy most of the time by going in circles to avoid falling over the edge and eventually causes the opponent agent to fall over. We report that a simple size-based reward strategy surpasses the performance of a basic DQN agent by employing a rather aggressive strategy compared to the basic DQN agent. Such motivation based reward strategies thus provides an insight to the internal motivation aspect of an agent when trained using Reinforcement Learning.

1 Introduction

In recent years, there have been several advancements in the field of reinforcement learning with several applications including self driving cars, automated robotics and playing computer games, to name a few. Reinforcement learning (hereafter referred as RL) has also been employed in various domains namely healthcare (generating automated reports), finance (automated trading), retail (recommendation engines), etc. To employ these agents directly in the real world is difficult due to the complexity of the real environment and the interaction between real agents (humans).

Hence, we first test these agents in a simulated computer-based environment so that we can test and compare various strategies and policies. Most of such simulated computer-based environments work on extrinsic rewards where an agent receives a reward from the environment based on the agent's action. These rewards help an agent understand the world (simulated environment) around it and learn the best action strategy to maximize rewards. This idea has led to state-of-the-art RL systems which are capable of even beating human experts in such environments [3].

Such tasks are not limited to rewards as exploiting the idea of incentives holds the potential of improving the performance of RL agents. By enabling them with a sense of internal motivation towards their actions, it can lead them to maximize their extrinsic rewards from an environment. To elaborate on this point, let us consider two children performing a standard task (for example, completing their school homework). The final reward to this task in a real world would be getting a grade for their work. Now, we provide only one child with an incentive of additional rewards (say, ice cream). According to the Incentive Theory of Motivation [4], the child with this positive incentive is more likely to perform the given task in a faster and optimized manner compared to the child without any incentive.

In this paper, we aim to explore this aspect of incentive in a RL scenario and in that sense we attempt to answer the question: *"Is it possible to create a simple motivation based reward strategy that can improve an RL agent's performance?"*. The primary goal is to understand and compare various reward engineering strategies and how they impact the training and productivity of agents. These strategies are divided into 2 broad categories, namely 1) Size-Based and 2) Time-Based.

In the size-based category, we try out different additional rewards (positive/negative) provided to an agent besides the environment's reward. The idea here is to demonstrate a simple reward engineering strategy that will aid faster convergence and help us find the most optimal policy. Alternatively, in the time based category, we devise strategies to incentivize the agent to win an episode in the minimum amount of time. We employ a Deep Q network [3] (hereafter referred as DQN) with no reward engineering strategies to act as a baseline for our experiments. We then run our experiments with our aforementioned strategies applied to the baseline DQN architecture.

2 Background

RL enables an agent to learn a complex task in a computer-simulated environment by taking a sequence of actions that maximizes its expected future rewards for the given task. In this paper, we use a model-free, off-policy RL algorithm called DQN which does not require the internal state of the environment for an agent to learn. Instead, the agent learns via raw pixel data which represents the current state of the environment.

2.1 Q-Learning

Q-learning is a traditional model-free reinforcement algorithm introduced by Watkins [6]. The learning starts with giving Q an arbitrary value. Further, based on the actions taken by the agent, the reward received and the new state reached, this Q value is updated. This means that the algorithm updates the value of Q in every iteration for the state, action, reward and next state pair. Although, the method worked well for simple tasks, the Q look-up table would be huge and hard to maintain for complex real world tasks. Also, most states are rarely reached and it would be extremely difficult for the Q-table to converge.

2.2 DQN

Deep learning has recently become highly popular for tasks such as computer vision and speech recognition. Deep learning involves the training of large amount of data using multiple layers of artificial neurons stacked together and learning through the process of back-propagation. By feeding sufficient data into deep neural networks, it is often possible to learn better representations than handcrafted features [1]. Hence, the issues with Q-learning are resolved by replacing the standard Q-function by a deep neural network . When we input this network with the state and action pair, it outputs a Q value(expected reward) and hence is called a Deep Q Network(DQN) . DQN allows the agent to explore the unstructured environment and acquire knowledge which makes it possible to imitate human like behaviour.

Alternatively, reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the Q function. This instability has several causes: the correlations present in the sequence of observations,the small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values(Q) and the target values [5]. These instabilities are addressed using experience replay that randomizes over the data, thereby removing correlations in the observation sequence and

smoothing over changes in the data distribution. Secondly , we use an iterative update that adjusts the Q towards target values that are only periodically updated, thereby reducing correlations with the target

On a higher level, the Deep Q Learning algorithm works as follows:

Step 1: Initialise the trained network, target network and empty experience replay buffer

Step 2: Interact with the environment and store [state, action, reward, next state] in an experience replay buffer

Step 3: Sample random batches of experiences from the replay buffer

Step 4: Calculate loss

Step 5: Update the Q network using stochastic gradient descent

Step 6: Copy weights from trained network to target network

Step 7: Repeat from step 2 until converged

3 Experiments

3.1 Environment

The environment called NeuroSmash has been built using the unity platform. We access the environment using the TCP/IP interface. The environment consists of two agents: Red and Blue. The red agent is controlled by the user and the blue is controlled by the environment. The agents are programmed to move forward with a velocity of 3.5 m/s*. It is important to note that the blue agent is artificial but not really intelligent. It merely updates its destination to the current position of the red agent plus some random variation (a surrounding circle with a radius of 1.75 m) and smoothly turns to that position every 0.5 s. When the agents come in close proximity with each other (a frontal sphere with 0.5 m radius),they get pushed away automatically with a speed of 3.5 m/s. The actions that the agents can perform is to turn right or left with an angular speed of 180 degrees/s.

This implies there are three potential discrete activities that the agent can make at each step: Turn nowhere, turn left and turn right. For convenience, there is another built-in action that is to turn left or right with uniform likelihood. An episode starts when the environment is reset and stops when one of the agents falls off the platform. At the end of each episode, the winning agent gets a reward of 10 points while the other gets none. It is to be noted that all times are simulation time. That is, 0.02 s per step when timescale is set to one. A screenshot of the environment can be seen in Figure 1.

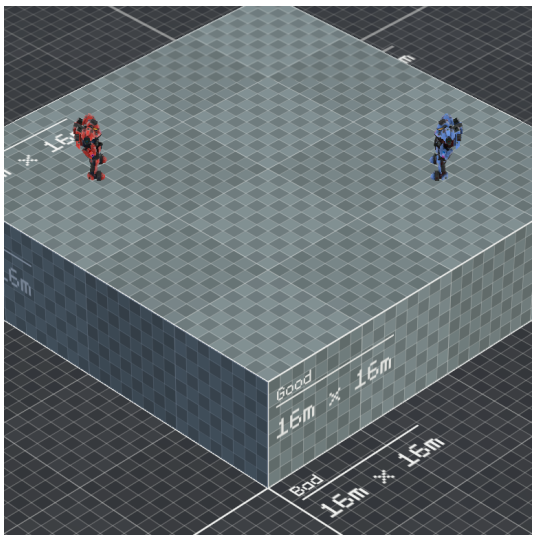

Figure 1: Screenshot of the Environment

3.2 Evaluation Metrics

In supervised learning, we can easily track the performance of the model during training by evaluating it with the training and validation sets. But with reinforcement learning, the evaluation process of an agent during training is challenging because evaluation metric is the total reward the agent collects in an episode or over a series of episodes. Hence, we demonstrate the performance of the agent through a test run of 100 episodes and calculate the win rate of the agent.

3.3 Training

We first train the agent using the REINFORCE Algorithm [7] and a Convolutional Neural Network [2] (hereafter referred as CNN) as a function approximator for the policy gradient function. The CNN consists of three 2D convolutional layers of channel size 16, 32 and 32 respectively. All convolutional layers have filter size of 5x5 with stride equal to 2. It is followed by a fully connected linear output layer with rectified non-linearity. The raw RGB input image size of 256x256x3 is transformed into a PyTorch tensor and normalized with a mean and standard deviation of 0.5. We train the agent for 500 episodes. We execute this step purely from a performance analysis standpoint.

We then train our agent using the DQN Algorithm and CNN as a function approximator for the Q Action-Value function. The model architecture of the CNN is similar to the previous model mentioned above. The CNN consists of three 2D convolutional layers of channel size 16, 32 and 32 respectively. All convolutional layers have filter size of 5x5 with stride equal to 2. It is followed by a fully connected linear output layer with rectified non-linearity. The raw RGB input image size of 256x256x3 is transformed into a PyTorch tensor and normalized with a mean and standard deviation of 0.5. We train the agent for 500 episodes due to resource constraints and observe a win rate of 59% over 100 episodes of test run. With these results, we consider the agent to be trained on our baseline DQN model and we proceed with our experiments with reward engineering strategies.

The environment only sends a non-zero reward at the end of an episode only if the agent wins. Thus, the size-based reward strategy focuses on providing a positive reward if the agent wins and a negative reward when the agent loses. The time-based reward strategy focuses on providing a positive reward to the agent if it wins within a threshold (T1) number of moves and a negative reward to the agent if it wins after a threshold (T2) number of moves. Both threshold values, T1 and T2, are manually selected by observing the duration of every episode while training the agent on our baseline DQN model. For the time-based reward strategy, we chose the standard rewards for agent based on the results of the size-based reward strategy. All the experiments after training our baseline DQN model are trained over 300 episodes and tested over 100 episodes due to resource constraints. See Figures 2 and 3 for visualization of the training phase which shows the duration of each episode and reward history of all episodes respectively.

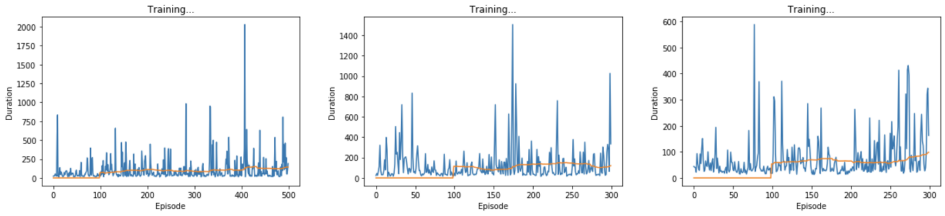

Figure 2: Training of Agent: (Left to Right) Baseline DQN, Size-based strategy 3, Time-based strategy 1

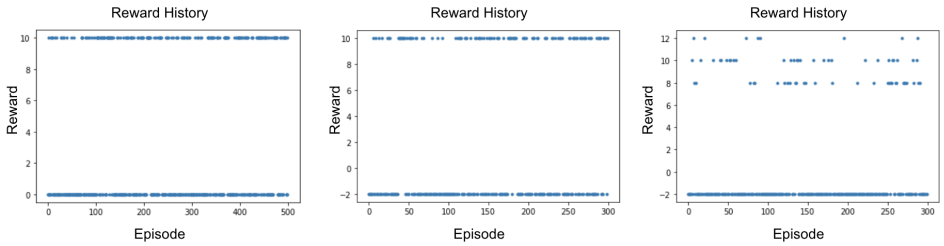

Figure 3: Reward History of Agent: (Left to Right) Baseline DQN, Size-based strategy 3, Time-based strategy 1

4 Results and Discussion

The results of the above experiments can be observed in Table 1. The REINFORCE agent performs poorly in this environment and a significant improvement is observed in our baseline DQN model. During the test run of our baseline agent, we observed that the optimal strategy was defensive in nature. The agent would go around in circles to save itself from falling off the edge which most times would cause the opponent agent to fall over the edge and our agent would win the episode.

Model	**Reward: Winning the episode**	**Reward: Within the episode**	**Reward: Losing the episode**	**Win Rate**
REINFORCE	10	0	0	8%
DQN	10	0	0	59%
Size-based Strategy 1	10	0	-10	22%
Size-based Strategy 2	10	0	-5	2%
Size-based Strategy 3	10	0	-2	61%
Time-based Strategy 1	(10, 10+2, 10-2)	0	-2	55%
Time-based Strategy 2	(10, 10+5, 10-5)	0	-2	12%
Time-based Strategy 3	(10, 10+5, 10-2)	0	-2	14%

Table 1: Performance Analysis of trained models (T1=30, T2=70). For time-based strategies, winning reward is ordered as: moves between T1 and T2, moves less than T1, moves more than T2

During the test run of our proposed size-based reward strategy, (specifically strategy 3) we observe a very interesting pattern. The agent moves around in circles at the same spot until the opponent moves towards our agent. Our agent then attempts to push the opponent off the edge. This is a slightly more aggressive strategy compared to our baseline DQN model. In other size-based strategies, we observed that the agent maximizes towards negative rewards and thus ends up finding the fastest way to fall over the edge. As observed in Table 1, the win rate is nearly 2% more than our baseline model. Although our current implementation does not support exact replication of test runs, in all our test runs, size-based strategy 3 has always surpassed our baseline DQN model. This demonstrates the fact that a simple motivation based reward system can be designed to improve an agent's learning and performance.

During the test run of our proposed time-based reward strategy, we observe in Table 1 that the win rate is not very promising as it is not able to surpass our baseline model. We believe that this is due to the limitations of experience replay in DQN learning. Although experience replay reduces bias during training, it lacks the ability to sample important state variables. Due to it's random sampling, most of the times redundant state variables are sampled which don't really help in converging the Q Action-Value function to an optimum solution. Thus when we attempt to provide a reward strategy

based on the number of moves an agent makes, the DQN model is unable to learn this temporal aspect.

5 Conclusion

In this paper, we are able to demonstrate a simple motivation based reward strategy which surpasses the performance of our baseline DQN model. Although the time-based reward strategy didn't provide positive results, the size-based reward strategy does show a lot of potential and supports in answering the question we posed in Section 1. Such motivation based reward strategies provide a path for RL agents which can emulate motivation as expressed by humans in the real world. Despite the fact that this paper only explores a manually engineered reward strategy, future work can include optimization of such strategies where the amount of additional positive/negative rewards can be dynamically altered based on the current state of the agent. Certain RL approaches tackle the limitations of experience replay. Our proposed time-based reward strategy can be implemented on these new approaches to test if our initial hypothesis can be proved right.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] 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.

[4] Sarah Mae Sincero. Incentive theory of motivation. `https://explorable.com/incentive-theory-of-motivation`, 2012. Retrieved Jan 14, 2020.

[5] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, A Rusu Andrei, and Veness Joel. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] CJCH Watkins. Learning form delayed rewards. *Ph. D. thesis, King's College, University of Cambridge*, 1989.

[7] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.