

UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Deep reinforcement learning for Flappy Bird using TensorFlowJS

Author:
Matvii KOVTUN

Supervisor:
Mykhailo IVANKIV

*A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science*

in the

Department of Computer Sciences
Faculty of Applied Sciences



APPLIED
SCIENCES
FACULTY ●

Lviv 2019

Declaration of Authorship

I, Matvii KOVTUN, declare that this thesis titled, “Deep reinforcement learning for Flappy Bird using TensorFlowJS” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Deep reinforcement learning for Flappy Bird using TensorFlowJS

by Matvii KOVTUN

Abstract

In this paper, I will cover a specific topic of Deep Reinforcement Learning which will be performed in the browser. I will provide an architectural overview of a system, describe a process of learning, show a deployment process. There are two key parts in this work - environment of an execution which is a browser and a learning approach which is Deep Reinforcement Learning. Motivation for this was a rapid development in both of these technologies deep learning and web.

Demonstration of my work can be found here:

[AI playing Flappy Bird](#)

Code can be found here:

[Github repository](#)

Contents

Declaration of Authorship	ii
Abstract	iii
1 Introduction	1
1.1 Motivation	1
1.2 Frontend computations	1
1.3 Reinforcement Learning	2
1.4 Goal	2
2 Background Information	3
2.1 Artificial Neural Network	3
2.2 Reinforcement Learning	3
2.3 Frameworks	5
2.4 Flappy bird	5
3 Related Works	8
3.1 TensorflowJS	8
3.2 Deep Reinforcement Learning	9
3.3 DeepLearningFlappyBird by yenchelin	9
3.4 FlappyBirdRL by SarvagyaVaish	10
4 Solution overview	11
4.1 Environment	11
4.2 Model architecture	13
4.3 Inputs	14
4.4 Training	15
5 Experiments	16
5.1 Convergence	16
6 Conclusion	19
Bibliography	20

List of Figures

2.1	Typical scenario of classical reinforcement learning approach.	4
2.2	Original Flappy Bird game design.	6
2.3	Simplified example of the Flappy Bird game.	7
3.1	On the left code for TensorflowJS, on the right for Keras.	8
4.1	Environment architecture overview.	13
4.2	Final model architecture view, 2 input values, layer with 4 neurons, layer with 4 neurons, output 2 values.	14
5.1	Losses of some runs.	17
5.2	Number of episodes before convergence.	18

List of Abbreviations

RL Reinforcement-Learning
ANN Artificial-Neural Network

Chapter 1

Introduction

1.1 Motivation

Working as a Data Science Engineer requires a lot of skills and knowledge. Data Scientist has a lot of tasks to solve, for example, understand nature of data, visualize data, get a business understanding of a process, build a model which will suit needs of a customer, deploying the model to production and many more task. Here I will focus on one particular task which is deploying a model to production in order to get output results. Historically one could only deploy model which does computations on a back end and never on a front end, meaning a server is responsible for all the hard computations. That leads to having huge clusters running models on a back end and outputting results to the front end. Which eventually requires more amount of money for the business to host. In the recent years, deploying model which operates on a front end became possible so the purpose of this paper is to show an overview how browser can handle very demanding computations like working with Artificial Neural Network and therefore decrease the number of servers and their computational power. Which will impact business in a money-saving way. The selected area of an AI is - Deep Reinforcement Learning. It is a machine learning concept where the agent is put into a certain environment, the agent has to take actions which are computed with Artificial Neural Network in order to maximize some cumulative reward. This approach doesn't have much of business value since this type of machine learning requires a lot of trials and errors until an agent becomes an expert in the given environment. However, in recent years, OpenAI has been working on the bot which used a similar approach to learning, for one of the most complex games ever to exist - Dota 2. *Description of OpenAI mission* Their motivation was to create a bot for the game, and transfer knowledge gained from this game to more useful applications.

1.2 Frontend computations

There are a few motivational factors behind using exactly browser as a computational environment. First is business value - it is very inexpensive to have a server on the backend which just serves model and weights which are being fetched from the front end. In contrast to having model and weights loaded on the back end and making a prediction for every single user. A certainly smaller amount of less powerful servers leads to a reduced amount of cost for their maintenance. The second reason is the rapid development of technologies and hardware acceleration. In the modern age, one of the most used applications of all time is the browser. Every user has at least one browser on their laptop, PC, smartphone, etc. That demand for browsers led to its fast development and upgrades. The most used programming language

in the browser environment is Javascript, this language has the biggest community around the globe. Browser as a program is a very unique software product, it has a huge breaking point, meaning it will escape crashing in every possible way in order not to spoil user experience by crashing some page. Additional important aspects of the browser are that it renders all the content and computes Javascript on the client side, reducing the load on the server side. *Brief history of Javascript*

1.3 Reinforcement Learning

Every specific topic in Machine Learning is very unique and interesting. A lot of those topics have a huge business impact and bring a lot of value, for instance, time series predictions, computer vision, recommender engines. *Business value of RL* However, there are some topics which are not really applicable to the modern needs of a business, nevertheless, those topics are still interesting and relevant. One of these less valuable topics is Reinforcement Learning. It has a pretty unique approach to solving tasks. It requires a lot of repetitions until the model converges. Unlike supervised learning it can't simply label each state, each state is getting some value when the agent comes to the state thus it prevents us from gathering every possible state of the game to the training set. Anyway, there are a few real-world tasks which might be solved using mentioned learning approach, for instance, resources management in computer clusters. This task requires delegating specific jobs to specific computers in order to reduce slowdowns and increase the overall performance of a system. The other example could be robotics, it can help robots become better at movement, make robots move more precise and accurate.

1.4 Goal

The aim of this work is to create and teach an agent based on Deep Reinforcement Learning, also create an environment which will operate in a similar way to game Flappy Bird. This work has to show that browser is capable of Neural Network computations and can be pretty efficient in reinforcement learning for Flappy Bird. Further training of an agent is split into 2 phases. In the first phases, an environment will be stable, meaning that the complexity of the world is small. When agent will show that it's capable of passing stable world, agent and environment will improve its complexity and proceed to the next phase. The second phase is gradually increasing randomness of the world. Built agent will represent a bird, which as a decision maker will have an Artificial Neural Network. Next step is to allow the agent to take actions based on its computed weights. Weights will be updated over time according to the reward factor of the world.

Chapter 2

Background Information

2.1 Artificial Neural Network

In this paragraph, I will cover key components and key terms which made the possible implementation of this work. The first component is Artificial Neural Network (ANN), those are complex computing systems, their name stems from biological neural networks of brains, but in real life there is almost no connection between those phenomena. The neural network is not an algorithm, but rather a framework which has certain rules and certain number of tools inside it. This framework can be used for many different machine learning algorithms in order to find hidden patterns, to develop certain behaviours and process complex data inputs. Such systems "learn" during training process, to perform tasks by considering examples, generally without being programmed with any task-specific rules. An ANN is based on a collection of connected units or nodes called neurons or nodes, those neurons performing some mathematical operations on each step. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Decrease and increase in weights are determined by the loss function and performed by the optimizer. Typically, artificial neurons are aggregated into layers and can perform strict number of tasks. Krizhevsky, Sutskever, and Hinton, 2012 Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly after traversing the layers multiple times. *Artificial Neural Networks*

At the end of a day, ANNs represent a very complex mathematical formula which would be impossible to create manually by hand.

An important part here is weight update, it is made possible due to the invention of an algorithm called backpropagation. This algorithm allows efficient weight update through the computation of a gradient, based on the loss function. The loss function is a function which indicates how far or close ANN is getting to the local minimum or in the other words to the desired result. *Description of a backpropagation*

2.2 Reinforcement Learning

Reinforcement Learning (RL) is an area of Machine Learning that deals with sequential decision making. The core component of RL is an agent which is put into a certain environment. The idea behind this approach is that the agent will learn a good/winning behavior through the pass of trials and errors. Agent contacting with an environment through states and actions. Difference between RL and classical algorithms which aim to find the optimal policy such as dynamic programming is that an agent doesn't have to have a full picture of the world. Meaning RL agent

can operate and discover the world in the process of training, therefore enhancing its action taking mechanism. Reinforcement learning starts from Markov decision process. *Introduction to Reinforcement Learning*

The typical scenario of learning using this approach looks somewhat similar to this:

- pass the state from the system to an agent
- agent chooses action to take based on the state
- pass selected action to the environment
- get a reward for the taken action
- update weight matrix according to previously selected action and reward

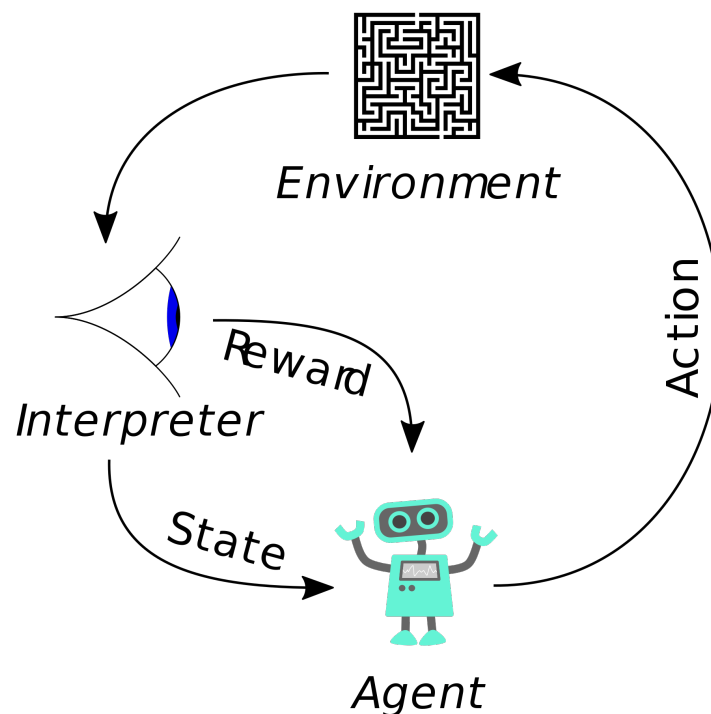


FIGURE 2.1: Typical scenario of classical reinforcement learning approach.

There are a few working algorithms within RL. They often rely on the weight matrix which is being updated over time with every new state. So the decision-making process could be described as follows agent receives state of the world in numeric form, where each entry corresponds to some information of the world. Those entries are treated as X and being passed to the matrix in order to find resulting Y which represents an action to be taken. After matrix multiplication over X , the action is found and returned to the system as a decision that the agent made. Over the next iteration, the world state is changed and the new world state is passed to the agent and also reward for the previous action is calculated and passed to the agent as well. Now agent has to decide whether the previous action was correct or not according to the received reward. Next step is to adjust weights in the matrix in order to maximize reward in the future. In the classical RL exists various techniques

which allow reducing the number of weight kept in the matrix in order to simplify the world. One of those techniques called “binning”. It can be simply described as breaking down multiple different variables into separate bins. For instance, we can bin a distance to “more than 5 cm” or “less or equal to 5 cm”, this will result in having 2 bins. Sometimes this matrix is called Q-table and it is a core feature of RL which differs it from other learning methods. One of the most used ones and the one I used in this papers is called SARS. This abbreviation stands for State Action Reward State. At any point of time, this algorithm saves such variables as a state, an action which was taken from this state, a reward which was given for the taken action and resulting state. Often times, additional variables like the status of being dead or alive, is this a new game or not is saved as well. [A brief introduction to RL](#)

2.3 Frameworks

Nowadays coding computations for ANNs is never been easier. There are a few main frameworks which has been open sourced and publicly available. Most used ones are PyTorch, TensorFlow, and Keras. [Deep learning frameworks](#) Basically, those frameworks offer a suitable interface for programming ANNs which can be called via python. Here we will focus on the usage of Tensorflow because this is the only framework which can use Javascript and be run in the browser. This framework was created and open sourced by Google. The later company released TensorFlowJS, it allowed programming ANNs using Javascript. Javascript nowadays is the most supported language among browsers.

2.4 Flappy bird

This is one of the most popular game of its time. However, this game didn't last long enough. Let's dive into some history to discover why. 'Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen, under his game development company dotGears. The game is a side-scroller where the player controls a bird, attempting to fly between columns of green pipes without hitting them. Nguyen created the game over the period of several days, using a bird protagonist that he had designed for a canceled game in 2012. This game was released in May 2013 but received a sudden rise in popularity in early 2014. The game was free to play and Nguyen said in an interview with The Verge that the game was earning around \$50,000 a day in revenue through its in-game advertising. Original version of the game contained pipes and looked something similar to this.' [History of a Flappy Bird game](#) The game is truly addictive. It doesn't really matter whether a person plays or watches someone plays it. I find very fascinating to watch how AI plays it.



FIGURE 2.2: Original Flappy Bird game design.

My game is a bit simplified in terms of graphic. However, in terms of speed my created Flappy Bird is a bit faster.

For the sake of speed of rendering and training therefore graphic intentionally was simplified. The game sense and rules remained the same.

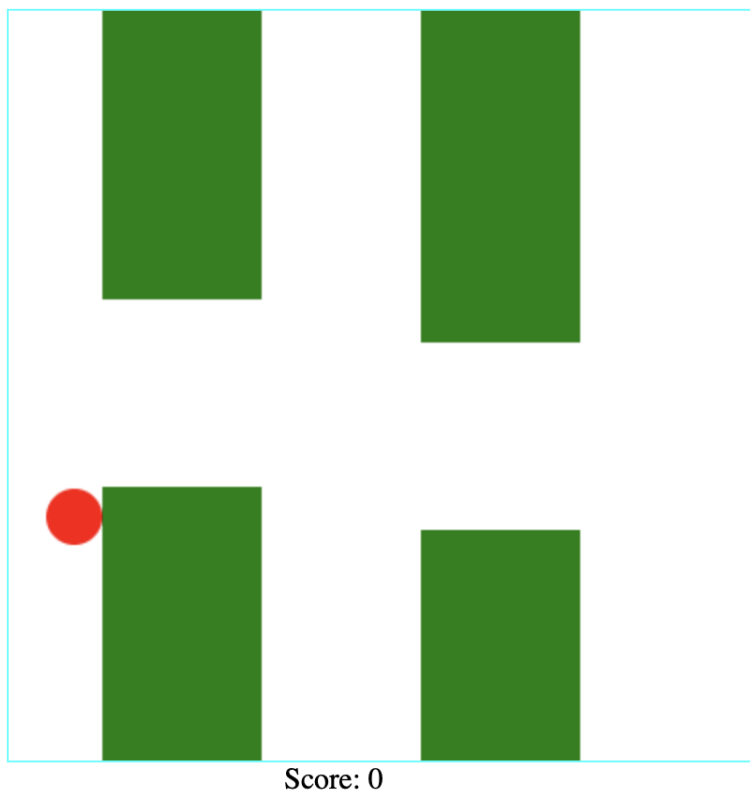


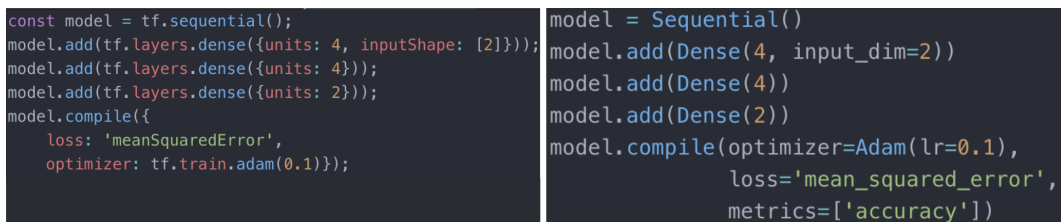
FIGURE 2.3: Simplified example of the Flappy Bird game.

Chapter 3

Related Works

3.1 TensorflowJS

It is a relatively new framework created by Google. It is using Javascript as programming the main programming language. This has certain up and downsides. One of the bad things about it is that it is less efficient in computations which results in slower training/testing processes. Not to mention C++, it loses to Python when training small networks in 1.5 times of speed. And training a large network might be slower in 10 times. However, this framework has its advantages, the first one is that it runs in the browser on the client side, and the browser is the most used program so far. This advantage covers cross-platform execution in some way since the browser is installed on the major number of devices across the globe. The second advantage is that it can be run using Node so it can perform computations on the backend as well. Abadi et al., 2015 I also prefer tensorflowjs compared to the original tensorflow, since it has syntax more like a Keras, which is more suitable and easier. Even though



```

const model = tf.sequential();
model.add(tf.layers.dense({units: 4, inputShape: [2]}));
model.add(tf.layers.dense({units: 4}));
model.add(tf.layers.dense({units: 2}));
model.compile({
  loss: 'meanSquaredError',
  optimizer: tf.train.adam(0.1)});

```

```

model = Sequential()
model.add(Dense(4, input_dim=2))
model.add(Dense(4))
model.add(Dense(2))
model.compile(optimizer=Adam(lr=0.1),
              loss='mean_squared_error',
              metrics=['accuracy'])

```

FIGURE 3.1: On the left code for TensorflowJS, on the right for Keras.

this framework is new, but already has some applications and interesting examples. One of the most recent achievements in the direction of the Web and TensorflowJS is **ml5.js**. Creators of this framework say that they are heavily inspired by **Processing** and **p5js**. This library allows doing machine learning classification in a few lines of code, and for sure it operates on the client side, and demands no computations on the back end. In order to have it working user just have add script reference to the library in the head of the document. Then add upload field for the image. Next step is just write a few line of code in order to classify image. Authors also claim they have plenty of classifiers and other machine learning algorithms in the library, and their usage is fairly simple. [Overview on how ml5js works](#)

One more useful and interesting example of using TensorflowJS is **face-api.js**. This library allows doing face detection using both Node and browser. It also gives user an ability to detect face landmarks and facial expressions. And the last feature is age and gender detection. This application is very notable since it uses browser and can be run from mobile device. That means developing demos or even some solutions using android / ios applications might become irrelevant and overcomplicated

compared to simple solution using browser and external library. Authors claim they have ssdMobilenet inside of their system. This model is well known for its speed and size. [Description of how face-api.js works](#)

3.2 Deep Reinforcement Learning

It is a subcategory of RL. This approach is relatively new, it stems from classical reinforcement learning and ANNs approach. It became more and more popular in tasks which involve RL because it does not necessarily require bins in order to reduce dimension space of an input data. Often times raw state data from the world is directly passed to the model. In this work I particularly use Deep Q-learning (DQN), [Deep reinforcement learning, DQN](#) the idea behind this approach is to combine Q-learning with a neural network. ANN is a key factor in this type of learning since it serves as a place where weights are stored, instead of a matrix, as in traditional Q-learning. [An Introduction to Deep Reinforcement Learning](#) This method of learning definitely has its advantages and disadvantages. To advantages, we can add that it doesn't require binning, the decision-making process can be more complex than 2d matrix operations. However, there are some disadvantages as well, for example, it requires more data samples to learn from since weights are stored in ANN and being optimized through gradient descent based on loss. This process has rather small tweaks in weights and needs more samples in order to converge. One more disadvantage is that this method requires more computational power than a standard Q-learning algorithm. But in nowadays increase in computational power can be easily achieved so the last disadvantage is arguable.

3.3 DeepLearningFlappyBird by yenchenlin

This work consists of GitHub code and references to a few related articles. Direct reference to an article on which this code is based doesn't exist, probably there is no such an article. The main programming language of this work is python 3. Stack of technologies is - Tensorflow 0.7, pygame, OpenCV. For training the neural network and weight updates, for creation of a gaming environment, for passing images of the world to the neural network respectively. [Computer Vision for Flappy Bird](#)

The author uses pygame to create an environment of the game. As a learning algorithm author selected RL. However, a significant feature of this work that it has a deep neural network which as an input accepts an image of the game field. So it grey scales image, reshapes it and then feed into ANN. The core component (ANN) author selected Convolution Neural Network since it is the best performing approach for working with images. The created model consists of a few layers, important mention is that every convolutional layer is passed to relu activation directly. The network starts from convolution + bias layer, selected parameters for the convolutional layer are: kernel size - 8x8; striding - 4; the number of kernels - 32. Next layer is a standard max pooling with striding of 2. Next two layers are convolutional with no max pooling involved, its parameters are (4, 4, 64, 2) and (3, 3, 64, 1). Finally flattening resulting images passing to fully connected layer in order to map to resulting array of 2 numbers. (might add weakness of the model) As an output model predicts a reward which could be achieved in case of taking jump action or not.

Training of this model includes the state as an input image, reward, action and next state as another image. Implementation of the game is rather heavy and written in a script like the style. As a reward, the author gives "1" if the bird is alive and "-1" if

the bird is dead. For the best performance author proposes a number of constant to take into account for instance number of previous episodes which are then used for training is equal to 50000. Also, the author proposes an observation stage from the beginning of the game. This stage is played with high exploration rate meaning, the bird is doing random moves with no predictions involved. This stage is important for initial training because it possibly fastens convergence time. So constants for this part are number of observational episodes before first training - 100000, number of episodes where the bird is able to explore states at random is set to 200000.

3.4 FlappyBirdRL by SarvagyaVaish

This flappy bird made to work in the browser, the game is calculated and played only using front end. An author as he claims didn't develop the game, he developed only an algorithm for playing. In his blog author describes that he used classical RL approach - Q learning. State space, which is basically a matrix where weights are stored, is split into 3 variables - vertical distance to the lower pipe, horizontal distance to the next pair of pipes, status dead or alive bird. Action space in this game is standard - do nothing or jump. Rewards author selected differently from the previous creator bird gets "+1" if taken action didn't kill the bird and "-1000" if a bird has died. I believe that this difference in rewards is supported by the difference in a number of observations of each state. Since the bird is primarily alive, meaning in the game there are way more states where a bird is alive compared to the number of states where the bird is dead. Thus the author decides to punish every move leading to death way more than every move that keeps a bird alive. The author mentions that it took around 6-7 hours for a bird to become pretty good in the game and score 150 points. [*Article about classical RL for Flappy Bird*](#)

Chapter 4

Solution overview

4.1 Environment

I believe it is worth explaining why I didn't pick any of existing environments but developed my own. There are a few main reasons to make such a decision. The first one is because almost every created RL agent for Flappy Bird operated using python. So far there is no chance to run python simulator in the browser on the client side. The second reason is that existing simulators which are written in Javascript a long time ago. They primarily use JQuery library, I am not a fan of this library, and in times of having ECMA6 as standard - JQuery is one of the dead approaches. *Usage of JQuery for the Flappy Bird*

The created world consists of a few pieces: blocks, bird, game field, agent, world. Blocks are represented by the class which has its x and y coordinates, height and width of the block. Also, this class, as well as any graphical element, has to render method which renders essence to the gaming field using canvas. The bird is a simple struct-like class which contains core information about the bird. Its x and y coordinates as well as render method. The game field is represented by class Game which does all function of the game, for example, starting a new game, rendering block and bird, check if the game is over, moving bird and blocks to the next position, performs a jump action if needed. Next is Agent class, this is basically RL agent which decides on an action to take based on the given input. There is also class - World, which creates a new game and new agent and also renders useful information to the screen.

Components

Agent.js - class represents Deep RL network. This class represents core algorithm of the system which does decision making and outputs actions. This component is crucial as it contains core ANN which does action taking. It also decides which reward is provided to the ANN for each action. Inside this component it has 2 main methods which are being called from outside. Method *act* accepts as an argument world state provided by another component. To be precise world state is an object, which has is packed and passed as an argument by the world. State object contains a lot of useful information which describes world. It has such properties: "bird", "blocks", "ticks", "gameIsOver", etc. Mentioned properties are essential in order to take an action. Next step is to predict rewards and according to the highest reward possible determine the next action to be taken. Next step is to save the current state to the history array after that, it updates the previous state and assigns a current state to the variable of *nextState* of the previous state from the history. The last thing is that this method returns an action which the agent decided to take. The second important method is *retrainModel*, it accepts no inputs. This method gathers a batch of states from the history, puts it to the arrays of Xs and Ys and passes to the model in order to retrain it.

Bird.js - struct-like a class which is used to set initial variables of the bird. Variables include x and y coordinates, jump array it is used to encode jump shift in any given point of time. Render method outputs a graphic representation of the bird on canvas.

Block.js - class which represents pipes in the original game. Consists of a few methods and properties. Key properties are *lowerBlock* and *upperBlock* they represent upper and lower pipes in the original game. Render method works similarly to the one in bird class. Other methods are pretty self explanatory.

config.js - the most influential file. It has all the key factors of the game such as height and width of the world, game speed, size of the blocks and bird, etc. Through this class, all the parameters changes are made.

Game.js - class which represents a game of flappy bird. The *startNewGame* method creates a new bird and game field with only one block. Method *gameIsOver* is used on every step of the game, its aim is to check whether the bird is dead or alive. This method checks collision with game field and blocks. Next important method is *performAction*, as an argument accepts action which is considered to be 0 or 1, do nothing or jump accordingly. One of the key methods is *getFrame* method first of all this method shifts bird to the next position, shifts blocks. Returns current game state as an object. Method - *renderFrame* is called immediately after *getFrame* method in order to render game changes to the canvas. In **Game.js** only three methods are being called from outside of the class which is *getFrame*, *renderFrame*, and *gameIsOver*.

World.js - the purpose of this class is to put together **Agent.js** and **Game.js** classes, it represents world order for both and triggers the most important methods of both. Besides the constructor, this class consists of 2 methods. First is *async graphicMode* with no inputs required, this method does all the general flow of the game and RL process. Key steps in the flow are:

- receive current state of the system from **Game.js**
- pass world state to the **Agent.js**
- return action from the **Agent.js** to the **Game.js**

Beside these very important steps this class does supportive functions. The first of them is rendering current state to the canvas so the user can see it. The second is to call the method to retrain the model under some certain conditions. These conditions are purely custom and selected by me, those can be number of episodes to retrain after, at which point of the game is to retrain model, etc. Last things are to render useful information to the user interface and make artificial pauses so the game won't run too fast.

utils.js - file consists of useful functions which don't really have a common context of execution. Those functions are *calcDistance*, *getRandomInt*, etc. Every function of this file is being exported.

index.js - just an entry point of an application. It creates world and plays the game. This file is an entry point for webpack builder.

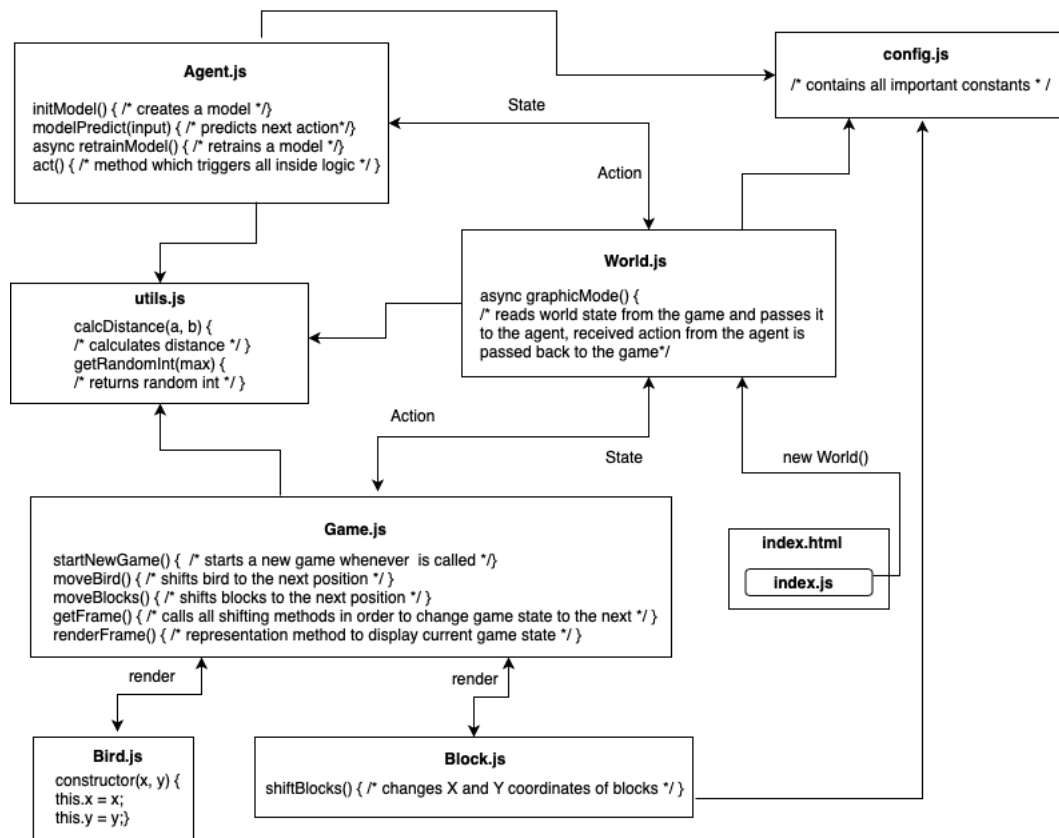


FIGURE 4.1: Environment architecture overview.

4.2 Model architecture

As mentioned earlier Deep Q learning is selected approach. So the core component here is ANN primarily based on fully connected layers. So the final architecture has following layers: starting from a dense layer which passes 2 input values to 4 hidden parameters, next dense layer maps 4 neurons to next 4, and the final layer outputs 2 values which later are interpreted as actions. This architecture showed the best performance compared to other tested architectures. However, I will describe the previous architectures and its performance. Initial architecture consisted of these layers: mapping input 2 values to 4 neurons, 4 neurons mapped to 2 output values and interpreted as a predicted reward for each action. This architecture could only

perform well in a static world, the meaning the distance between blocks and blocks height is static, so the learning algorithm needed to learn only a little to be able to pass the game. Next tested architecture was a bit more complex, it consisted of mapping 2 input values to 4 neurons, mapping 4 neurons to next 2 neurons, mapping resulting 2 values to 2 output values. This architecture performed well in a static world, however, convergence in the world with big random factor required a lot of time. Randomness measured in height of the blocks and distance between blocks. Every model had the same loss function since models are trying to predict reward in case of taking action, the loss is calculated as the mean squared error between predicted reward and actual reward.

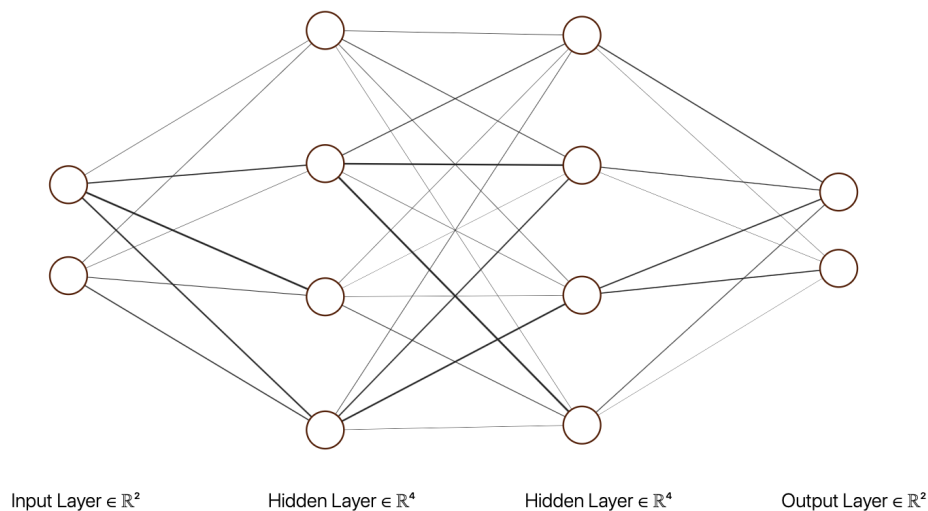


FIGURE 4.2: Final model architecture view, 2 input values, layer with 4 neurons, layer with 4 neurons, output 2 values.

4.3 Inputs

Every RL task starts from the environment and a space state. In this task, the environment has such parameters as the location of the blocks, width, and height of each block, position of the bird. In order to understand selected inputs, I need to explain the conditions of the death of the bird. The bird dies when it collides with any block or any part of the world. So I had to find a set of variables which would definitely determine the position of the bird. Therefore, in initial model setup input values were distance to the ground and distance to the closest lower block, distances calculated as euclidian. The bad side of this approach is that the distance to the lower block is always floating point, meaning there are more variations of input states which model receives. It might have been one of the most influential factors of underperformance of initial architectures. I was concerned with floating points calculations and had to come up with another 2 input variables which have to determine the

position of the bird. My next and final set of input variables consisted of values calculated in a different fashion. So first is the distance between lower block Y and bird Y, which is always an integer, since the bird is shifted by integer values. And blocks coordinates are always integers. The second value is the distance from the birds X to the X of end of the closest pair of blocks, which is also always an integer.

4.4 Training

The process of training is the following. It starts from the observation phase which lasts around 100 states, the state is a measure of 1 in-game step. During this phase agent makes random actions, this phase is created in order to have the agent gather more different states to learn from, and not same states repeated from episode to episode. The observation phase is followed by the actual training. Initial attempts to the training were slightly different from the final one and they were less efficient. First attempt was using 4000 saved states, it took more time and gave less accuracy because the agent was heavily influenced by the data from the past, in order to remove the last effect I needed to come up with a discount factor or smaller batch size. The second attempt was to train agent more often, every second game state, and take a random batch of size 32. This attempt gave absolutely different outcomes compared to the first one. It brought rapidly changing the behavior of an agent. It's is not a good quality since sometimes agent got the right understanding of the game, however, due to often retraining fell out of that local minimum. So the last and final changing attempt works in the following way. Further retraining of the model is happening after each death of the bird. They are done in the following way, retrieve 1000 last states from the history, select inputs variables for the model from states and put them into an array of Xs. Select reward from each action taken from the previously selected step, put the reward in a new array with 2 numbers, under the index of taken action from that step. Last 2d array is Ys which are passed directly to the model. After that model is being retrained, the loss is calculated as the mean squared error between predicted reward and actual reward. Accordingly, to the loss of every weight in each dense layer is being pulled to the local minimum.

Chapter 5

Experiments

This chapter is dedicated to experiments I have conducted with working agent in my own created world.

5.1 Convergence

First of all, let me describe world configuration under which this test is being held. Here are some useful constants and their values:

- world resolution - 400x400 px
- bird size - 15x15 px
- bird spawn point - (35, 200)
- block width - 85px
- falling speed - 6px
- horizontal speed - 2px
- jump length - 25 game ticks
- jump power - 7px
- gap between blocks - 100px

Next important constants are connected to the ANN model which is a decision maker in this approach:

- Adam optimizer with learning rate - 0.1
- exploration rate - 0.001
- architecture of the model stays immutable
- batch size - 1000
- loss function is - 'rmse'
- horizontal speed 2px

In this model, the exploration rate has the possibility of 0.001 to occur at any moment of the game.

The world is not remaining constant, so it has randomness to it:

- block heights [115px, 165px]

- distance between blocks [125px, 155px]

Convergence condition is equivalent to scoring 50+ points in the game. Model inputs are the same as described in the previous chapters. First 2500 states in-game are passed without training. After that stage training performed on every death. Here are some results of a few arbitrary selected runs.

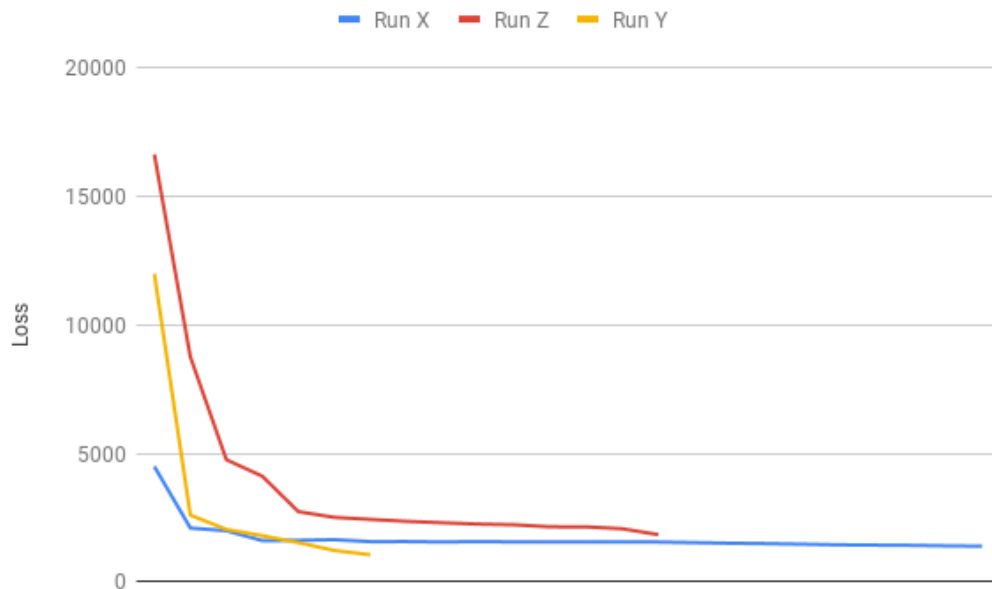


FIGURE 5.1: Losses of some runs.

This diagram shows how losses decrease as weights getting closer to the point of convergence. At which agent will be able to play the game without dying. One can notice that it's not really important how weights are initially set. However, it is important which learning rate is set, because when the loss is almost at the point of convergence it takes extra episodes to finally got the right weights set up.

Here is an example of a diagram representing number of episodes before convergence.

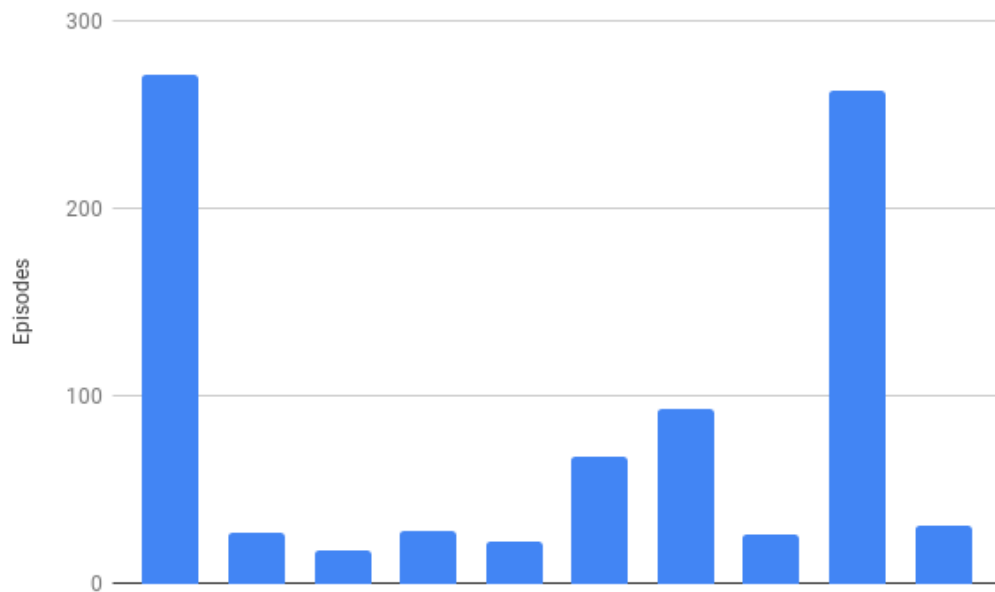


FIGURE 5.2: Number of episodes before convergence.

As one may notice a number of episodes before convergence is varying from 18 to 271. Its mean is 94,1, its median is 29. In my opinion, there are a few important factors which influence this value, those are learning rate, batch size, retaining frequency, exploration rate. In order to find optimal combinations further test has to be executed.

Chapter 6

Conclusion

I can certainly say that I accomplished the set goal of creating an environment, then creating an agent based on Deep RL and make the agent play the best. However, I believe the work is not done yet, since yet I didn't find the best parameters for the model. So far, I didn't teach AI to pass levels which have inhuman complexity. There is plenty of work to do even though the main goal is achieved!

Bibliography

- A brief introduction to RL.* <https://medium.freecodecamp.org/a-brief-introduction-to-reinforcement-learning-7799af5840db>.
- Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from [tensorflow.org](https://www.tensorflow.org/). URL: <https://www.tensorflow.org/>.
- An Introduction to Deep Reinforcement Learning.* <https://arxiv.org/pdf/1811.12560.pdf>.
- Article about classical RL for Flappy Bird.* <https://sarvagyaVaish.github.io/FlappyBirdRL/>.
- Artificial Neural Networks.* <https://arxiv.org/pdf/1901.05639.pdf>.
- Brief history of Javascript.* <https://developer.mozilla.org/ru/docs/Web/JavaScript>.
- Business value of RL.* <https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12>.
- Computer Vision for Flappy Bird.* <https://github.com/yenchenlin/DeepLearningFlappyBird>.
- Deep learning frameworks.* <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>.
- Deep reinforcement learning, DQN.* <https://arxiv.org/pdf/1312.5602.pdf>.
- Description of a backpropagation.* <http://neuralnetworksanddeeplearning.com/chap2.html>.
- Description of how face-api.js works.* <https://github.com/justadudewhohacks/face-api.js>.
- Description of OpenAI mission.* <https://openai.com/five/>.
- History of a Flappy Bird game.* https://en.wikipedia.org/wiki/Flappy_Bird.
- Introduction to Reinforcement Learning.* <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- Overview on how ml5js works.* <https://ml5js.org/docs/getting-started>.
- Usage of JQuery for the Flappy Bird.* <https://github.com/SarvagyaVaish/FlappyBirdRL/blob/master/index.html>.