Final Project
Reinforcement Learning On Atari Pong

Netanel Hugi 203553490
Shimon Hagag 311367536
Shalom weinberger 203179409

September 26, 2019

1 Introduction

Atari 2600 Pong is a game environment provided on the OpenAI "Gym" platform. Pong is a two-dimensional sport game that simulates table tennis which released it in 1972 by Atari. The player controls an in-game paddle by moving it vertically across the left or right side of the screen. They can compete against another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth and have three action ("stay", "down", and "up"). The goal is for each player to reach 21 points before the opponent, points are earned when one fails to return the ball to the other. The OpenAI "gym" platform is a toolkit for developing and comparing reinforcement learning algorithms. It support teaching agents everything from walking to playing games. The unique feature it's has is that we can train NN (Neural network) on one game and use it to play other games in the gym environment. We create four AI agents that generates the optimal actions, two of them taking raw pixels as features. One by feeding them into a convolutional neural network (CNN), also known as deep Q-learning. One by simple Q-learning The two other use features given before training to generate the optimal actions. And we will discase them in detail in this paper.

2 Dataset and Features

The openAI "gym"(4) framework give us the dataset we need in terms of observations at every time step. An observation consists of pixel values of the game screen taken for a window of k consecutive frames. This is a numpy.array of shape (210, 160, 3), where the second value x is the height of the screen, the third value y stands for the width of the screen, and the fourth represents the RGB dimension for each pixel at coordinate (x, y). For each of the method we used we approach differently: For the Q-learning and the DQN(deep Q learning) we simply chosen the raw pixels processed into black and white pixels in binary

as features, and reduce the screen. For the two other methods use different approach and therefore we make a new dataset based on the data given from the openAi "gym" framework.

3 Related work

There are many other papers tackling using deep learning for playing video games. We will like to mention three: The first paper "Human-level control through deep reinforcement learning"(1) was the inspiration for are work, can we implement such system, system that learn to play be itself and even beat human score. Can we implement system in the level of "DeepMind" for example. Anther paper is "Playing Atari with Deep Reinforcement Learning"(2) addresses how convolutional neural networks and deep reinforcement learning combine together to accomplish a high performance AI agent that play Atari games. The paper analyzes how Reinforcement learning (RL) provides a good solution to game playing problem and also the challenges in Deep Learning brought about by RL from the data representation perspective. The paper proposes deep reinforcement learning on game playing agents, which is similar to our goal. The third one is: "Deep Learning for Video Game Playing"(3) It's given example for some game and NN that work for them (and show are main method not cheeked (or maybe doomed to fail.))

4 Methods

4.1 Q-Learning

Q-learning(5) is a value-based Reinforcement Learning technique. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. In this method we will use a dictionary(as a Q-table) that will include all the possible states in the game as a key, and for each state 3 possible actions as a value.Each state in the dictionary is a vector of 6 variables [(Ball-x,Ball-y),P1,P2,(Ball direction)], Where (Ball-x,Ball-y) represents the coordinates of the ball, P1(Computer) and P2(Agent) is the y coordinate of the left-top part of each player, and Ball direction represented by 2 points Old and New as (New.x - Old.x, New.y - Old.y).
To learn each value of this Q-table, we'll use the following Q-learning algorithm:

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

At the end of the learning process the agent can use the table to determine the optimal action for him.

- **Description of the formula:**

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

  - **Learning rate(α):** Determines how much the agent learns. From the formula we can see that if an α is close to 0 then it relies more on the old value, whereas if α is close to 1 then it relies more on the current information it has learned.
  - **Discount factor(γ):** Determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward.

4.1.1 Exploration vs. Exploitation

At the beginning of the learning process we would like to explore more of the actions that the agent can perform, and as we proceed we will want to Exploit the knowledge we have accumulated. In order to implement this idea, we define a variable(ε), which will start high (1.0) and the agent will learn, the variable will be reduced to a certain value (0.1). At each step of the agent, it will choose a random number between 0 and 1, if the number obtained is greater than ε, the agent will select a random action to explore. else, he will select the optimal action according to the Q-table.

4.2 Deep Q-Learning

This is an improvement of Q learning. As has been said before, Q-Learning helps us especially when dealing with game with a relatively small number of states, so that we can hold all possible states within a table in memory. In contrast, we will use deep Q learning when we have a large number of possible states in the game, so we cannot hold a table to such a cue, or it will be ineffective. In this case, we will use a neural network to predict the Q-values (and the next action for the agent). The difference in the Q-value calculation process is that in Q-learning we simply extract the Q-value from the table (by state and action), while in deep Q-learning, the neuron network predicts the Q values for all actions (based on what the network has learned so far). Our Deep Q Neural Network takes a stack of four frames as an input(We do this in order to simulate the movement of the ball). These pass through its network and output a vector of Q-values for each action possible in the given state. We need to take the biggest Q-value of this vector to find our best action. During the project, we implemented 2 models in deep Q-learning for our agent: Frames and Features based models.

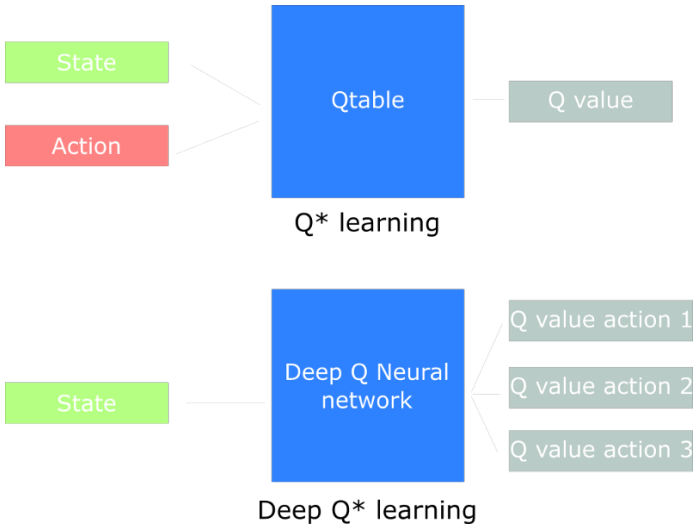

Figure 1: The differences between the methods.

4.2.1 Frames based model

This model uses a convolutional neural network. The learning process is performed on input that is frames(states) from the game. We process each frame, rather than saving the entire matrix (160 * 210 * 3 pixels), we cut the irrelevant parts of the image (e.g. scoreboard) and normalize the matrix (turning it black and white). The processed frame enters the stack, which is sent to the neural network.

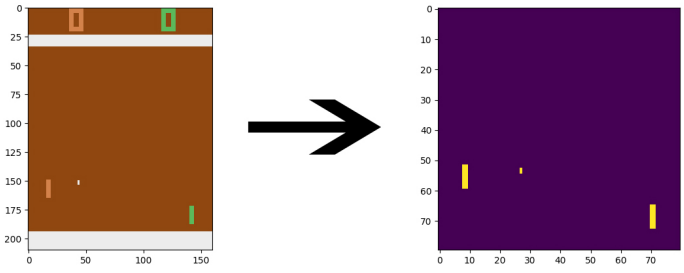

Figure 2: Original Frame vs. Processed Frame.

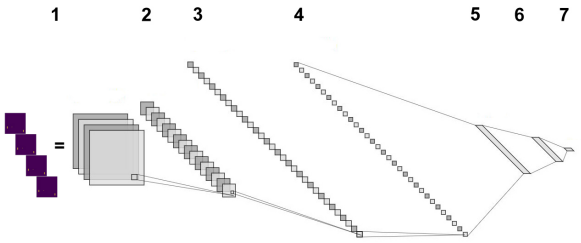

Figure 3: CNN architecture.

- **1. Input layer – The preprocessed frames, Size: [k,80,80,4].**
  - **k - The number of stacks entered on the network as input.**
  - **[80 * 80] – Board size.**
  - **4 frames in each stack.**
- **2. Convolutional layer 1 - kernel size: [8, 8], stride: [4, 4], output size: [k, 32, 19, 19].**
  - **32 Matrix of size 19*19.**
- **3. Convolutional layer 2 - kernel size: [4, 4], stride: [2, 2], output size: [k, 64, 8, 8].**
- **4. Convolutional layer 3 - kernel size: [8, 8], stride: [4, 4], output size: [k, 64, 6, 6].**
- **5. Flatten layer – size: [k, 64*6*6].**
- **6. Fully connected layer – size: [k, 512].**
- **7. Output layer – size: [k,6]. (Q-value for each possible action).**

4.2.2 Features based model

This model uses a regular neural network(without convolutional layers). The learning process is performed on an input which is an array containing numeric values that represent important features from within the game. The array contains data on players and ball positions. For the ball we will take the coordinates X and Y of its current location, whereas for the players we will only take the Y coordinate of their upper left edge (assuming the X value is constant and does not change during the game). In this model, too, we process the image (before searching for the locations), we cut the relevant section of the game board, reduce it from 160* 160 to 80 * 80, and then perform the search according to the colors of the players and the ball (which we know in advance).

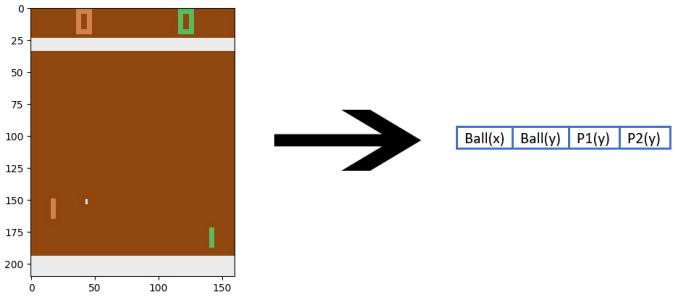

Figure 4: Original Frame vs. Features vector.

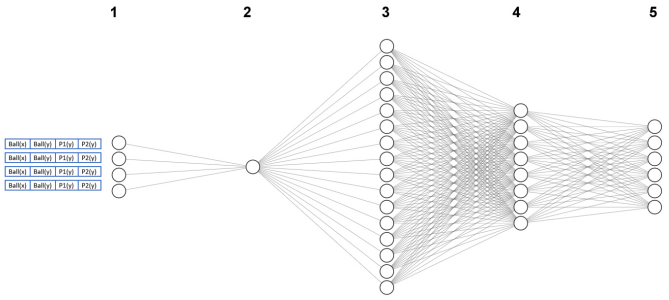

Figure 5: NN architecture.

- **1. Input layer – The preprocessed frames, Size: [k,4,4].**
  - **k - The number of stacks entered on the network as input.**
  - **4 – state size.**
  - **4 – frames in each stack.**
- **2. Flatten layer – size: [k, 4*4].**
- **3. Hidden layer 1 - output size: [k, 128].**
- **4. Hidden layer 2 - output size: [k, 64].**
- **5. Output layer – size: [k,6].**

Both models are actually quite similar, both using the same algorithms and methods for learning. The difference between them is expressed in the size of the input and the neural networks. Our main goal in the project was to create this model(features-based), in order to save resources in terms of memory and power that the computer exerts in order to train the agent. Finally, we saw that the standard (frame-based) model is more efficient in terms of learning time and results, but the same frame-based model was sometimes too heavy for the computers on which we ran the project.

4.3 Human behavior approach

Instead of going with unsupervised method, we try working on a smaller problem. Given a vector we made earlier as a data can a NN make a decision what action to use? instead of playing the game we build a NN that find which action to do based on the vector. We took the same structure of the vector and build a dataset of more then 1000000 vectors. for each vector we give an action what human player (or machine using rules) will chose given the situation . The new NN build : We try couple of models some with 1 layer more/less. Some with ADAMOptimizer and some with GradientDescentOptimizer. The basic was: Input layer in size of 6. 2 hidden layers holding 20-30 neurons and output layer with one neuron using the SoftMax function. Softmax, is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Using softmax and Cross Entropy Loss we get "one hot encoding" that give as the action prediction. The result is better than random but less than we expected. We must remember that even with good result this model success is limited by the dataset and therefore limited by humans players. Using this NN it is passable to build a platform that will play the game using only the vector and not the full screen as we tried in the beginning (but we can't be sure it will work on other Atari game in the openAi "gym" platform.)

5 Conclusions

We spend two semesters trying to build better model then the basic model without jumping to higher level of complexity NN models. Sadly we didn't find better approach then that already known and tested. Evan that our idea was support to get batter result on the paper. This gives us a lesson that not everything that seems effective and possible on theory, works in reality. In the end we build 4 different models . the model we try to mimic where as we assumed while the others were not up to our assumptions We must remember that this field is new and are work couldn't be done 5 years ago on regular computer and there is a lot of question and problem's to explore in the world of NN.

References

[1] Human-level control through deep reinforcement learning.
https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

[2] Playing Atari with Deep Reinforcement Learning.
https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf

[3] Deep Learning for Video Game Playing.
https://arxiv.org/pdf/1708.07902.pdf

[4] OpenAI gym: Game environment.
https://gym.openai.com/envs/Pong-v0/

[5] Q-Learning: Description and terms.
https://en.wikipedia.org/wiki/Q-learning