Solving Flappy bird using Deep Q Learning

Vardaan Kishore Kumar
Department of Computer Science
Stevens Institute Of Technology
Hoboken, NJ 07030
vkishore@stevens.edu

Rutul Thakkar
Department of Computer Science
Stevens Institute Of Technology
Hoboken, NJ 07030
rthakka3@stevens.edu

1 Introduction

Reinforcement learning has become a pivotal method of solving machine learning problems. It involves an agent learning through experiments by exploring and exploiting to gain maximum rewards. At each stage the agent receives an input on which it is expected to take an action. Based on the goodness of the action, the agent either gets a positive reward or a negative reward. The goal of the agent is to maximize rewards by choosing the best action for the selected set of inputs. The agent may consider short term rewards or long term rewards. The rewards can be given after each step or at the end of an episode. The advantage of reinforcement learning is that we don't need to manually program every single step of a process. Let us say we have a robot whose job is to collect garbage and place it in the dustbin, it would be very inefficient for us to program everything, even if we knew all the uncertainties, which is not practical in real life scenarios. Reinforcement learning helps us solve this problem by making the robot learn how to deal with uncertainties and by simulating many scenarios and making it learn from them. Video games are one of the areas where reinforcement learning has been implemented widely, given the wide state space and also uncertainties in games, they are as close as we can get to real life scenarios. Alpha Go is a prime example of how important and powerful reinforcement learning is. The agent was able to defeat the world champion of the game. Go is a very complex game and it is nearly impossible for someone to code it, it has a combination of steps greater than the atoms in the observable universe. The objective of this project is to implement a reinforcement learning algorithm called deep Q learning on Flappy Bird.
The a simple idea about the game can be understood from the figure 1:

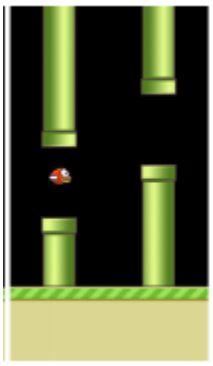

Figure 1: A hard step in the game for a normal user

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

2 Background

The breakthrough state of the art in deep reinforcement learning was presented by Google deep mind in their paper[3] on how they trained a Deep Q network on Atari games and surpassed human expert levels on 3 out of the 7 games they tested this on. The main advantage of their approach was that they only used game pixels as an input without explicitly telling anything about the game and its environment to the model, allowing it to learn this by its own. They also used experience replay to help de-correlate game by using a collection of past memories to help train the model in a better way.

3 Methodology

The most important step in reinforcement learning is to formulate a Markov Decision Process(MDP). In this particular problem, the agent can make two decisions , UP =1 or Nothing=0. Here we need store state information in the way of a stack, this state information is important as it helps provide temporal information that can be used to make better decisions in the future.
Below we define some of the most important terms related to reinforcement learning for better understanding.:

3.0.1 Episode

An episode is defined as a finite set of state, action and reward, most of the reinforcement learning algorithms use completing episodes for their work. An episode is considered terminated if it ends after a period of time.

3.0.2 Policy

A policy is an agents behavior, it is a mapping from states to actions. We can define a policy π is defined as follows: $\pi(a|s) = P[A = a|S = s]$

3.0.3 Reward

Reward is defined by the user, it is the total reward the algorithm gets at the end of an episode, this is defined by G in equation 1.

$$G_t = R_{t+1} + R_{t+2} + ...R_T \tag{1}$$

But there is an issue with the above equation, it give the same weight to the future and the immediate rewards, which is not always practical in real life scenarios, thus we introduce a discount factor, γ. This is why we include a discount factor:

- Uncertainty in the future, we can't have a perfect model.
- Avoids infinite returns in cyclical processes.
- It is mathematically convenient.

The discounted reward function can be formulated as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}..... \tag{2}$$

3.0.4 Value Function

A value of a state s under policy π is defined as the expected return when starting in state s and following policy π. This can be formulated as follows:

$$v_\pi = E_\pi[G_t|S_t = s] \tag{3}$$

In a similar way we also need to define the value of taking an action 'a' in state s under policy π. This is defined as a action-value function.This can be formulated as:

$$q_\pi = E_\pi[G_t|S_t = s, A_t = a] \tag{4}$$

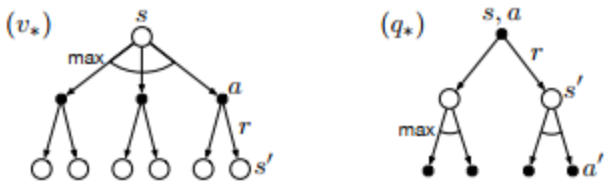

Figure 2: Graphs for finding optimal v,q

Our goal is to find the optimal value and action-value functions V* and q*. These optimal functions can be solved by using a recursive equation called the bellman equation and solving it.
The bellman equation for the value and action-value can be defined as follows.

$$v_\pi = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{5}$$

$$v*(s) = max_\pi v_\pi(s) q*(s, a) = max_\pi q_\pi(s, a) \tag{6}$$

These iterative functions can be expressed in the form of the graphs shown in figure 2

3.0.5 Model

A model predicts what the environment might do next. It is mainly comprised of two parts.

- Transitions P: predicts the probability of next state. $P^a_{SS'} = P[S' = s'|S = s, A = a]$
- Rewards R: predicts the next(immediate) reward. $R^a_S = E[R|S = s, A = a]$

In RL we have two primary functions.

- Prediction:
  This is to evaluate the future, how well will we do it we follow the current policy.
- Control:
  This is to find the best policy by optimizing for the future.

We typically need to solve prediction to solve control.
In our problem we use a model free control approach, we use it for the following reasons.:
The MDP for the model is unknown but we can learn from sampled experiences.
MDP is known but it is too big to use.

3.0.6 Types of Reinforcement Learning Algorithms

- On-Policy
- Off-Policy

On-Policy is where we are given a policy and we follow that, it is equivalent to "Learning on the job". We learn about policy π from experience sampled from π.

In off-policy learning we "Look over someone else shoulder". We learn about policy π from experience sampled from another policy μ.

Here we use an off-policy algorithm, so let us look at in greater detail.

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$.
- This is done while following a behaviour policy$\mu(a|s)$.

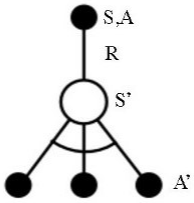

Figure 3: Backup diagram of Q-Learning

- We use experiences generated from old policies.
- Learn about optimal policy while following exploratory policy.
- Learn about multiple policies while following one policy

3.1 Q-Learning

Q-Learning is a type off-policy RL algorithm. We used the bellman equation optimize the following equation.

$$Q(S,A) < -Q(S+A) + \alpha(R + \gamma \underset{a'}{max}\, Q(S',A') - Q(S,A))(7)$$

Here the function Q(S',A') S',A' are the sate and actions of the future. The Q-learning algorithm takes the following inputs.
The current state 'S', it performs action 'A', gets reward 'R', moves to new state S' can we summarized in the backup diagram as shown in figure 3:
The pseudo code for Q learning is presented in figure 4.

1. Initialize Q-values ($Q(s, a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s, \cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s, a) := Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$

Figure 4: Pseudo code for Q-learning

In the normal approach we maintain a Q-table which gives us the reward for each state and the action we take, but this is not practical in complex spaces.

4 Deep Q Learning

In a Deep Q learning methodology we use a Deep Neural network that takes in a state and tries to approximate the best Q value for the sate. The difference is highlighted in figure 5.Recent breakthroughs in computer vision and speech recognition have relied on efficiently training deep neural networks on very large training sets. The most successful approaches are trained directly from the raw inputs, using lightweight updates based on stochastic gradient descent. By feeding sufficient data into deep neural networks, it is often possible to learn better representations than handcrafted features[2]. These successes motivate our approach to reinforcement learning. Our goal is to connect a reinforcement learning algorithm to a deep neural network which operates directly on RGB images and efficiently process training data by using stochastic gradient updates[3].

5 Experimental Design

In our experiment we are trying to solve a sate which has two possible action Up or Nothing. The rewards are of the following type:

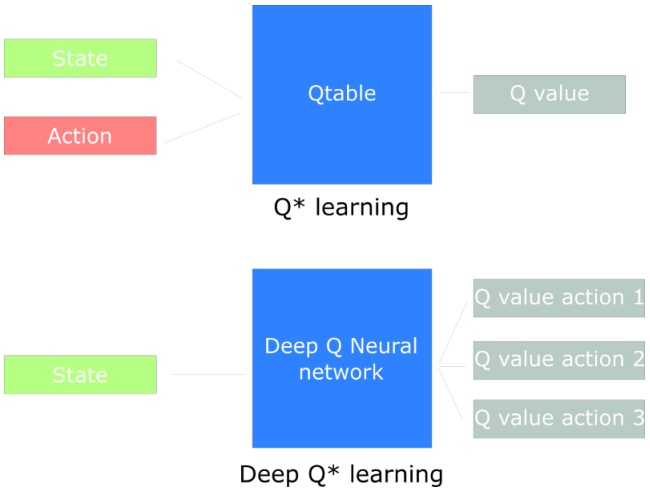

Figure 5: Difference between Q and Deep Q learning

- Reward for Staying Alive.
- Rreward for Dying.
- Reward for Crossing a Pipe

We try to minimize and optimize the loss function based on equation 7.
This loss 'L' can be defined as:

$$L = \sum_{s,a,r,s'} ((r + \gamma \underset{a'}{max} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta) \tag{8}$$

The gradient of the above loss function is defined as.

$$\nabla_{\theta_i} L_i(\theta_i) = \sum_{s,a,r,s'} [(r + \gamma \underset{a'}{max} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta)) \nabla_{\theta_i} Q(s, a; \theta_i)] \tag{9}$$

5.0.1 Pre-Processing

Before we can start the training process we need to do some preprocessing for faster convergence of the algorithm. We take screen shots of the flappy bird game and do the following processes before using them.:

- Resize the image to an 80x80 shape.
- Convert the image to GreyScale.
- Remove Sound Effects.

We take screenshots of each action perfomed in a contionous game, so in real time it looks like it doing multiple jumps but in the algorithm what happens is that it only looks at the current state and decides to jump or not, based on the previous images. We stack 4 images together, this is done to provide temporal information to the algorithm for better understanding of the state space and to take the best decision.
To extract the most information out of these stacked images we use a Convolutional Neural Network, which is state of the art in image processing and extracting features and spatial information. The CNN structure we use in based on the network used by deep mind in [3]. The network structure is

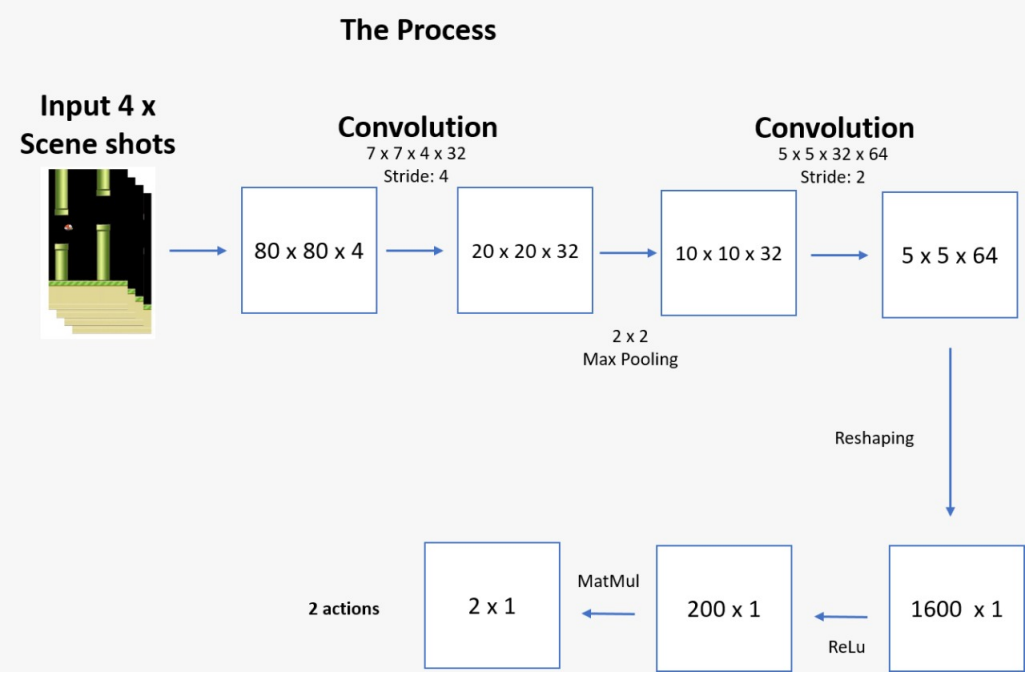

Figure 6: Network Structrure

shown in figure 6.:
During the first iteration the 4 images are the same as we are just starting out.
However there as issues if we use the above model, there is an issue of correlation, what this means is that the model is only focused on the previous images and learns inefficiently, to solve this we use a replay buffer, this is a tuple of past experiences in the form of queue, then from the queue we randomly sample to help train out Deep Neural Network, this can help de-correlate and make sure the model does not overfit to our current experience.
There is another stability issue with the network, in equation number 7 we try to move our prediction to a target at each step, but as the game goes on we are also moving and updating the target, this is very unstable, thus we use the concept of Frozen Q values from previous experiences as a target value.
We else take a greedy approach, to solve the exploration-exploration problem which means that if we only we just exploit a given path and do not explore newer states,we might end up at a local optimum, so here we at random try to break the sequence and based on a random parameter e, select between exploration and exploitation.
The rewards and penalties were set up as follows:

- Crossing a pipe = +1
- Surviving = +0.1
- Crash = -1

We observed for 100 timesteps before we start training, the replay memory is set to 25000, the minibatch size is set to 32 for sampling and updating the parameters of our neural network to prevent correlation and to implement Experience Replay.

6 Results

We trained the network one a GPU for 15 hours and the model was saved for future use.

Figure 7 shows the reward obtained at each iteration.
Figure 8 shows the max Loss value with respect to each iteration.

Initialize replay memory;
Assign random θ for the neural network;
repeat
    Begin a new Episode;
    Take a Screenshot of each action;
    repeat
        Pass the stacked screenshots to the CNN;
        Extract the values of action(a) from it according to the loss function;
        With probability ϵ choose a random action else;
        Take the action with highest Q value from the listed;
        Add this to the replay buffer for use later;
        Sample mini batches of size 32 to update DQN parameters;
        Update probabilityϵ;
        Based on if prediction network is close to target network, update the target network;
        update state, action, rewards;
        reflect this in the game screen;
    until *Until Death*;
    restart the game;
until *Until given no of iterations*;

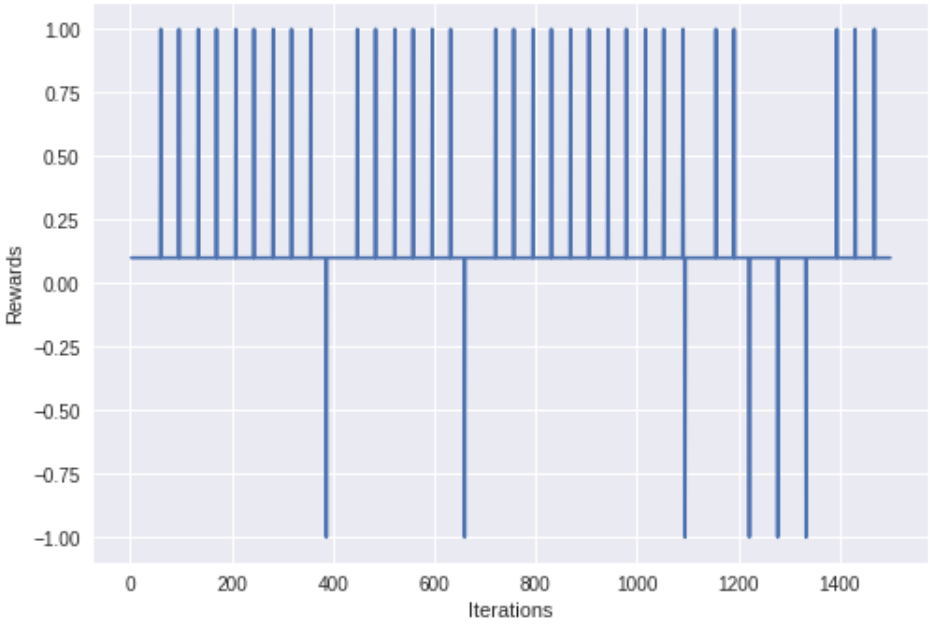

Figure 7: Reward per Iteration of trained network

As there is no particular error metric for understanding how well the model works, we just use the real time video of the agent playing to see how well it does, the same is included as a gif with the paper.

7 Conclusion and Future Work

In this project we try to solve the flappy bird game using Deep- Q Learning, the results were impressive as we can see from the learned model playing on it's own.
Future work in this area would include optimizing the model to help it train faster and also make it less data hungry. The work is inspried by the excellent lecture series [1] and the book [4]

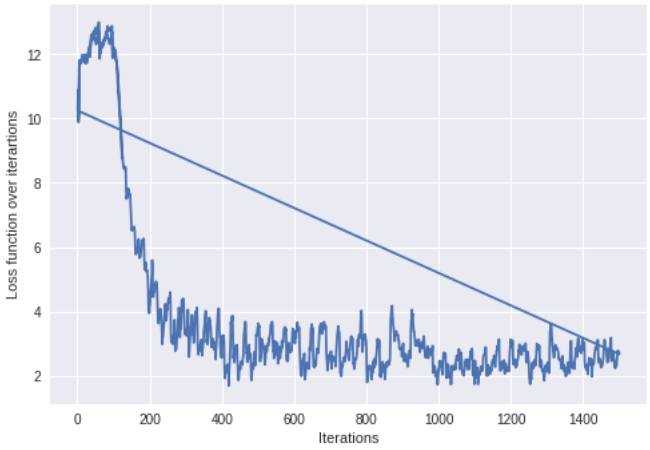

Figure 8: Loss after each iteration

References

[1] http://www0.cs.ucl.ac.uk/staff/d.silver/web/teaching.html.

[2] 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.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.