CIS 8395 - Big Data Analytics Experience (Fall 2021)

Project Paper

Dec 10, 2021

Topic: Reinforcement Learning for Flappy Bird Game

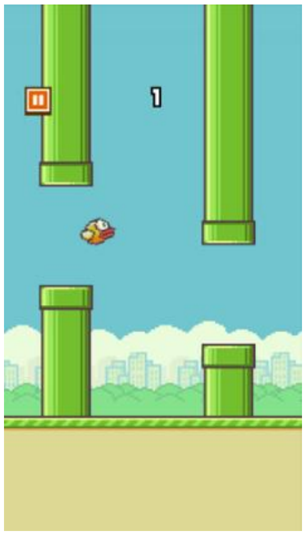
Team Name: JAKS

Team Members:

1. Jay Patel
2. Ajay Nawale
3. Kashyap Dave
4. Saurabh Binani

Table of Contents

Introduction



Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen under his game development company Gears. 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 cancelled game in 2012.

The game was released in May 2013 but received a sudden rise in popularity in early 2014 and became a sleeper hit. Flappy Bird received poor reviews from some critics, who criticized its high level of difficulty and alleged plagiarism in graphics and game mechanics, while other reviewers found it addictive. At the end of January 2014, it was the most downloaded free game in the App Store for iOS. During this period, its developer said that Flappy Bird was earning $50,000 a day from in-app advertisements as well as sales.

Flappy Bird was removed from both the App Store and Google Play by its creator on February 10, 2014. He claims that he felt guilt over what he considered to be its addictive nature and overuse. Its popularity and sudden removal caused phones with the game installed before its removal to be put up for sale for high prices over the Internet.

Gameplay:

Flappy Bird is an arcade-style game in which the player controls the bird Faby, which moves persistently to the right. The player is tasked with navigating Faby through pairs of pipes that have equally sized gaps placed at random heights. Faby automatically descends and only ascends when the player taps the touchscreen. Each successful pass through a pair of pipes awards the player one point. Colliding with a pipe or the ground ends the gameplay. During the game over screen, the player is awarded a bronze medal if they reached ten or more points, a silver medal from twenty points, a gold medal from thirty points, and a platinum medal from forty points.

Data

Data Sourcing:

Data sourcing is the process by which companies extract and integrate data from multiple internal and external sources. This procedure creates the firm's data infrastructure that is used for handling daily workflows and achieving various business objectives. As such, this process is an integral part of doing business in the heavily data-based markets of today.

Data sourcing for Reinforcement learning (RL):

This technique implementation utilizes input data as sample sequences that consist of states, rewards, and actions. Based on such training examples, it allows the RL agent, to learn an optimal policy which defines the best possible action in each and every state. So, it is evident that there is no input data required to develop a reinforcement learning.

Data Cleansing:

Data cleansing (also known as data cleaning) is a process of detecting and rectifying (or deleting) of untrustworthy, inaccurate or outdated information from a data set, archives, table, or database. It helps you to identify incomplete, incorrect, inaccurate or irrelevant parts of the data. By doing this you can then replace, modify, or delete the bad data. Data cleaning can be performed interactively with data wrangling tools, or as batch processing through scripting.

Below are the data cleansing/pre-processing steps that will be performed for our project:

- Convert image to grayscale: This process helps to convert the colorful elements into grayscale as they do not have any impact on the scoring system of the game. This will help remove any color biases and train faster as well as with higher accuracy.

- Resize image to 80x80: The environment which consists of the obstacles and the bird is getting covered in the 80x80 square frame. Any additional size of the image will not be useful so we are removing the same.

- Stack last 4 frames to produce an 80x80x4 input array for network: This step will help us gain every instance of the gameplay and combine them into one action to make a better action-reward mechanism.

- Removal of the background appeared in the original game can make it converge faster: Background elements like sky, clouds will not be useful in the RL algorithm and unnecessary add to the noise in the data. Hence, they are planned to be removed from the original game. The background will be made to a solid color, preferable black as we are converting it to grayscale.

Data Quality:

Data quality refers to the development and implementation of activities that apply quality management techniques to data in order to ensure the data is fit to serve the specific needs of an organization in a particular context. Data that is deemed fit for its intended purpose is considered high quality data.

Examples of data quality issues include duplicated data, incomplete data, inconsistent data, incorrect data, poorly defined data, poorly organized data, and poor data security.

An increasing number of organizations are using data to inform their decisions regarding marketing, product development, communications strategies and more. High quality data can be processed and analyzed quickly, leading to better and faster insights that drive business intelligence efforts and big data analytics.

Good data quality management helps extract greater value from data sets, and contributes to reduced risks and costs, increased efficiency and productivity, more informed decision-making, better audience targeting, more effective marketing campaigns, better customer relations, and an overall stronger competitive edge. Poor data quality standards can cloud visibility in operations, making it challenging to meet regulatory compliance; waste time and labor on manually reprocessing inaccurate data; provide a disaggregated view of data, making it difficult to discover valuable customer opportunities; damage brand reputation; and even threaten the safety of the public.

Six elements of Data Quality:

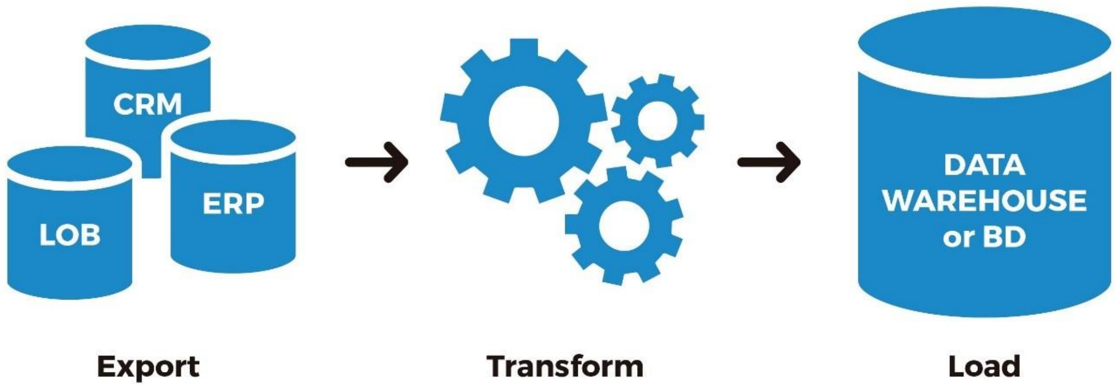
- **Completeness:** Completeness is defined as expected comprehensiveness. Data can be complete even if optional data is missing. As long as the data meets the expectations then the data is considered complete. For example, a customer’s first name and last name are mandatory but middle name is optional; so a record can be considered complete even if a middle name is not available.
- **Consistency:** Consistency means data across all systems reflects the same information and are in synch with each other across the enterprise. Examples: A business unit status is closed but there are sales for that business unit, Employee status is terminated but pay status is active.
- **Conformity:** Conformity means the data is following the set of standard data definitions like data type, size and format. For example, date of birth of customer is in the format “mm/dd/yyyy”.
- **Accuracy:** Accuracy is the degree to which data correctly reflects the real world object OR an event being described. Examples: Sales of the business unit are the real value, Address of an employee in the employee database is the real address.
- **Integrity:** Integrity means validity of data across the relationships and ensures that all data in a database can be traced and connected to other data. For example, in a customer database, there should be a valid customer, addresses and relationship between them. If there is an address relationship data without a customer then that data is not valid and is considered an orphaned record.
- **Timeliness:** Timeliness references whether information is available when it is expected and needed. Timeliness of data is very important. The timeliness depends on user expectation. Online availability of data could be required for room allocation system in hospitality, but nightly data could be perfectly acceptable for a billing system.

Extract, Transform, Load (ETL):

Purpose: - ETL allows businesses to consolidate data from multiple databases and other sources into a single repository with data that has been properly formatted and qualified in preparation for analysis.

What is ETL: - As the amount of data, data sources, and data types at organizations grow, the importance of making use of that data in analytics, data science and machine learning initiatives to derive business insights grows as well.

ETL stands for extract, transform, and load, is the process engineers use to extract data from different sources, transform the data into a usable and trusted resource, and load that data into the systems end-users can access and use downstream to solve business problems.



Extract:

The first step of this process is extracting data from the target sources that are usually heterogeneous such as business systems, APIs, sensor data, marketing tools, and transaction databases, and others.

Three Data Extraction methods:

Partial Extraction – The easiest way to obtain the data is if the if the source system notifies you when a record has been changed

Partial Extraction (with update notification) - Not all systems can provide a notification in case an update has taken place; however, they can point to those records that have been changed and provide an extract of such records.

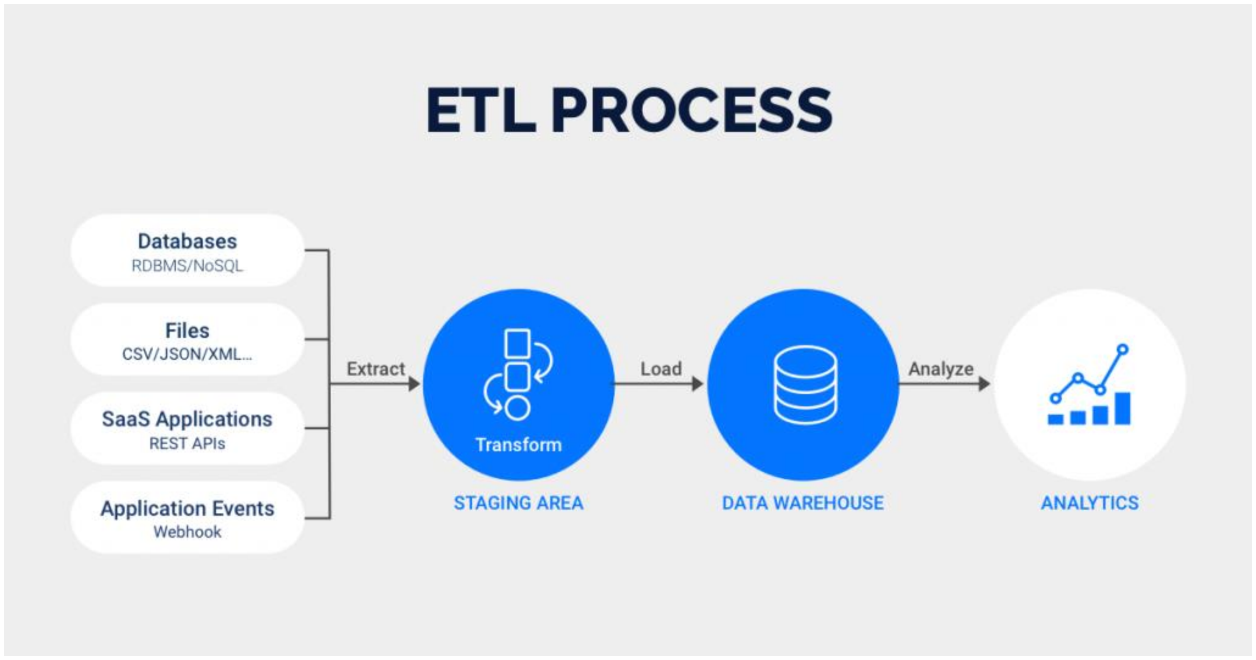
Full extract – There are certain systems that cannot identify which data has been changed at all. In this case, a full extract is the only possibility to extract the data out of the system. This method requires having a copy of the last extract in the same format so you can identify the changes that have been made.

Transform:

The second step consists of transforming the raw data that has been extracted from the sources into a format that can be used by different applications. In this stage, data gets cleansed, mapped and transformed, often to a specific schema, so it meets operational needs.

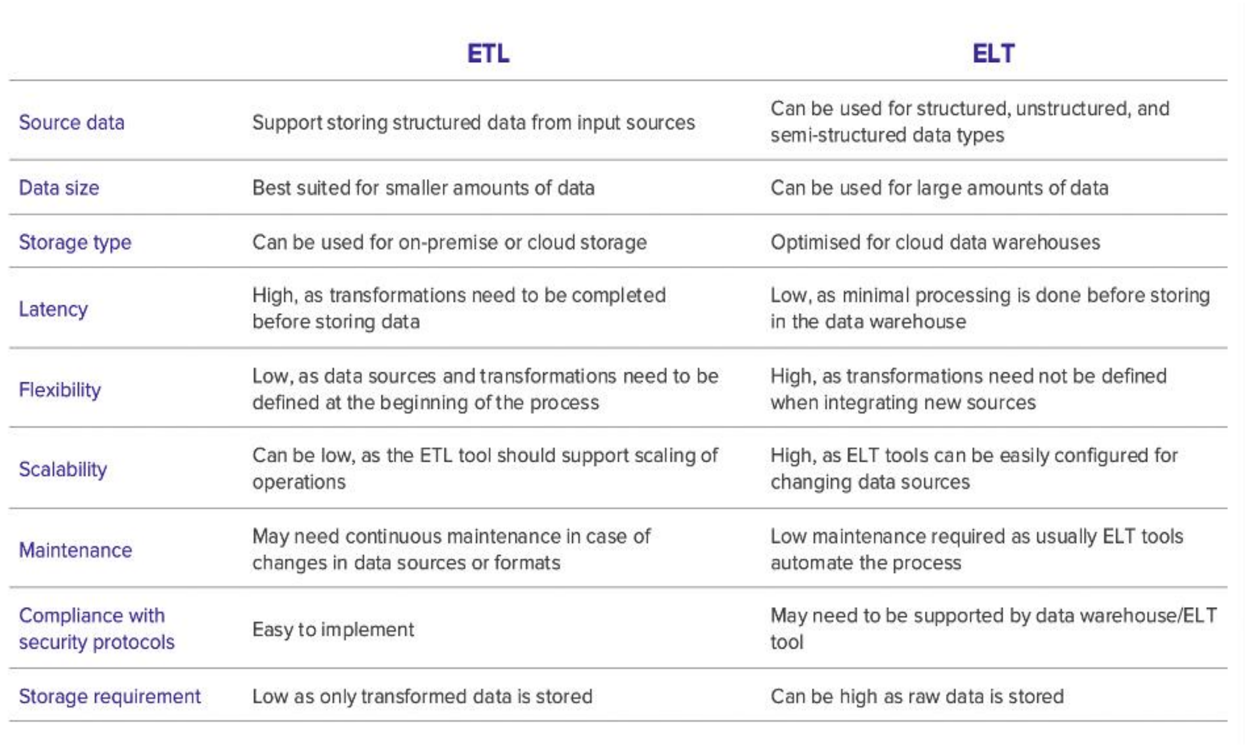
Load:

Finally, the load function is the process of writing converted data from a staging area to a target database, which may or may not have previously existed.



An ETL pipeline (or data pipeline) is the mechanism by which ETL processes occur. Data pipelines are a set of tools and activities for moving data from one system with its method of data storage and processing to another system in which it can be stored and managed differently. Moreover, pipelines allow for automatically getting information from many disparate sources, then transforming and consolidating it in one high-performing data storage.

ETL VS ELT:



	ETL	ELT
Source data	Support storing structured data from input sources	Can be used for structured, unstructured, and semi-structured data types
Data size	Best suited for smaller amounts of data	Can be used for large amounts of data
Storage type	Can be used for on-premise or cloud storage	Optimised for cloud data warehouses
Latency	High, as transformations need to be completed before storing data	Low, as minimal processing is done before storing in the data warehouse
Flexibility	Low, as data sources and transformations need to be defined at the beginning of the process	High, as transformations need not be defined when integrating new sources
Scalability	Can be low, as the ETL tool should support scaling of operations	High, as ELT tools can be easily configured for changing data sources
Maintenance	May need continuous maintenance in case of changes in data sources or formats	Low maintenance required as usually ELT tools automate the process
Compliance with security protocols	Easy to implement	May need to be supported by data warehouse/ELT tool
Storage requirement	Low as only transformed data is stored	Can be high as raw data is stored

How ETL is not relevant to our project

Our project consists of data that is generated by the algorithm when it trains and hence there is not data loading or transformation that is being performed. Basically, the algorithm is self-generating the dataset on runtime.

Machine Learning

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. Recommendation engines are a common use case for machine learning. Other popular uses include fraud detection, spam filtering, malware threat detection, business process automation (BPA) and predictive maintenance.

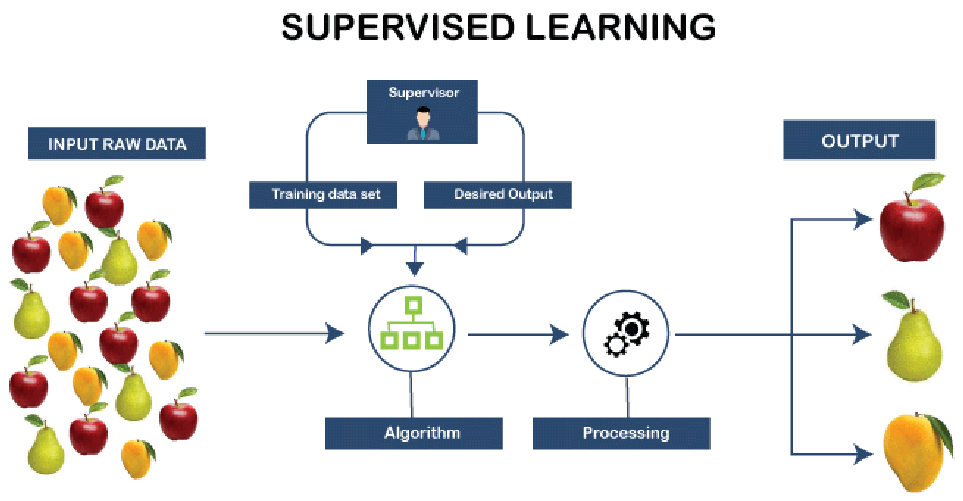
Machine learning is important because it gives enterprises a view of trends in customer behavior and business operational patterns, as well as supports the development of new products. Many of today's leading companies, such as Facebook, Google and Uber, make machine learning a central part of their operations. Machine learning has become a significant competitive differentiator for many companies.

Classical machine learning is often categorized by how an algorithm learns to become more accurate in its predictions. There are 3 types of ML Algorithms:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

The type of algorithm data scientists choose to use depends on what type of data they want to predict.

Supervised Learning:



The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

Y = f(X)

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

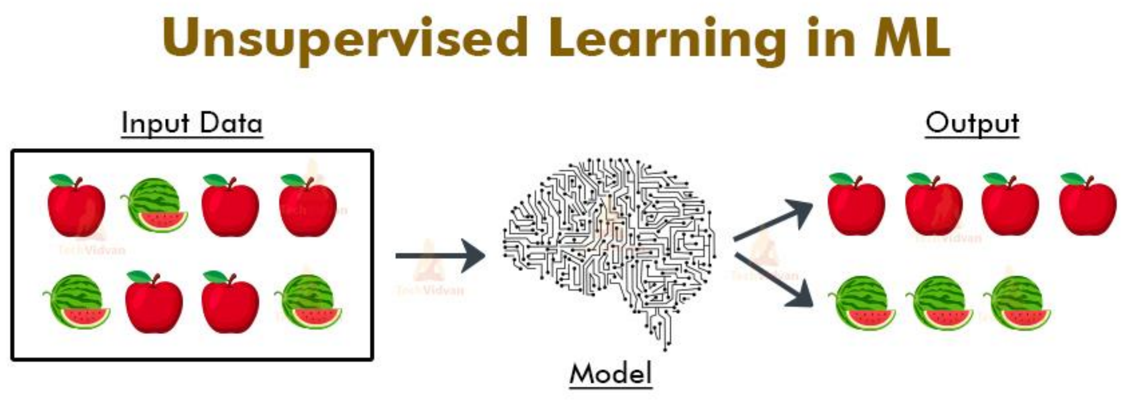
Supervised learning problems can be further grouped into regression and classification problems.

- Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Some common types of problems built on top of classification and regression include recommendation and time series prediction respectively. Some popular examples of supervised machine learning algorithms are:

- Linear regression for regression problems.
- Random forest for classification and regression problems.
- Support vector machines for classification problems

Unsupervised Learning:



Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

- Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are:

- k-means for clustering problems.
- Apriority algorithm for association rule learning problems.

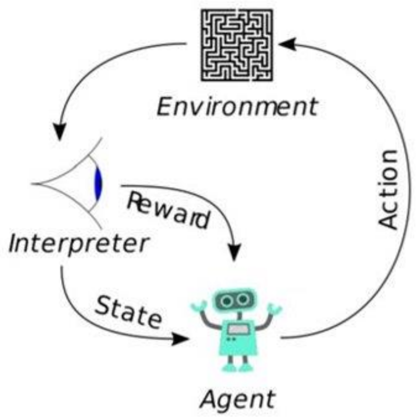
Supervised v/s Unsupervised Learning:

Parameters	Supervised machine learning technique	Unsupervised machine learning technique
Process	In a supervised learning model, input and output variables are given	In unsupervised learning model, only input data will be given
Input Data	Algorithms are trained using labeled data	Algorithms are used against data which is not labeled
Algorithms used	Support Vector Machine, Neural Network, Linear and logistic regression, Random Forest and Classification trees	Unsupervised algorithms can be divided into different categories: like Cluster algorithms, K-means, Hierarchical clustering etc.
Computational Complexity	Supervised learning is a simpler method	Unsupervised learning is computationally complex
Use of Data	Supervised learning model uses training data to learn a link between the input and the outputs	Unsupervised learning does not use output data
Accuracy of Results	Highly accurate and trustworthy method	Less accurate and trustworthy method
Real Time Learning	Learning method takes place offline	Learning method takes place in real time
Number of Classes	Number of classes is known	Number of classes is not known
Main Drawback	Classifying big data can be a real challenge in supervised learning	You cannot get precise information regarding data sorting, and the output as data used in unsupervised learning is not labeled and known

Reinforcement Learning:

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.

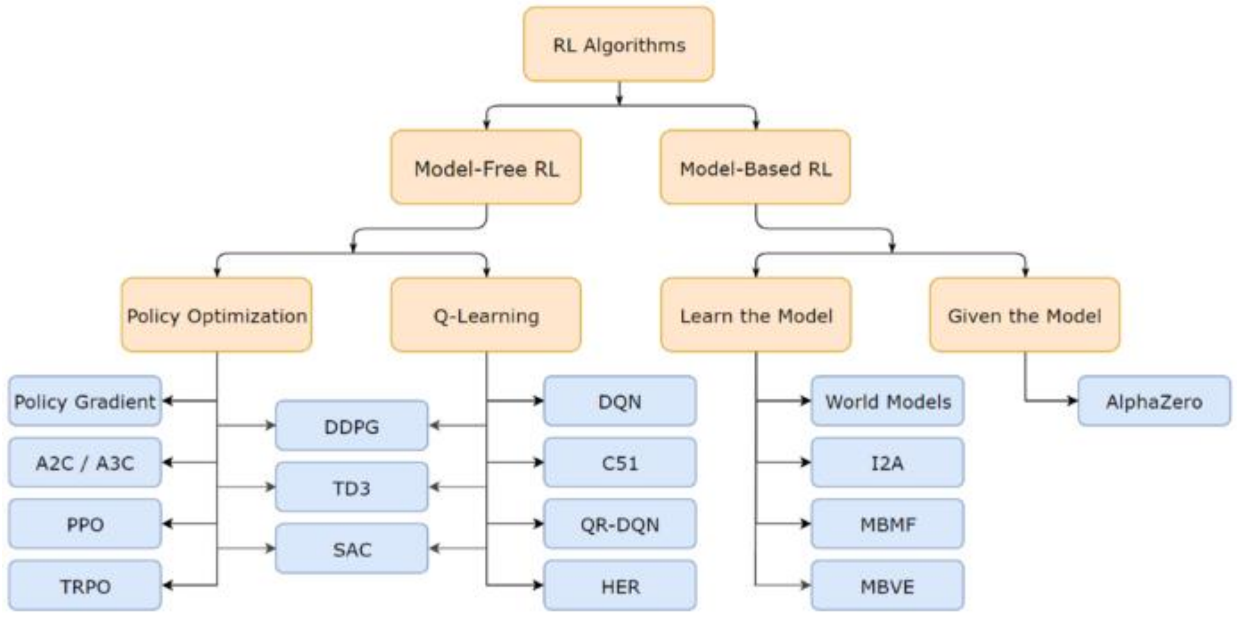
Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It's up to the model to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint machine's creativity. In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm is run on a sufficiently powerful computer infrastructure.



Terms in Reinforcement Learning:

- Agent: An entity that can perceive/explore the environment and act upon it.
- Environment: A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- Action: Actions are the moves taken by an agent within the environment.
- State: State is a situation returned by the environment after each action taken by the agent.
- Reward: A feedback returned to the agent from the environment to evaluate the action of the agent.
- Policy: Policy is a strategy applied by the agent for the next action based on the current state.
- Value: It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- Q-value: It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Types of Reinforcement Learning:



- **Model-based RL** uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Appropriate actions are then chosen by searching or planning in this world model.

- **Model-free RL**, on the other hand, uses experience to learn directly one or both of two simpler quantities (state/ action values or policies) which can achieve the same optimal behavior but without estimation or use of a world model. Given a policy, a state has a value, defined in terms of the future utility that is expected to accrue starting from that state.

Elements of Reinforcement Learning:

There are four main elements of Reinforcement Learning, which are given below:

- Policy: A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent.

- Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a reward signal. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.
- Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the immediate signal for each good and

bad action, whereas a value function specifies the good state and action for the future. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

- Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

How RL differs from other ML paradigms:

In supervised learning, the machine learns from training data. The training data consists of a labeled pair of inputs and outputs. So, we train the model (agent) using the training data in such a way that the model can generalize its learning to new unseen data. It is called supervised learning because the training data acts as a supervisor, since it has a labeled pair of inputs and outputs, and it guides the model in learning the given task.

Now, let's understand the difference between supervised and reinforcement learning with an example. Consider the dog analogy. In supervised learning, to teach the dog to catch a ball, we will teach it explicitly by specifying turn left, go right, move forward seven steps, catch the ball, and so on in the form of training data. But in RL, we just throw a ball, and every time the dog catches the ball, we give it a cookie (reward). So, the dog will learn to catch the ball while trying to maximize the cookies (reward) it can get.

Let's consider one more example. Say we want to train the model to play chess using supervised learning. In this case, we will have training data that includes all the moves a player can make in each state, along with labels indicating whether it is a good move or not. Then, we train the model to learn from this training data, whereas in the case of RL, our agent will not be given any sort of training data; instead, we just give a reward to the agent for each action it performs. Then, the agent will learn by interacting with the environment and, based on the reward it gets, it will choose its actions.

Similar to supervised learning, in unsupervised learning, we train the model (agent) based on the training data. But in the case of unsupervised learning, the training data does not contain any labels; that is, it consists of only inputs and not outputs. The goal of unsupervised learning is to determine hidden patterns in the input. There is a common misconception that RL is a kind of unsupervised learning, but it is not. In unsupervised learning, the model learns the hidden structure, whereas, in RL, the model learns by maximizing the reward.

For instance, consider a movie recommendation system. Say we want to recommend a new movie to the user. With unsupervised learning, the model (agent) will find movies similar to the movies the user (or users with a profile similar to the user) has viewed before and recommend new movies to the user.

With RL, the agent constantly receives feedback from the user. This feedback represents rewards (a reward could be ratings the user has given for a movie they have watched, time spent watching a movie, time spent watching trailers, and so on). Based on the rewards, an RL agent will understand the movie preference of the user and then suggest new movies accordingly.

Since the RL agent is learning with the aid of rewards, it can understand if the user's movie preference changes and suggest new movies according to the user's changed movie preference dynamically.

Thus, we can say that in both supervised and unsupervised learning the model (agent) learns based on the given training dataset, whereas in RL the agent learns by directly interacting with the environment. Thus, RL is essentially an interaction between the agent and its environment.

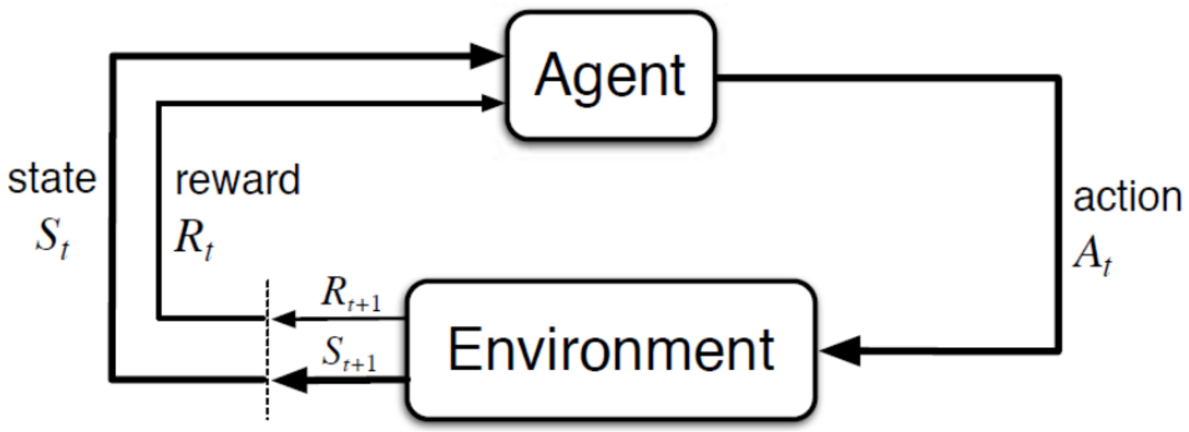
Applications of Reinforcement Learning:

RL has evolved rapidly over the past couple of years with a wide range of applications ranging from playing games to self-driving cars. One of the major reasons for this evolution is due to Deep Reinforcement Learning (DRL), which is a combination of RL and deep learning. Some real-life applications of RL:

- Manufacturing: In manufacturing, intelligent robots are trained using RL to place objects in the right position. The use of intelligent robots reduces labor costs and increases productivity.
- Dynamic pricing: One of the popular applications of RL is dynamic pricing. Dynamic pricing implies that we change the price of products based on demand and supply. We can train the RL agent for the dynamic pricing of products with the goal of maximizing revenue.
- Inventory management: RL is used extensively in inventory management, which is a crucial business activity. Some of these activities include supply chain management, demand forecasting, and handling several warehouse operations (such as placing products in warehouses to manage space efficiently).
- Recommendation system: RL is widely used in building a recommendation system where the behavior of the user constantly changes. For instance, in music recommendation systems, the behavior or the music preferences of the user changes from time to time. So, in those cases using an RL agent can be very useful as the agent constantly learns by interacting with the environment.
- Neural architecture search: In order for a neural network to perform a given task with good accuracy, the architecture of the network is very important, and it has to be properly designed. With RL, we can automate the process of complex neural architecture search by training the agent to find the best neural architecture for a given task with the goal of maximizing the accuracy.
- Natural Language Processing (NLP): With the increase in popularity of deep reinforcement algorithms, RL has been widely used in several NLP tasks, such as abstractive text summarization, chatbots, and more.
- Finance: RL is widely used in financial portfolio management, which is the process of constant redistribution of a fund into different financial products. RL is also used in predicting and trading in commercial transaction markets. JP Morgan has successfully used RL to provide better trade execution results for large orders.

Markov Decision Process:

Markov Decision Process (MDP) is a mathematical framework to describe an environment in reinforcement learning. The following figure shows agent-environment interaction in MDP:



More specifically, the agent and the environment interact at each discrete time step, t = 0, 1, 2, 3…At each time step, the agent gets information about the environment state St. Based on the environment state at instant t, the agent chooses an action At. In the following instant, the agent also receives a numerical reward signal Rt+1. This thus gives rise to a sequence like S0, A0, R1, S1, A1, R2…

The random variables Rt and St have well defined discrete probability distributions. These probability distributions are dependent only on the preceding state and action by virtue of Markov Property. Let S, A, and R be the sets of states, actions, and rewards. Then the probability that the values of St, Rt and At taking values s', r and a with previous state s is given by,

$$\mathrm{p}(s', r|s, a) = P\{S_t = s', R_t = r \,|S_{t-1} = s, A_{t-1} = a\}$$

The function p controls the dynamics of the process.

Q Learning:

Q-learning is a model-free reinforcement learning algorithm.

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

What's 'Q'?
The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

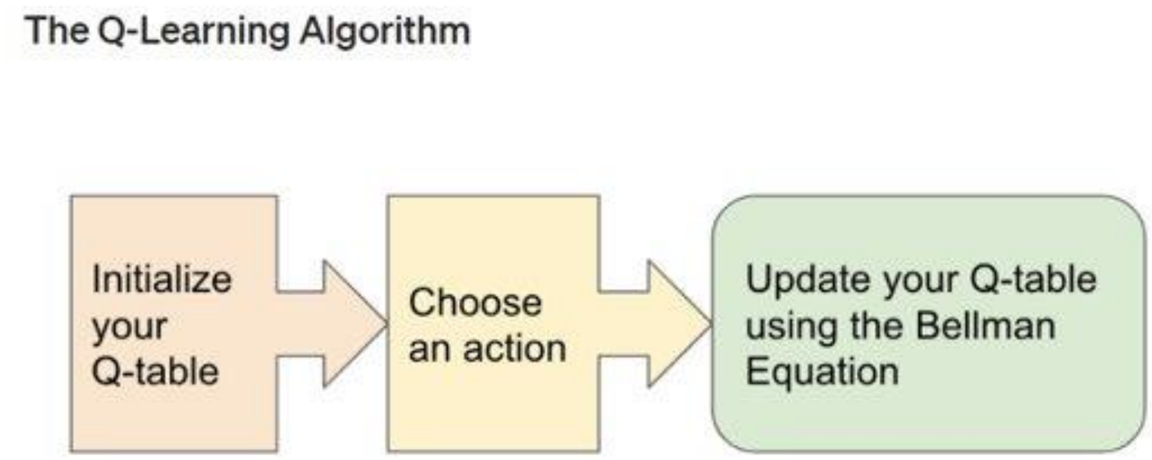


Figure 2: The Q-Learning Algorithm (Image by Author)

Implementation:
a. Creating a q table:
When q-learning is performed we create what's called a q-table or matrix that follows the shape of [state, action] and we initialize our values to zero. We then update and store our q-values after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value.

b. Q learning and making updates:
The next step is simply for the agent to interact with the environment and make updates to the state action pairs in our q-table Q [state, action].

An agent interacts with the environment in 1 of 2 ways (Exploiting or Exploration).

Exploitation:
The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as exploiting since we use the information, we have available to us to make a decision.
Exploration:
The second way to act is to act randomly. This is called exploring. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process.
You can balance exploration/exploitation using epsilon (ε) and setting the value of how often you want to explore vs exploit.

c. Updating the q-table

The updates occur after each step or action and ends when an episode is done. Done in this case means reaching some terminal point by the agent. A terminal state for example can be anything like landing on a checkout page, reaching the end of some game, completing some desired objective, etc. The agent will not learn much after a single episode, but eventually with enough exploring (steps and episodes) it will converge and learn the optimal q-values.

The Q-learning algorithm (which is nothing but a technique to solve the optimal policy problem) iteratively updates the Q-values for each state-action pair using the Bellman Optimality Equation until the Q-function (Action-Value function) converges to the optimal Q-function, q∗. This process is called Value-Iteration.

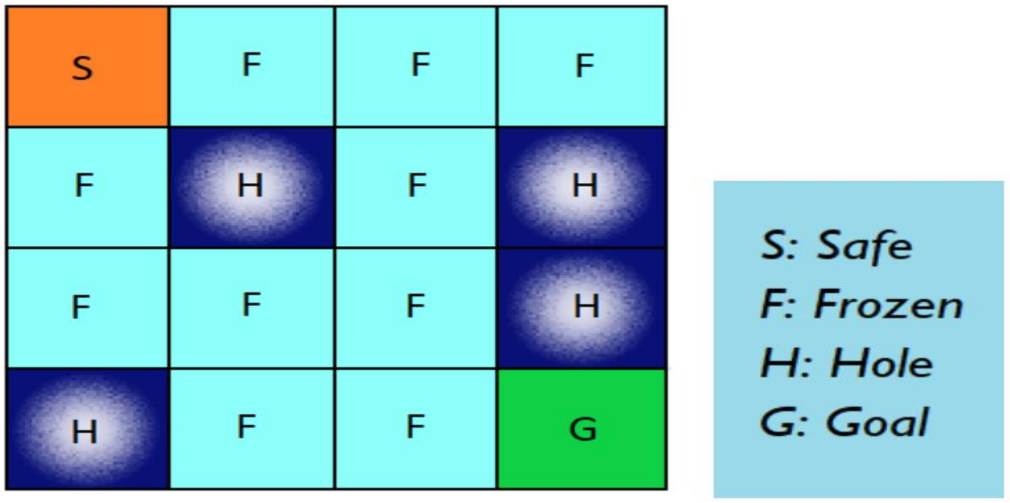
To make the Q-value eventually converge to an optimal Q-value q∗, what we have to do is —for the given state-action pair, we have to make the Q-value as near as we can to the right-hand side of the Bellman Optimality Equation. For this, the loss between the Q-value and the optimal Q-value for the given state-action pair will be compared iteratively, and then each time we encounter the same state-action pair, we will update the Q-value, again and again, to reduce the loss. The loss can be given as q∗ (s, a)-q (s, a).

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

We cannot overwrite the newly computed Q-value with the older value. Instead, we use a parameter called learning rate denoted by α, to determine how much information of the previously computed Q-value for the given state-action pair we retain over the newly computed Q-value calculated for the same state-action pair at a later time step. The higher the learning rate, the more quickly the agent will adopt the newly computed Q-value. Therefore, we need to take care of the trade-off between new and old Q-value using the appropriate learning rate.
Gamma is the discount factor which causes rewards to lose value over time so more immediate rewards are valued more highly.

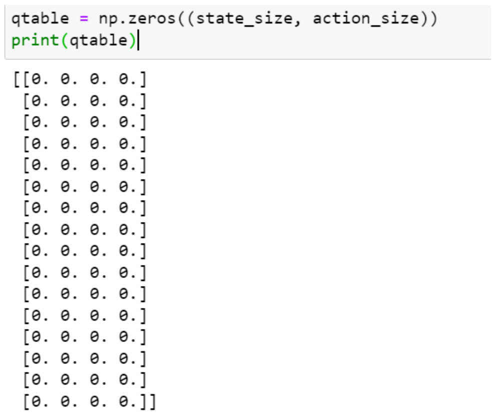
Frozen Lake Problem:

The Frozen Lake environment is a 4×4 grid which contain four possible areas — Safe (S), Frozen (F), Hole (H) and Goal (G). The agent moves around the grid until it reaches the goal or the hole. If it falls into the hole, it has to start from the beginning and is rewarded the value 0. The process continues until it learns from every mistake and reaches the goal eventually. Here is visual description of the Frozen Lake grid (4×4):



This grid has 16 possible blocks where the agent will be at a given time. At the current state, the agent will have four possibilities of deciding the next state. From the current state, the agent can move in four directions as mentioned which gives us a total of 16×4 possibilities. These are our weights, and we will update them according to the movement of our agent.

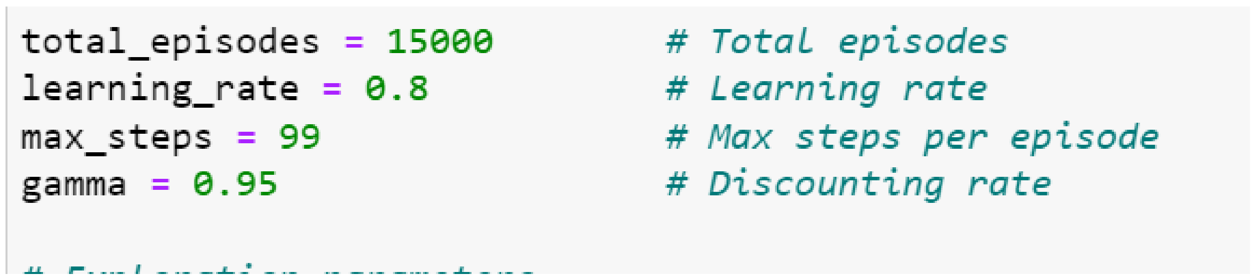
So first we will initialize the q Table values to all zeroes.



```
qtable = np.zeros((state_size, action_size))
print(qtable)
```

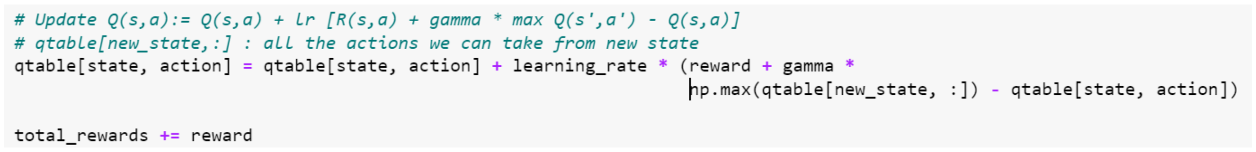
```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

The next steps are to define the total number of episodes we want to run the model, maximum number of steps in each episode, the learning rate and the gamma value.



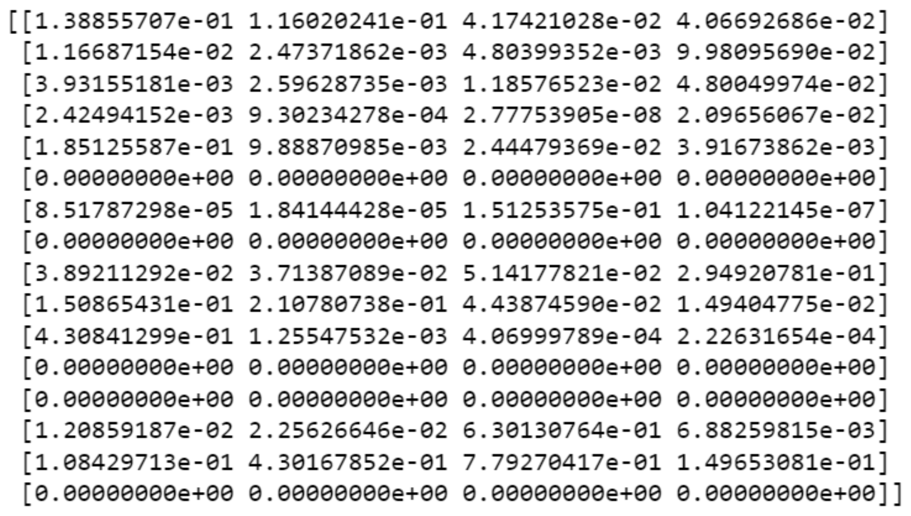
```
total_episodes = 15000        # Total episodes
learning_rate = 0.8           # Learning rate
max_steps = 99                # Max steps per episode
gamma = 0.95                  # Discounting rate
```

The model will run for 15000 episodes and will update the Q values in the table based upon the Bellman Equation and then the agent will identify the actions to be taken based on the maximum q value in each state from the table.



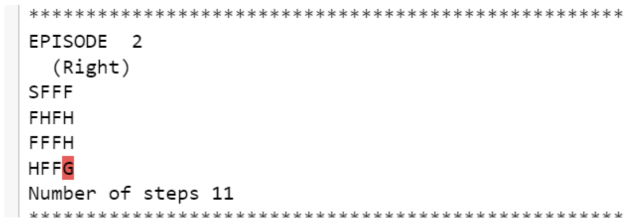
```
# Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
# qtable[new_state,:] : all the actions we can take from new state
qtable[state, action] = qtable[state, action] + learning_rate * (reward + gamma *
                                                    np.max(qtable[new_state, :]) - qtable[state, action])

total_rewards += reward
```

Updated Q Table:



```
[[1.38855707e-01 1.16020241e-01 4.17421028e-02 4.06692686e-02]
 [1.16687154e-02 2.47371862e-03 4.80399352e-03 9.98095690e-02]
 [3.93155181e-03 2.59628735e-03 1.18576523e-02 4.80049974e-02]
 [2.42494152e-03 9.30234278e-04 2.77753905e-08 2.09656067e-02]
 [1.85125587e-01 9.88870985e-03 2.44479369e-02 3.91673862e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [8.51787298e-05 1.84144428e-05 1.51253575e-01 1.04122145e-07]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [3.89211292e-02 3.71387089e-02 5.14177821e-02 2.94920781e-01]
 [1.50865431e-01 2.10780738e-01 4.43874590e-02 1.49404775e-02]
 [4.30841299e-01 1.25547532e-03 4.06999789e-04 2.22631654e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.20859187e-02 2.25626646e-02 6.30130764e-01 6.88259815e-03]
 [1.08429713e-01 4.30167852e-01 7.79270417e-01 1.49653081e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

This is the optimal q table we found while running for 5 episodes and the final state looks like:



```
****************************************************
EPISODE  2
  (Right)
SFFF
FHFH
FFFH
HFFG
Number of steps 11
****************************************************
```

Disadvantages of Q Learning:

1. The amount of memory required to save and update that table would increase as the number of states increases.
2. The amount of time required to explore each state to create the required Q-table would be unrealistic.

Deep Q Learning:

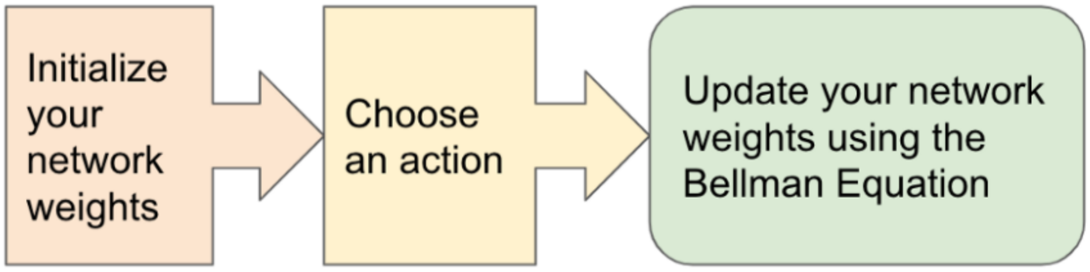
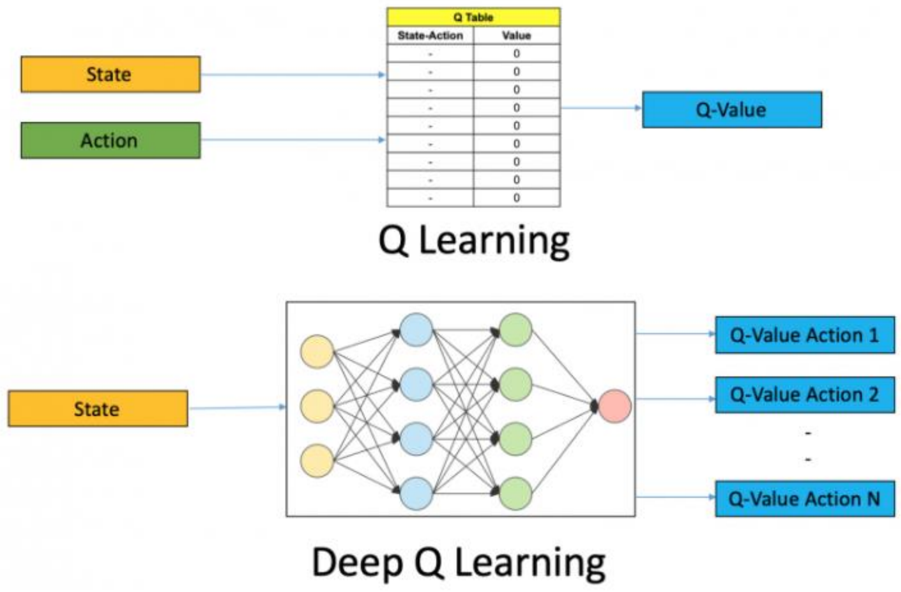

Initialize your network weights
Choose an action
Update your network weights using the Bellman Equation

Figure 5: The Deep Q-Network Algorithm (Image by Author)

In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning & deep Q-learning is wonderfully illustrated below:



The steps involved in reinforcement learning using deep Q-learning networks (DQNs)?

- All the past experience is stored by the user in memory
- The next action is determined by the maximum output of the Q-network

- The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q*. This is basically a regression problem. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem. Going back to the Q-value update equation derived from the Bellman equation. we have:

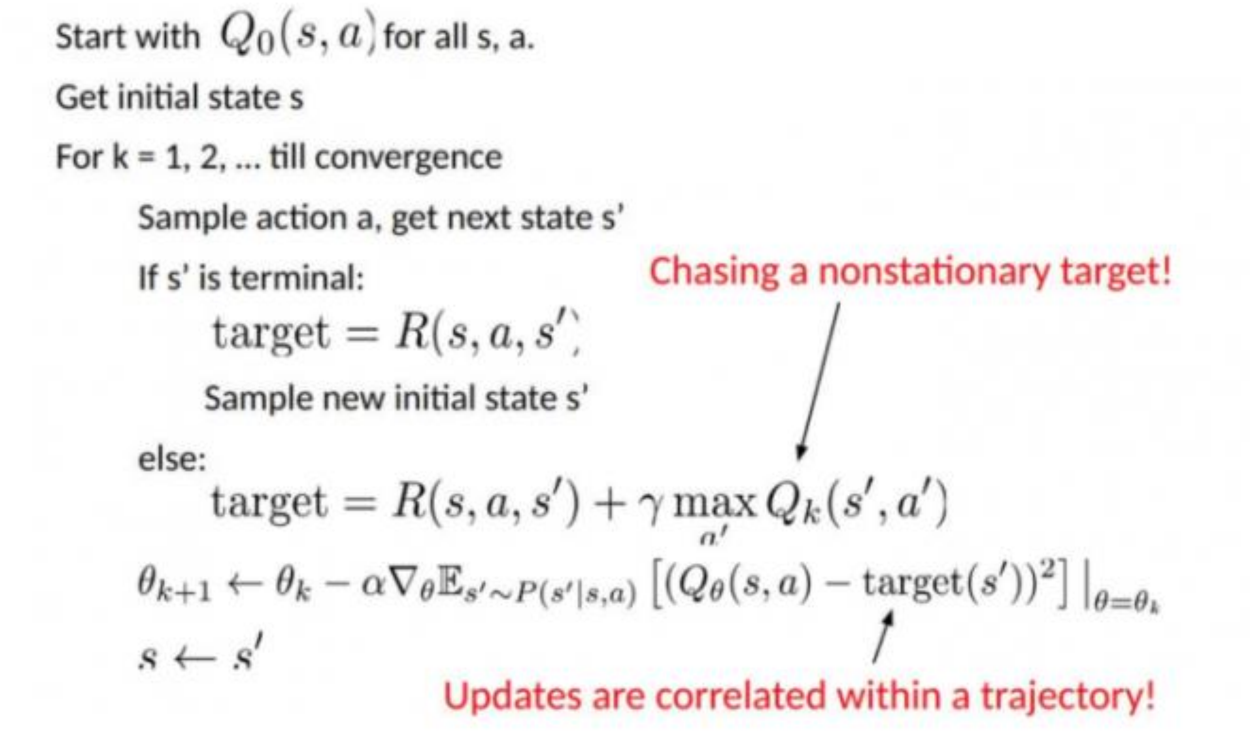
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[\boxed{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)} - Q(S_t, A_t)\right]$$

The section in green represents the target. We can argue that it is predicting its own value, but since R is the unbiased true reward, the network is going to update its gradient using backpropagation to finally converge.

Challenges in Deep RL as Compared to Deep Learning

We understood how neural networks can help the agent learn the best actions. However, there is a challenge when we compare deep RL to deep learning (DL):

Non-stationary or unstable target:



```
Start with $Q_0(s, a)$ for all s, a.
Get initial state s
For k = 1, 2, ... till convergence
    Sample action a, get next state s'
    If s' is terminal:
        $\text{target} = R(s, a, s')$
        Sample new initial state s'
    else:
        $\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
    $\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[(Q_\theta(s, a) - \text{target}(s'))^2\right] \big|_{\theta=\theta_k}$
    $s \leftarrow s'$
```

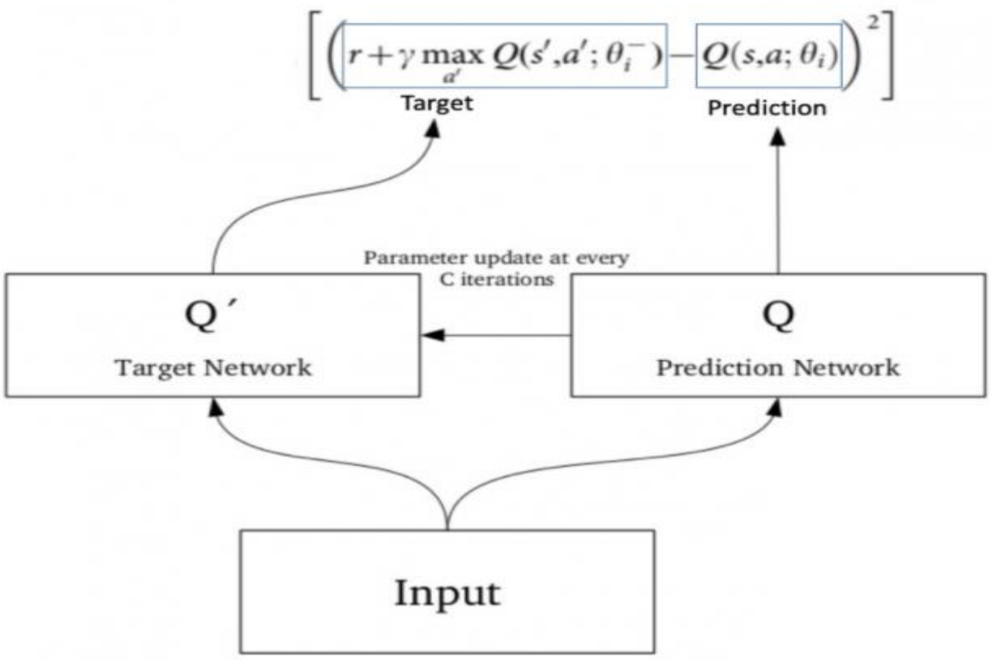
Chasing a nonstationary target!
Updates are correlated within a trajectory!

As you can see in the above code, the target is continuously changing with each iteration. In deep learning, the target variable does not change and hence the training is stable, which is just not true for RL. As we play out the game, we get to know more about the ground truth values of states and actions and hence, the output is also changing. So, we try to learn to map for a constantly changing input and output.

1. Target Network

Since the same network is calculating the predicted value and the target value, there could be a lot of divergence between these two. So, instead of using 1one neural network for learning, we can use two.

We could use a separate network to estimate the target. This target network has the same architecture as the function approximator but with frozen parameters. For every C iteration (a hyperparameter), the parameters from the prediction network are copied to the target network. This leads to more stable training because it keeps the target function fixed (for a while):



$\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$
Target
Prediction
Parameter update at every C iterations
Q´
Target Network
Q
Prediction Network
Input

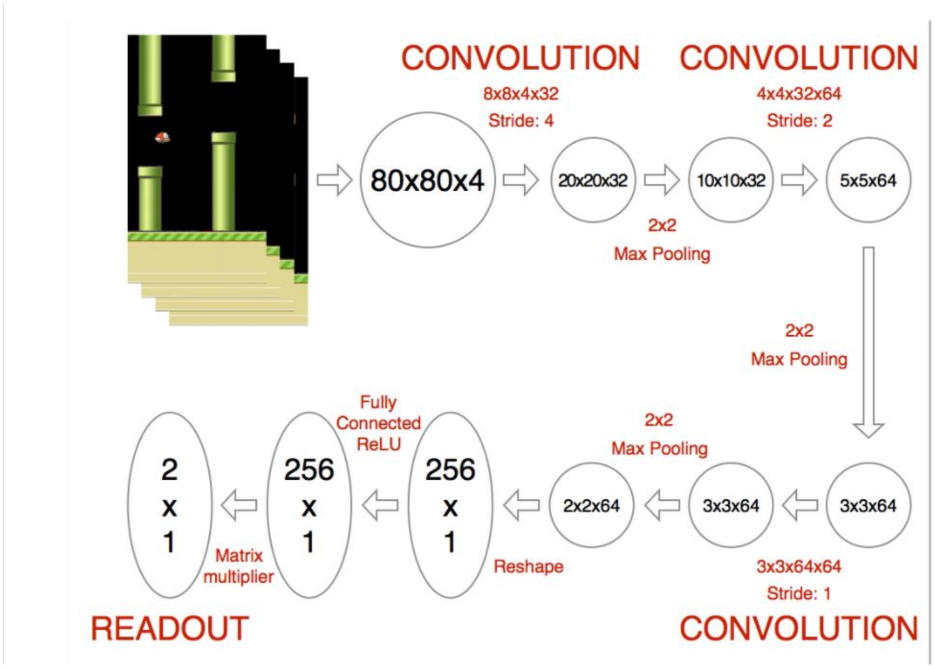
2. Experience Replay

To perform experience replay, we store the agent's experiences – et= (st, at, rt, st+1)

What does the above statement mean? Instead of running Q-learning on state/action pairs as they occur during simulation or the actual experience, the system stores the data discovered for [state, action, reward, next state] – in a large table.

Flappy Bird: Network Architecture and Training

The architecture of the network is shown in the figure below. The first layer convolves the input image with an 8x8x4x32 kernel at a stride size of 4. The output is then put through a 2x2 max pooling layer. The second layer convolves with a 4x4x32x64 kernel at a stride of 2. We then max pool again. The third layer convolves with a 3x3x64x64 kernel at a stride of 1. We then max pool one more time. The last hidden layer consists of 256 fully connected Reu nodes.



The final output layer has the same dimensionality as the number of valid actions which can be performed in the game, where the 0th index always corresponds to doing nothing. The values at this output layer represent the Q function given the input state for each valid action. At each time step, the network performs whichever action corresponds to the highest Q value using a ϵ greedy policy.

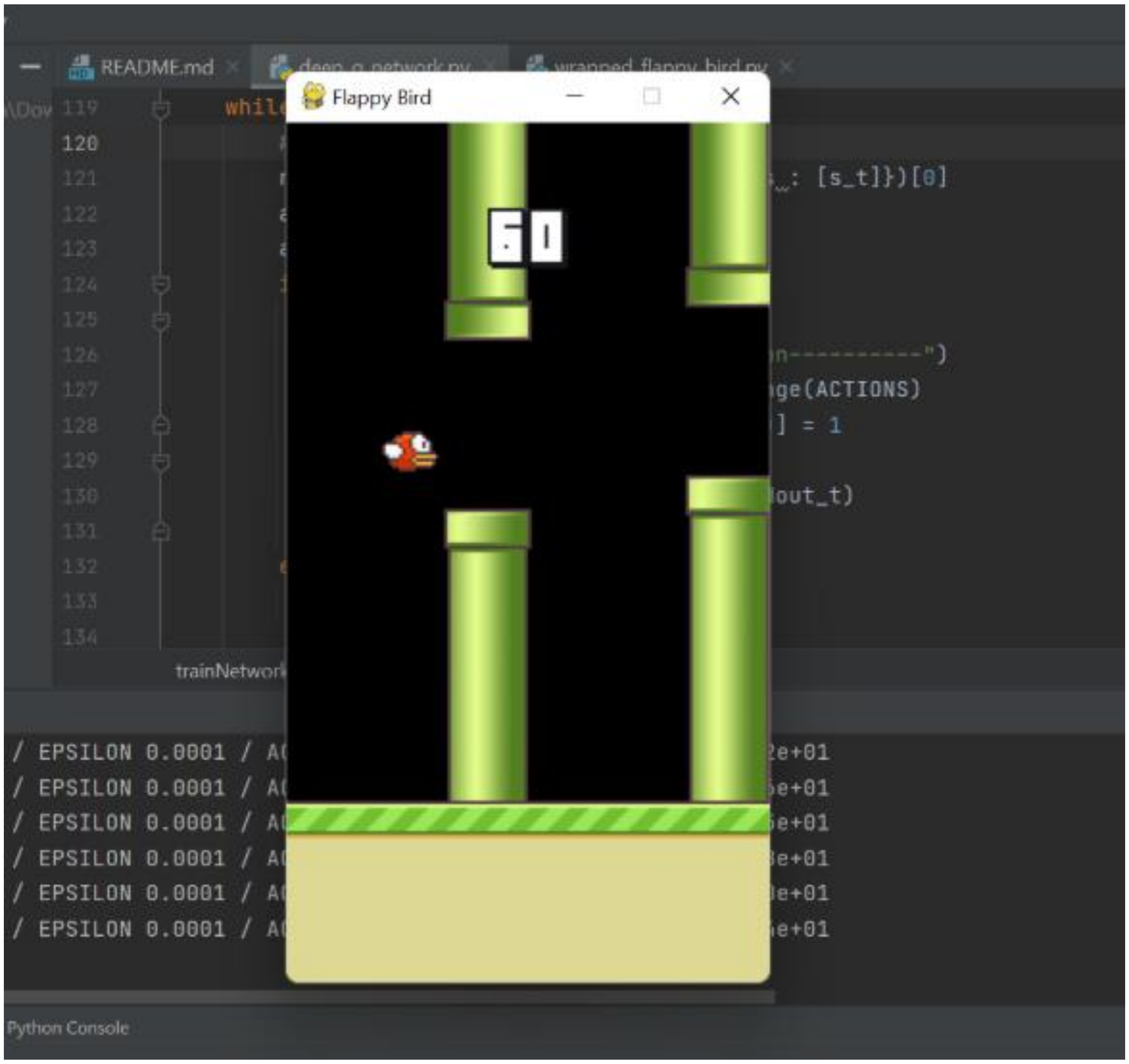
Training

At first, we initialize all weight matrices randomly using a normal distribution with a standard deviation of 0.01, then set the replay memory with a max size of 500,00 experiences.

we start training by choosing actions uniformly at random for the first 10,000-time steps, without updating the network weights. This allows the system to populate the replay memory before training begins.

Note that unlike [1], which initialize ϵ = 1, we linearly anneal ϵ from 0.1 to 0.0001 over the course of the next 3000,000 frames. The reason why we set it this way is that agent can choose an action every 0.03s (FPS=30) in our game, high ϵ will make it flap too much and thus keeps itself at the top of the game screen and finally bump the pipe in a clumsy way. This condition will make Q function converge relatively slow since it only starts to look other conditions when ϵ is low. However, in other games, initialize ϵ to 1 is more reasonable.

During training time, at each time step, the network samples minibatches of size 32 from the replay memory to train on and performs a gradient step on the loss function described above using the Adam optimization algorithm with a learning rate of 0.000001. After annealing finishes, the network continues to train indefinitely, with ϵ fixed at 0.001.



```
        whil
            #
            r                                   : [s_t]})[0]
            a
            a
            i

                                               n----------")
                                               ge(ACTIONS)
                                               ] = 1

                                               out_t)


            e
```

```
/ EPSILON 0.0001 / A                           2e+01
/ EPSILON 0.0001 / A                           6e+01
/ EPSILON 0.0001 / A                           6e+01
/ EPSILON 0.0001 / A                           3e+01
/ EPSILON 0.0001 / A                           3e+01
/ EPSILON 0.0001 / A                           6e+01
```

Architecture

There are 2 approaches to our implementation:

1. On-prem: It involves the use of local storage for data storage and PyCharm for running python as well as utilizing the system's Intel processor for the training environment

2. Clous-based: This architecture is hosted on the AWS cloud. Here, Amazon S3 storage will be used for storing the data, Amazon PageMaker will be used as an IDE for running algorithms and the training environment will be Amazon EC2.

Below are some of the AWS services that we will be utilizing:

Amazon EC2: Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.

Amazon S3: Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

Amazon sage Maker: Amazon SageMaker is a cloud machine-learning platform that was launched in November 2017. PageMaker enables developers to create, train, and deploy machine-learning (ML) models in the cloud. SageMaker also enables developers to deploy ML models on embedded systems and edge-devices.

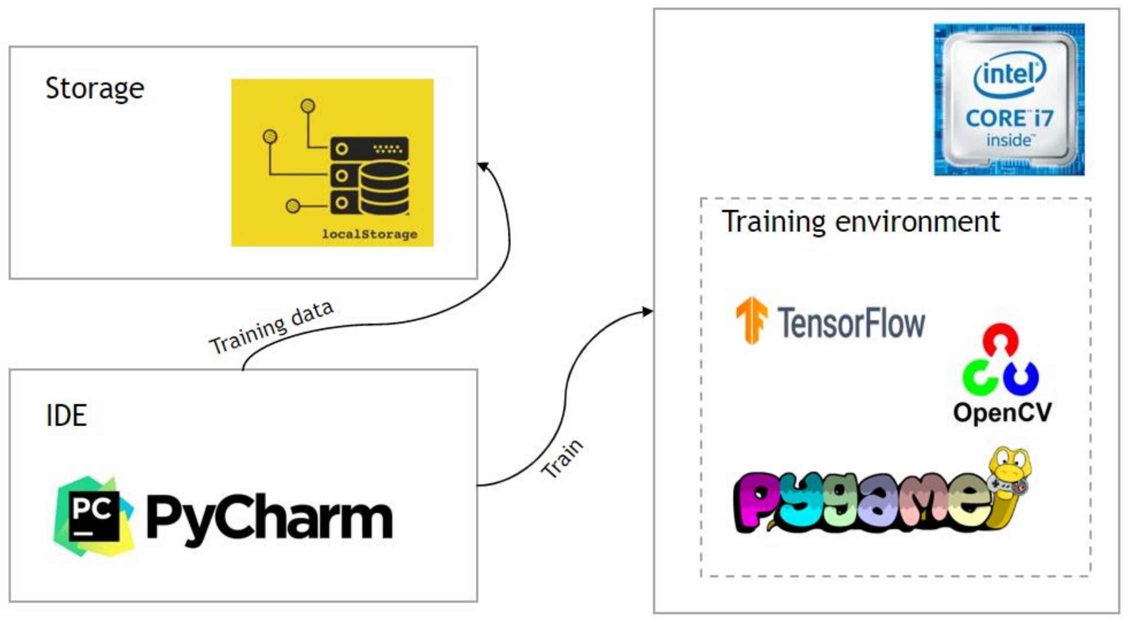
In addition to the above AWS services, we are also utilizing below modules for our project:

Python pygame module: pygame is a Python wrapper for the SDL library, which stands for Simple DirectMedia Layer. SDL provides cross-platform access to your system's underlying multimedia hardware components, such as sound, video, mouse, keyboard, and joystick. pygame started life as a replacement for the stalled PySDL project. The cross-platform nature of both SDL and pygame means you can write games and rich multimedia Python programs for every platform that supports them. To install pygame on your platform, use the appropriate pip command: $ pip install pygame

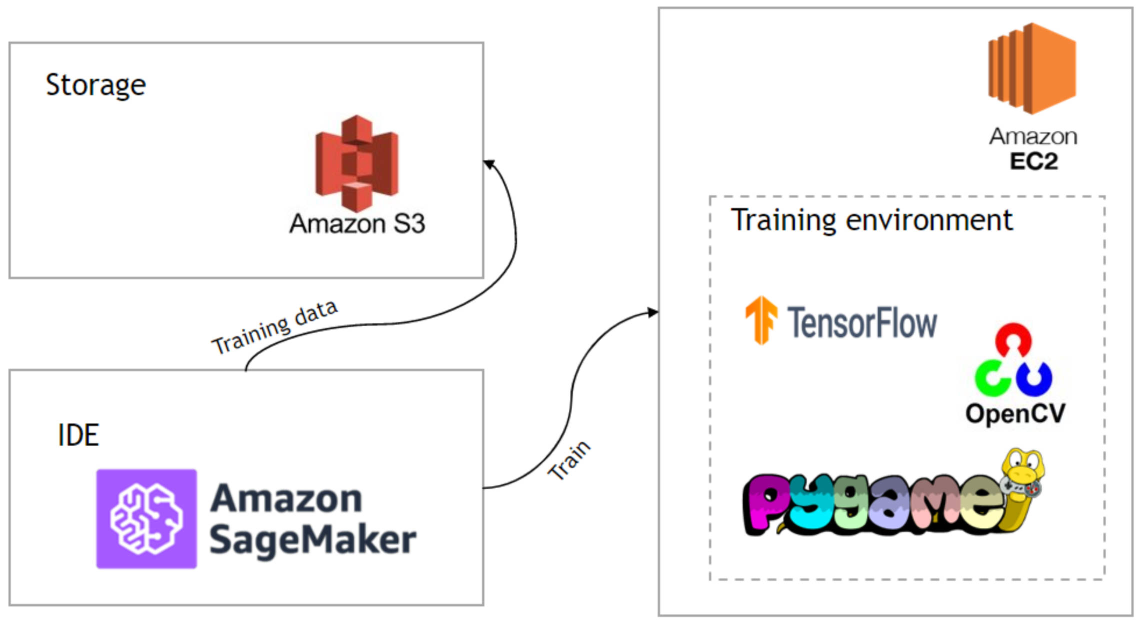
OpenCV: OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV

TensorFlow: TensorFlow is an open-source end-to-end platform for creating Machine Learning applications. It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks. It allows developers to create machine learning applications using various tools, libraries, and community resources.

On-prem architecture:



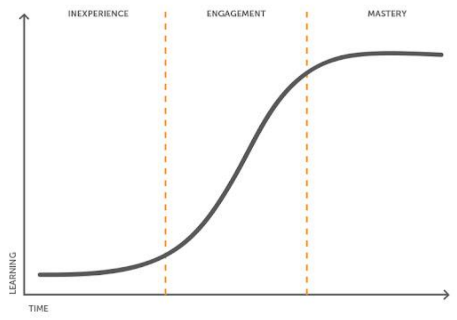
Cloud-based architecture:



Challenges

Time to train: Considering the fact that RL models start from a child's level of knowledge where it is just simply exploring the environment, the actions, and the consequences (these are the rewards and penalties in RL) we can see that getting to an "adult" level of control takes time. We have trained one model that was trained with XX million steps, that is YY episodes (game reached game over YY times), which is still not considered enough if we consider that the DeepMind research mentions 50 million training steps for mastering most Atari games.

Cost of training: The other issue that arises from using cloud infrastructure is the pricing. Although, cloud services are considered cheaper than running your own infrastructure, maintaining it, etc., for testing purposes training RL models is expensive. The SageMaker RL estimators can be configured to train only on more powerful instances like the ml.m4.4xlarge (16 vCPU, 64 GB RAM) which restricts the users to use cheaper instances.



Learning curve: Reinforcement learning requires a considerable amount of time to learn and understand the various concepts associated with it. As we had implemented supervised and unsupervised algorithms in our previous classes, this course gave us an opportunity to explore the domain of RL and gain practical exposure by implementing the same via a project use-case. Every team member did a thorough research before making the proposal and project paper.

Results & Learnings

- Our game of Flappy bird was able to achieve the score of 1147 which beats the highest human score of 999
- This was possible only because of the training time that we spent on the RL algorithm which was around 20-25 hours
- This course provided our team with an opportunity to learn about the field of Reinforcement Learning and how it can be applied to the real world use cases like Flappy bird game and frozen lake problem
- Another key learning that we achieved from this project was the differentiation between model based and model free RL. In addition to this, we gathered a deep understanding on the Q-learning and Deep Q-learning algorithms which are the most widely used algorithms for the implementation of Reinforcement Learning.
- Our team was able to also study the base concepts and the theories behind RL algorithms which include Markov Decision Process (MDP) and Bellman Equation
- We also learned about how RL differs from Supervised and Unsupervised learning and also RL implementation areas in the field of medicine, finance, trading and robotics
- From the classes, we were able to understand the different aspects of an analytics project in the industry and how to approach any data science project in a consultative manner
- Demonstration of projects by other teams also provided us with an opportunity to understand different areas of data science like image recognition, neural networks, natural language processing and machine learning
- To conclude, this project based course provided us a pathway to hone our skills as a Data Scientist and understand the rapidly evolving field of Reinforcement Learning

References

- https://medium.com/hengky-sanjaya-blog/supervised-vs-unsupervised-learning-aae0eb8c4878
- https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc’
- http://cs229.stanford.edu/proj2015/362_report.pdf
- https://aws.amazon.com/