Playing Atari with Deep Reinforcement Learning

An Introduction to the World of Reinforcement Learning

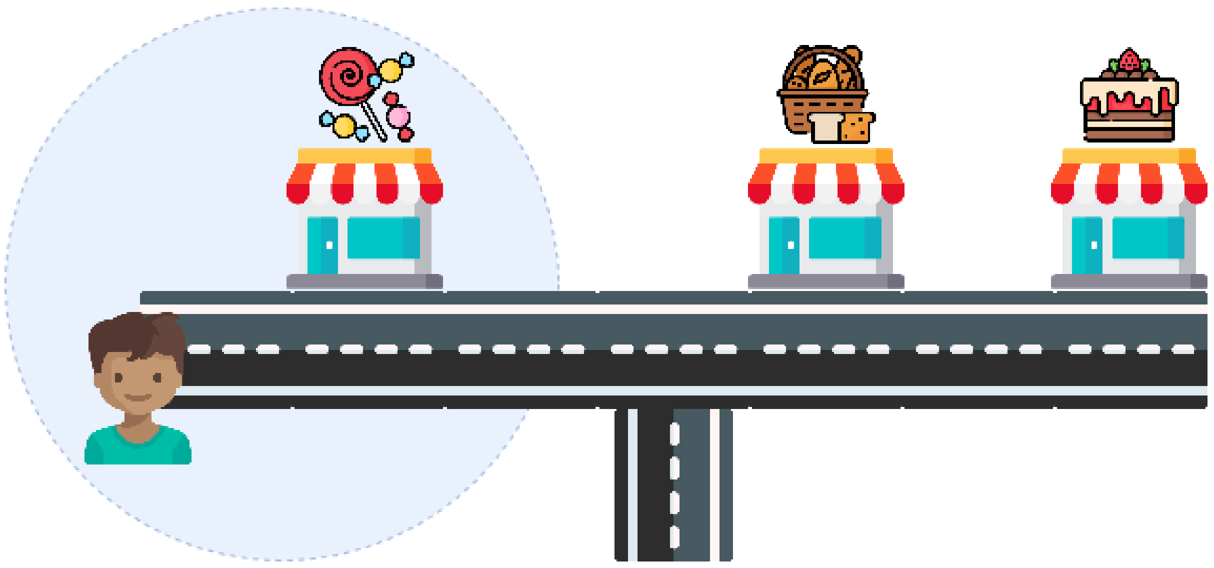
SRIN
Tutorial #01

Chrystian, 21 November 2022

Contents

Reinforcement Learning

The world is infinitely complex. The question is how can we comprehend the complicated details in our surrounding environment? Luckily for us, Reinforcement Learning can be a way to describe, generalize, and model our understanding by exploring the environment and learning the best way to acquire our desired target/ rewards.

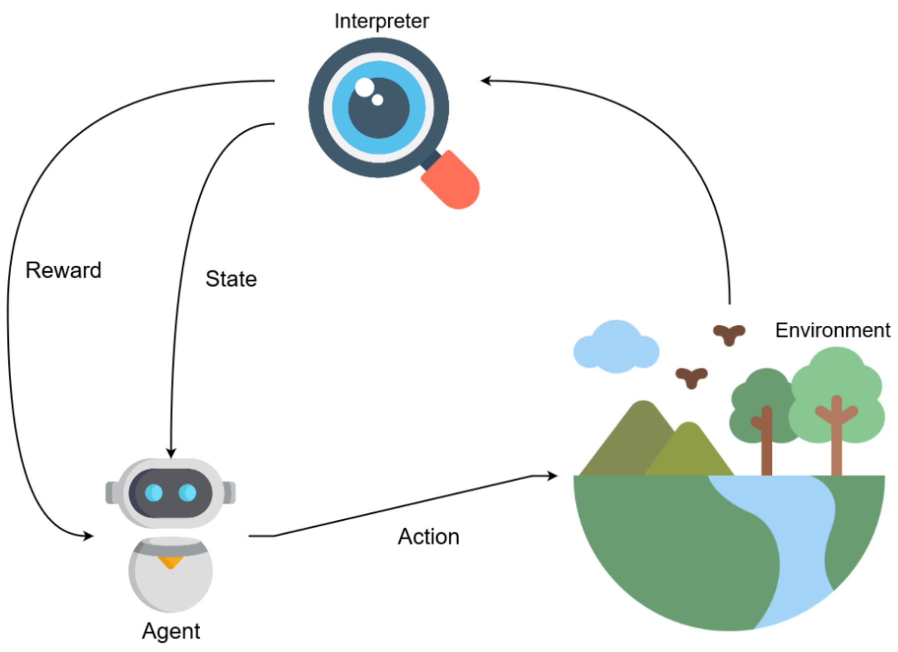


Let's say Joe love to eat desserts every day and wanted to buy some snack. After some time, he found a candy shop near his house. With the target to eat the best desserts, what should be his future action to fully maximize his target?

Normally, sometimes what we do is explore finding things outside of our knowledge. However, not most of the time, and depending on the new information, we then make a decision whether the action is worth the reward. This is roughly how reinforcement learning work.

<u>Exploitation vs Exploration</u>

Reinforcement Learning (RL) is a study of how intelligent agents ought to take action in an environment in order to maximize total rewards. RL focus on finding the best balance between **exploration** (unknown territory) and **exploitation** (current knowledge).



Terminology

Before continuing further. A simple case of Joe, with his quests on finding sweets, we can say that we have:

Agent: Joe

State: Joe current position on the road

Action: Walking to different street

Environment: Shops on streets

Reward: Desserts sold at the shops

Interpreter: Algorithm used

Q-learning

Reinforcement Learning is a big field with many unique methods and approaches to solve agent environment reward maximation problems. One most intuitive that we are interested in is Q-learning, an off-policy, model-free, value function algorithm.

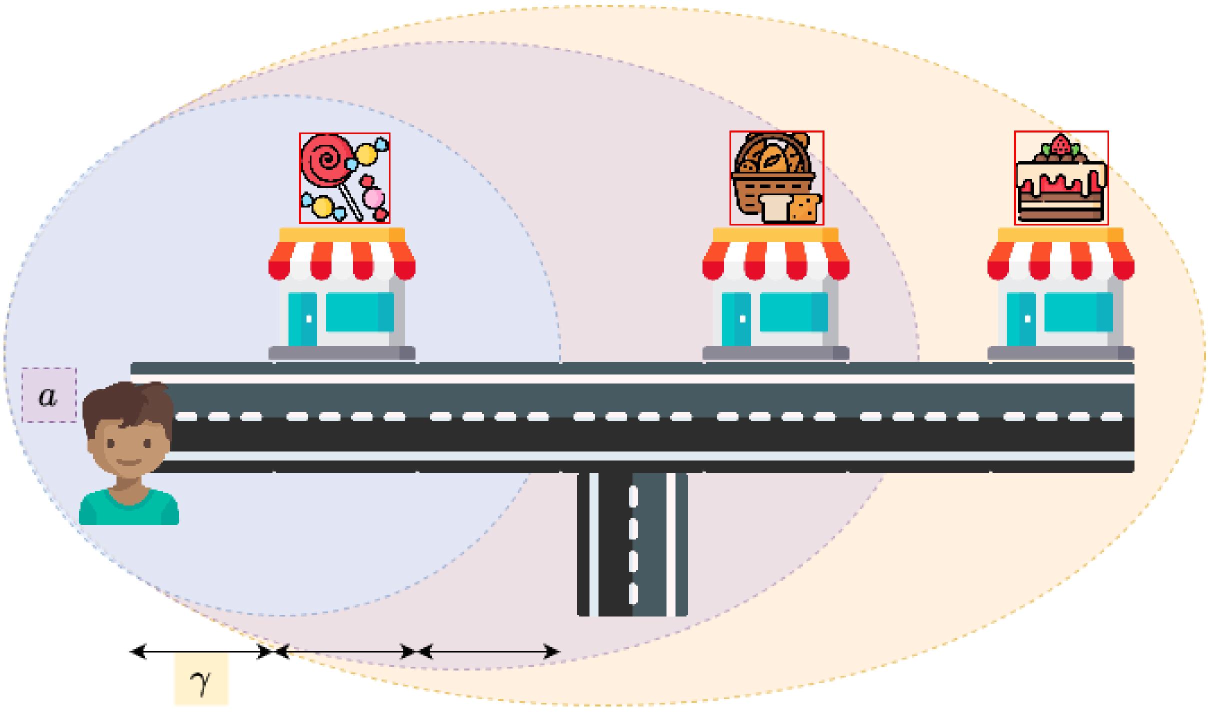
First, let's break down,

An off-policy means the value function is learned based on taking different random stochastic actions outside of the known value.

Model-free, as it does not require a model of the environment, thus making it flexible and can handle stochastic environments without requiring adaptations.

Lastly, Q-Learning is a value function algorithm, as it tries to maximize a function below:

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



The function, in essence, took into account the current state (Joe's location) and the action (walking to a different location) and then gives out a Q value that can be used to determine how good the certain action is at that particular state. When training these Q-values will be updated iteratively while the agent explores.

Variables that effect Q-Learning

Rewards $\boldsymbol{r}$

Reward is a tricky subject but, in a nutshell, it is an incentive mechanism that tells the agent what is correct and what is wrong using + score as reward and – negative as punishment.

In Joe case, reward can be looked at the quality of desserts he gets. Example, Joe might like cake more than candy, lastly pastry.

Delaying future rewards with Discount Factor $\boldsymbol{\gamma}$

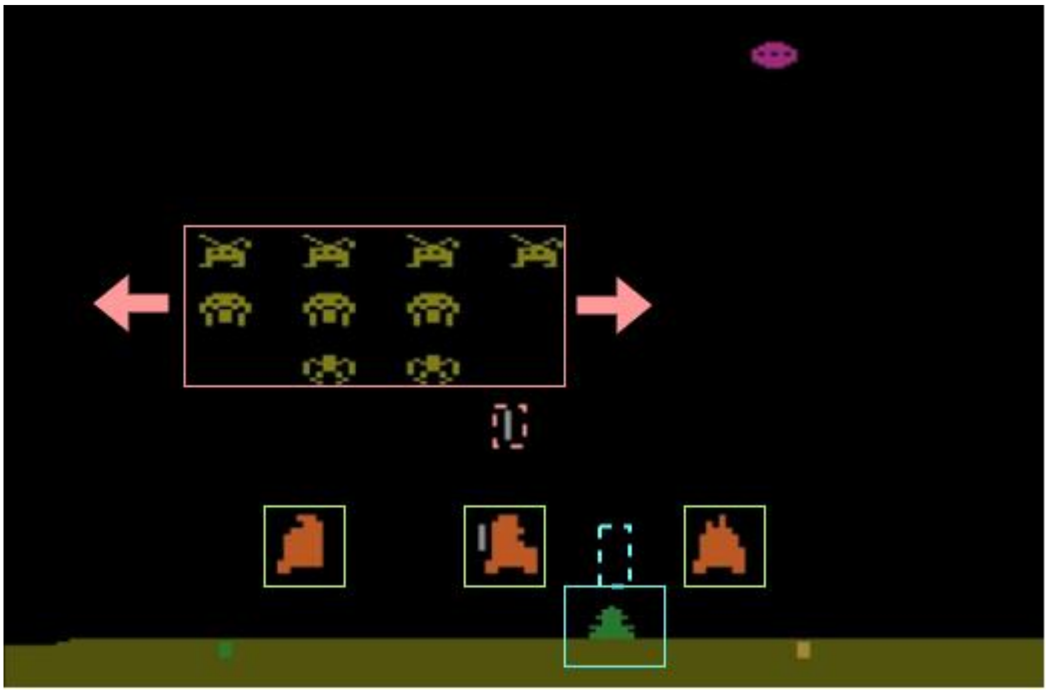
The core idea of Q-Learning is how an agent learns and takes into account future rewards. To control the effect of the possible future reward Q-Learning has a discount factor. This discount factor ranging from 0 to 1, controls how future rewards may affect agent behavior.

For example, in Joe's case, a factor that needs to be taken into account is what mode of transportation he is using. If he could only walk, farther far away from shops rewards are much more difficult to affect him. But if he has a motorcycle, even if the shops are further away the reward still may affect the agent.

In extreme cases where the discount factor is 0, the agent does not think about the future reward at all and only considers current available rewards. If the discount factor nears 1 then the model strives for long-term maximum reward. Keep in mind if the discount factor is 1 or above, the model may not converge or finish.

Unfortunately, Q-Learning has a limitation as it is designed only for quantifiable discrete actions and states. Additionally, with a complex continuous environment, Q-Learning is a challenge.

Atari Games Are Difficult Environment


Atari 2600 Space Invaders by NML32 on Youtube

While the idea of Q-Learning is straightforward. Real-world or even a retro old video game from 1980 has a complex state and action. As an example, above a simple Space Invaders game in that environment exists:

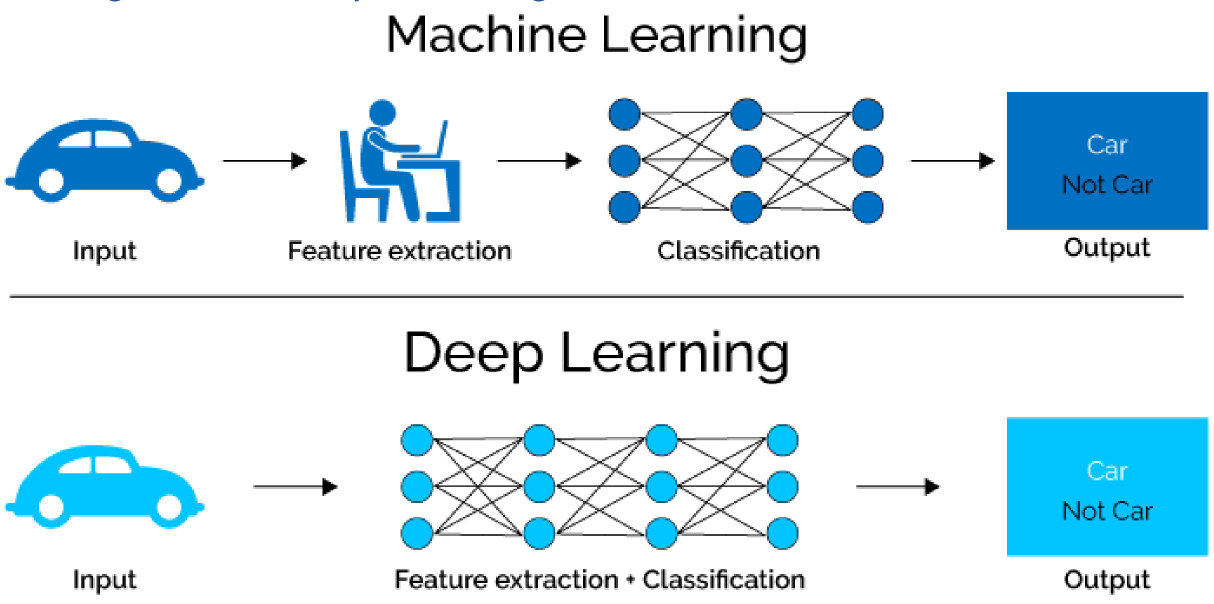
Red Box – a different type of enemy slowly moving down and periodically moving side to side shooting laser below, shown by red dashed box.

Blue Box – our agent needs to avoid the laser shown by the alien and simultaneously need to shoot down all the alien by shooting the laser up, the location shown by the blue dashed box.

Yellow box – environment change, where it degrades for every laser that it got hit. After enough hits, the box will be gone and the player needs to take into account this environment changes.

Not to mention this is only one out of many playable old Atari games. Then how can we use reinforcement learning to continuous and ever-changing games?

Emergence of Deep Learning

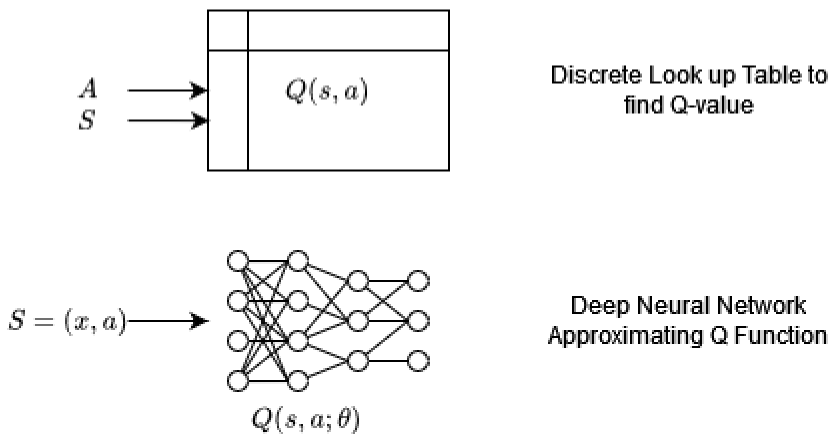


Machine Learning vs Deep Learning approach. Image from Towards Data Science

Previous research mostly uses handmade manual feature extraction methods such as Sarsa. However, this method is in itself inherently inefficient as in each game, the researcher needs to recalibrate and find the best feature.

In parallel, a recent breakthrough in Convolutional Neural Network makes it possible to do feature extraction convolutionally. This not only skips the feature extraction process entirely; this also makes the model much more flexible. The question now becomes, can we implement deep neural networks to automate feature extraction to RL and approximate Q-learning?

Deep Learning + Q-learning = Deep Q-Learning Network


A
S
$Q(s,a)$
Discrete Look up Table to find Q-value
$S=(x,a)$
$Q(s,a;\theta)$
Deep Neural Network Approximating Q Function

Remember in previous chapter where Q value function used to be discrete? The brilliance on “Playing Atari with Deep Reinforcement Learning” is that it successfully learns policies **directly from high-dimensional sensory input.** With Convolutional Neural Network (CNN) as its backbone, with image input x , the designed Deep Q-learning Network (DQN) were able to play seven games without adjusting the architecture or learning algorithm making it the most versatile model at the time.

The key concept of the paper, we know neural network is amazing at approximating complex function. Rather than using conventional Q-learning table, they adapt Q-learning into the CNN backbone. Figure below shows more detailed outline of the model. Where the model input is an image and its output is the action game input for the agent in the game environment. From seven of the game tested it achieve state-of-the-art on six games.

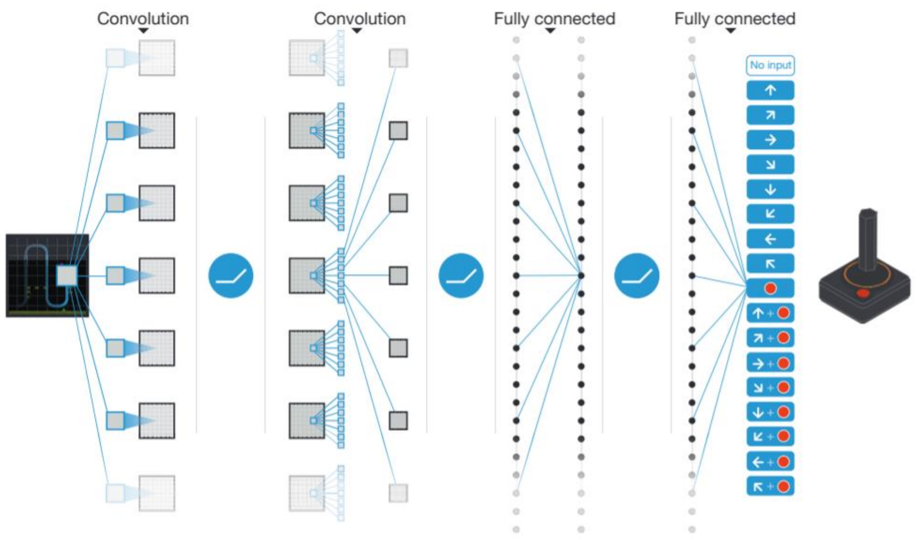
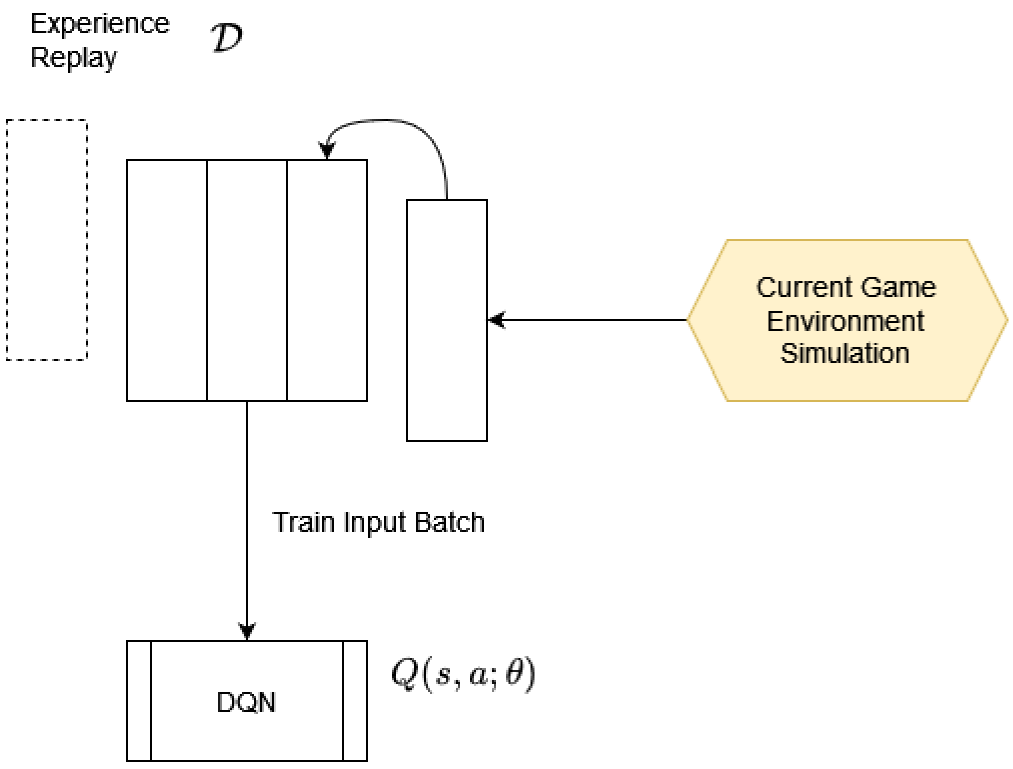


Illustration of DQN architecture. Image from Towards Data Science by Chao De-Yu.

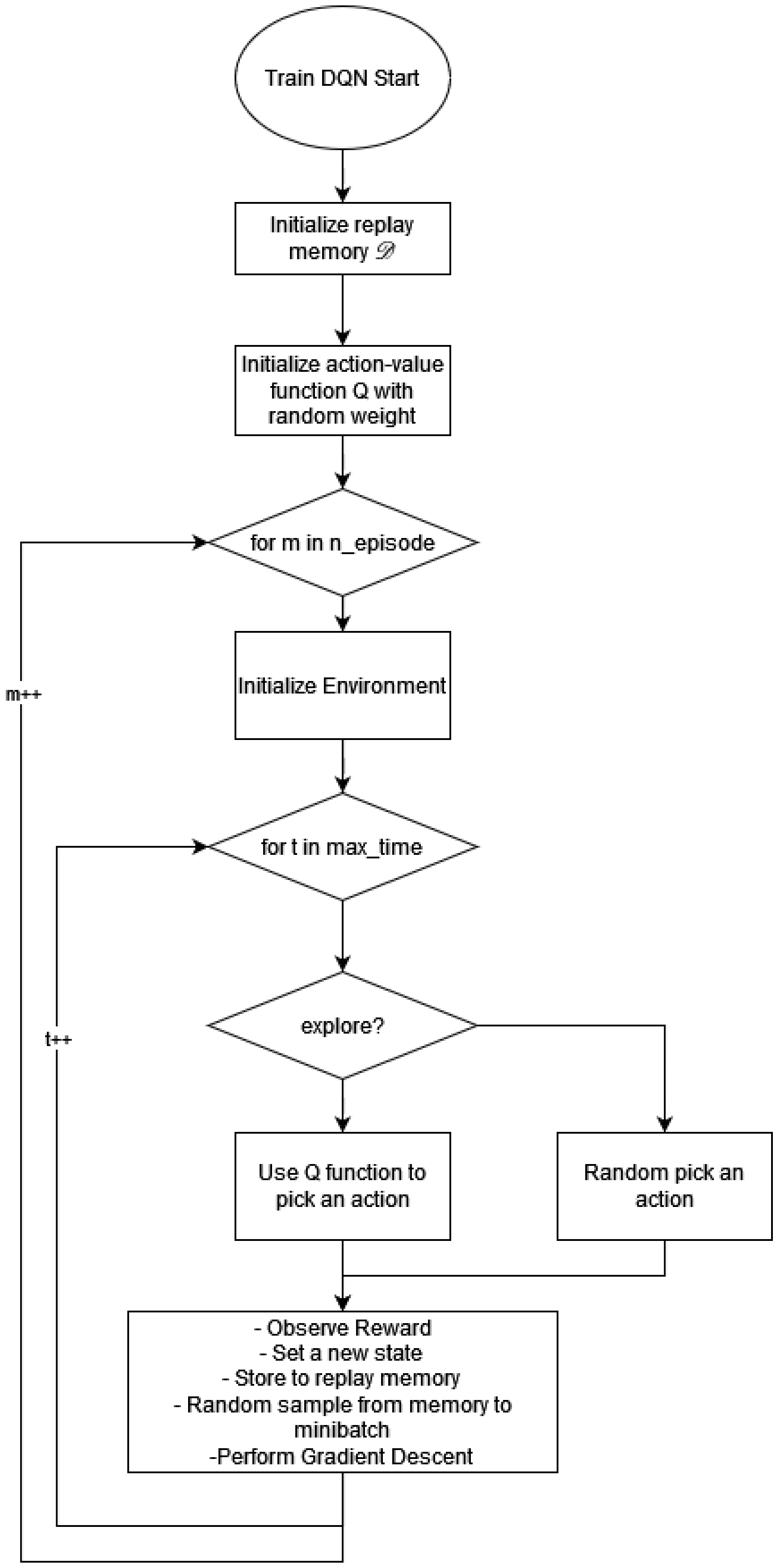
Novel Ideas and Technique

Other notable technique the paper introduced also are:

Experience Replay Memory

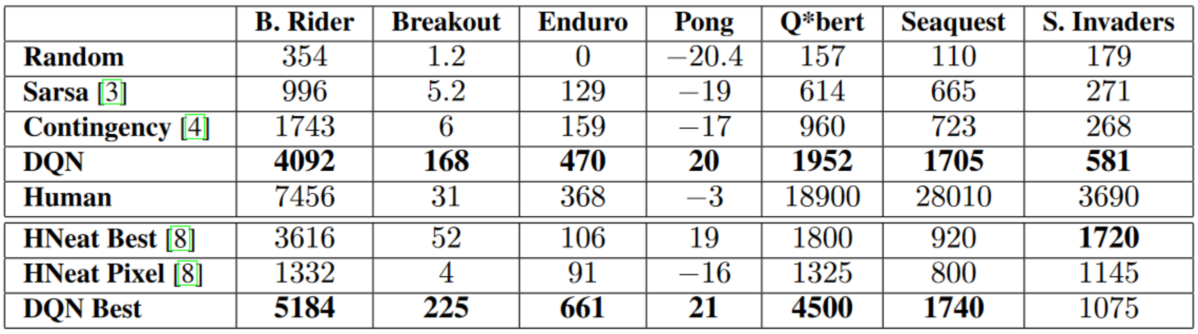


Experience Replay Memory created to save existing experience to optimize training. This memory resembles a Queue where new experience from simulation is saved until capacity full. Therefor the memory replay only saves N last experiences. One of the reasons this was used is due to neural network typically take batch of data to stabilize training altogether. By using batches, it also helps smoothen out the noise and ensure diversity in training data to make the network learn more meaningful weights.


Train DQN Start
Initialize replay memory $\mathscr{D}$
Initialize action-value function Q with random weight
for m in n_episode
Initialize Environment
for t in max_time
explore?
Use Q function to pick an action
Random pick an action
- Observe Reward
- Set a new state
- Store to replay memory
- Random sample from memory to minibatch
-Perform Gradient Descent
t++
m++

Outline of the training algorithm.

Results and Performances



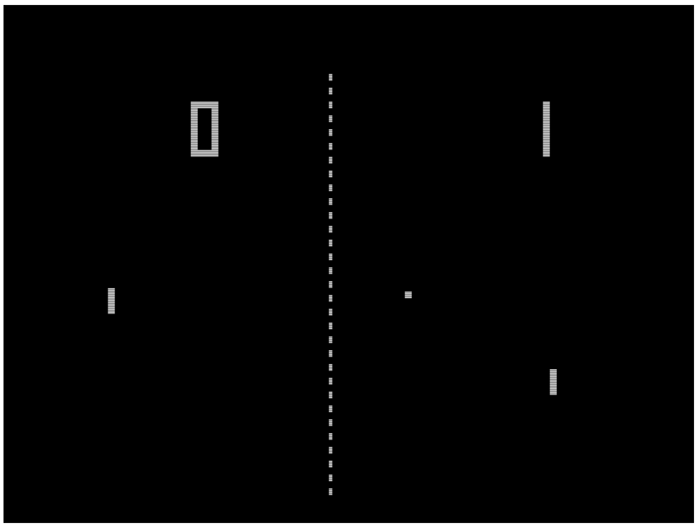
	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	-20.4	157	110	179
Sarsa [3]	996	5.2	129	-19	614	665	271
Contingency [4]	1743	6	159	-17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	-3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	-16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

DQN Result Table. The table shows the average total reward in comparison to DQN.

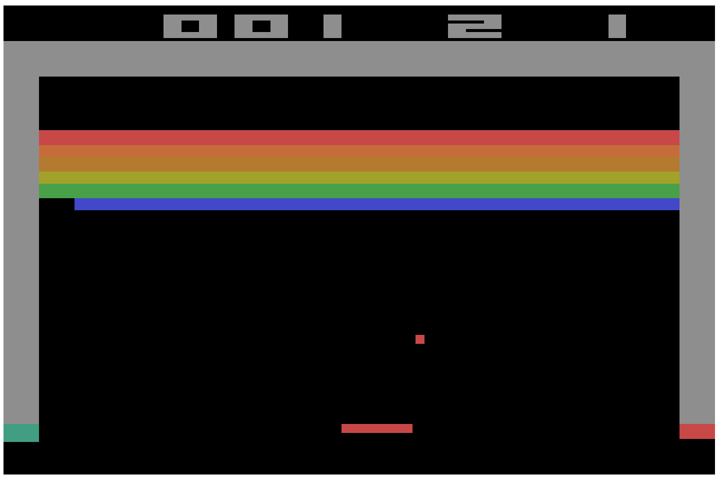
We can see while DQN beat other RL method on all task. However, interestingly DQN in comparison to human still lose except on few games such as Breakout, Enduro, and Pong.

If we analyze the game, we could see the game DQN outperform human mostly reflex-based game, where it makes sense that the algorithm can be much quicker than a human. For complex environments such as Space Invader, humans still win with a score 7 times higher and 16 times higher in Sea Quest. This means the model still have not fully grasped and processed the game environment as well as human.

Enduro Racing Game on Atari 2600. The game objective is to stay alive as long as possible while avoiding another car on screen.



Pong game on Atari 2600. The game simulates table tennis and player need to ensure ball did not pass through with moving paddle, blocking moving ball.



Breakout game on Atari 2600. The game stack layers of brick and it is the player tasks to destroy all layers by bouncing the ball with a given paddle.

Further Reading

Since its introduction many notable algorithms were introduced to fix the shortcoming of DQN, such as Double DQN, using Double Q-learning to reduce overestimation, or Duelling DQN.

However as mentioned, Q-Learning in general just one of massive area of Reinforcement Learning different approach altogether recent state-of-the-art are:

Temporal-Difference, Soft Actor Critic, DDPG, Monte-Carlo Tree Search (important algorithm behind many recent AI applications such as AlphaGo) and the new Deep RL for AlphaTensor).

For further hands-on tutorial, following link may be helpful:

Official PyTorch Reinforcement Learning DQN Implementation Tutorial

Official TensorFlow Reinforcement Learning DQN Implementation Tutorial