NEAT Algorithm vs. Hardcoded Paddle: A Comparative Study in Pong Game

36145142

***Abstract*—Pong is a game that attracted many people when it was first released in early 70s. The main interest of this project is to build an AI for Pong that resembles human player behaviour and is potentially able to beat the hardcoded paddle. We used evolutionary algorithm called NEAT to train 8 different models and then found the best one. We paired the models against hardcoded paddle and unfortunately none of them were able to beat it, however, it is still worth to note that some of them showcased superhuman abilities and were impossible to beat by a human player. Everything has been coded using PyGame library and the game allows the user to train their own AI to then test it by playing against it.**

I. Introduction

Pong, an iconic arcade video game, is widely recognized as one of the earliest successes in the history of video games. It was developed by Allan Alcorn and published by Atari Incorporated in 1972. Its rules are similar to table tennis where two players try to make the opponent miss the ball while protecting own side at the same time. In Pong, players use vertically moving paddles to hit the ball. In case of not having the friend to play against, it might be desirable to play against a ”computer” which has set of most optimal decisions. Most natural choice for such decisions is a paddle that moves up when the ball is above the paddle, and down when ball is below it. We called such paddle a ”hardcoded paddle”. Due to a rise of reinforcement learning techniques there is a hope of creating an AI model that is able to play in a more human-like way. Such AI player would definitely give more convincing experience when playing compared to hardcoded paddle. There are few possible approaches when creating AI to play a game.

Firstly, there are search algorithms that are based on simulating possible future game states and calculating their expected reward. Consequently computer is able to make decision that leads to highest reward state. However in our scenario, it might not be the best choice due to the real-time nature of the game which requires dynamic decision making. Moreover, in Pong the players might have different strategies, making it hard to generalize one algorithm.

Next approach can be using Q learning and its extension to neural networks, Deep Q learning. Traditional Q learning tries to approximate Q-table that associates each state-action with value based on observed and future expected reward. Such process is challenging to implement in our case since state space is represented by positions of ball and paddles which results in continuous high dimensional space. Deep Q learning addresses those issues by leveraging the neural networks, however we still had problems to effectively implement it. Main problem was the need of making model prediction and training every frame which made it impossible to visualise training process smoothly.

Instead, we implemented the approach that is an evolutionary algorithm. More specifically, we chose NEAT algorithm which uses genetic algorithm to evolve neural network for playing Pong. The whole training process resembles biological evolution.

In the accompanying Python code we implemented Pong game using PyGame library where the user can choose out of 3 different play modes: PVP (player versus player), Train AI and VS AI. We allowed to choose some of the parameters for training process as well as the mode in which the AI will be trained. In essence, it can learn either from playing against itself or a hardcoded paddle. After the model is trained the user can test it by playing against it himself, or by putting it in a match against a hardcoded paddle or another trained model. In the next section we review the relevant literature. Next, we will explain the NEAT algorithm and our game mechanics. After that, we will judge the performance of the trained models by making a tournament of them and setting the winner against a hardcoded paddle to see if it will be able to beat it or achieve the same level of competence. Lastly we will discuss our results.

II. Related work

The idea of using neuroevoultion approach for creating AI to play Atari games has been studied in before. In [1] the authors applied and compared NEAT along with other neuroevolution algorithms to develop a general agent that is able to play any Atari game without any game-specific heuristics. Moreover, 3 proposed strategies for different state representation were evaluated. In the first one, the agent receives the input of images of the objects that can interact with him. The agent then manually detects if the objects are present on the screen and where. Second approach is raw pixel representation, which simply provides pixel values of the screen to an agent. Lastly there was variation of the approach before which contained seeded noise to ensure that the agent does not blindly remember sequence of moves. It was found that the object representation performed the best in most of the games, including Pong, and our state representation will be similar to it.

There exist some work that managed to effectively implement Q learning [2] and DQN [3], [4], [5] in Pong game. In all cases agents were able to achieve performance better than a typical player. [4] recreated their own version of DeepMind’s DQN from Google. 2000 games were enough to train best performing model, however, what was found concerning is the same pattern of action of the agent which was always going to

the bottom of the screen and then trying to hit the ball when going up. In [5] the author implemented DQN in Pong using Unreal Engine environment, where it took 6 hours of training to achieve a model that is able to beat a built-in AI. In [3] an interesting question has been posed, which is whether an agent that has been trained adversarially is better than one trained non-adversarially. Essentially, it was found that changing the training process to set up the current model against its previous version increased the robustness of the agents significantly, which made them all-around harder opponents. In our project we will utilise similar idea.

III. Methodology

A. Game setup

The game screen consists of two paddles that can be controlled by W/S and UP/DOWN keys on the keyboard. Top of the screen is separated with the dotted line and contains current score of each player. The screenshot from the game can be seen on Figure 1. At the beginning of each round the ball gets random y coordinate velocity to ensure more robust agent that is resilient to more situations (at the same time it prevents infinite loop where two paddles can stay in the same place forever). The parameters for paddle width/height/speed along with ball speed/radius are easy to customise from the options screen, which allows to test future models in different scenarios. The game ends when the first player achieves 15 points. At the end of each round the data with the number of hits, and the current score gets saved into a .csv file that we will base our analysis upon.

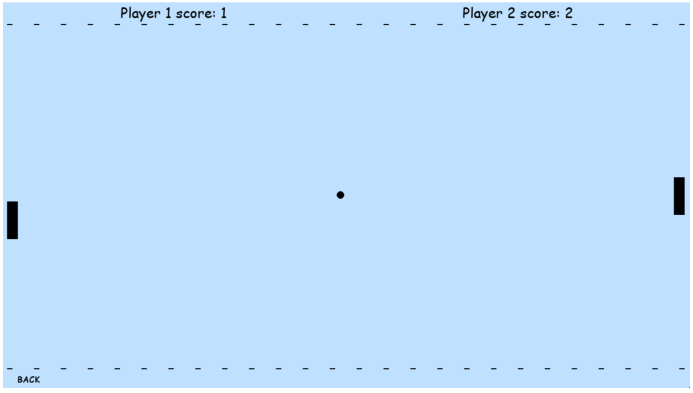

Fig. 1: Screenshot from a PVP game

B. NEAT Algorithm

Here, we will explain how the NEAT works in theory. To begin with, NEAT stands for NeuroEvolution of Augmenting Topologies, which suggests that the algorithm will be capable to grow the structure of neural networks over time. The process starts with random creation of n (in our project $n = 50$) small neural networks. Each member of that population then gets evaluated according to some predefined fitness function (our will be presented in the next subsection) and gets assigned a fitness score. The networks with higher score are more likely to be chosen for reproduction, which happens by crossover and mutation. Crossover combines two parent networks into an offspring. Each weight and connection is encoded into genes with each one being assigned its historic origin (identification number based on when it was first introduced). The genes with matching origin are selected randomly, while the disjoint ones are inherited from fitter parent. During mutation, the offspring goes through random changes in its structure (new nodes are created and weights are changed). This introduces exploration aspect into the population. Next step is speciation. Based on aforementioned historic origin, networks are split into species that share similar topology. The best performing percent of networks from each species is then chosen to mate. This step is put in place to avoid situation where the smaller type of network will extinct before it managed to optimise. The offsprings are then evaluated and the whole process is repeated until certain fitness score is achieved or number of generations is reached [6].

C. Training

We decided that the input layer will have 3 neurons for: y coordinate of the paddle, distance along the x axis between the ball and the paddle, and y coordinate of the ball. Based on these inputs we believe that it should be enough to train sensible agent. The reward/penalties system that we implemented in the fitness function is as follows:

- +0.2 fitness when the decision is to move (whether up or down)
- -1 fitness when the decision is to go up when paddle is at the top boundary or down at the bottom boundary (illegal move)
- +10 fitness for each time the paddle managed to hit the ball
- +1 fitness for won game

There is reward for moving due to the fact that we do not want the paddle to be too passive and even though the reward seems small, it is applied in each frame hence it adds up, and moreover, acknowledges agents that last the longest. The reward for winning the game seems relatively small but we wanted to mainly focus on the paddle to pick up the basic point of the hitting back the ball. In the long run it might be detrimental to the agent as it might be able to bounce most of the balls but not in the ”aggressive” enough way to win the game.

After going into ”Train AI” screen, user can choose some particular parameters of the training process. Firstly, user can specify number of generations that he wants to train the model for (for the model to be saved all the generations need to finish). Secondly, the activation function that the network will use. The ones available are: *relu* and *tanh*. For the *relu* the output layer has 3 neurons and the decision is made based on the index of neuron with the highest value. For *tanh* different approach was integrated. Essentially, there is only one output neuron and the decision is to:

- Go up if value of the output > 0.9
- Go down if value of the output < -0.9
- Stay in place if $-0.9 \leq$ value of the output ≤ 0.9

Lastly, user can choose the mode in which the agent will be trained. ”Vs hardcoded” means that the opposed paddle

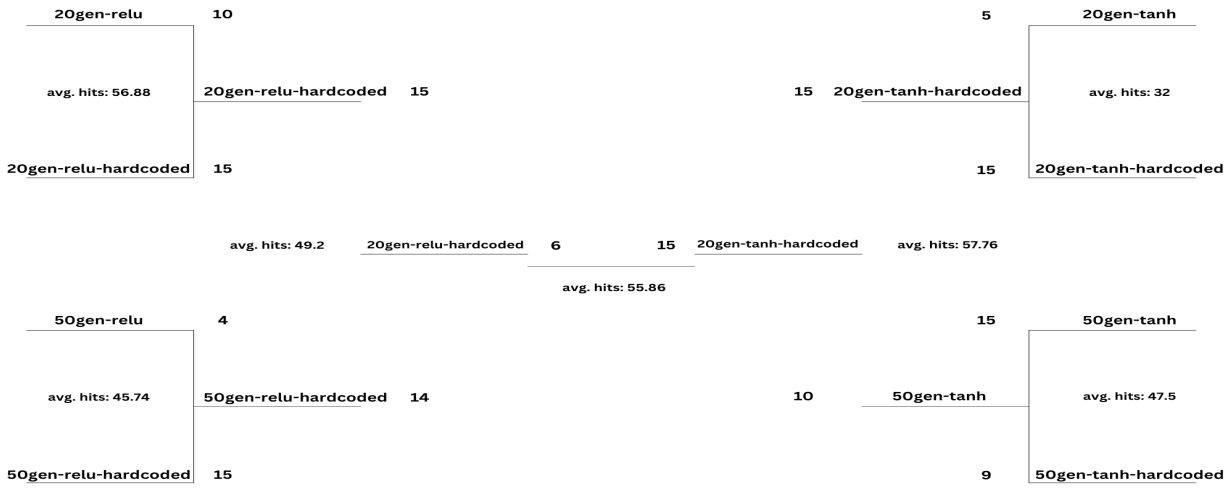

Fig. 2: Results of tournament between models

will strictly follow the ball. We thought that since we want to compare our models to hardcoded paddle later, it would be useful if the model would be trained against it. The one downside to this is that such model might be too specialised when playing with that paddle but its generalisation abilities might be poor. The potential remedy for it is another mode called "vs itself". It is implemented in a way where during evaluation step of NEAT, each genomes plays with every other. Consequently the length of the training is increased significantly. To shorten the training process the paddles play with themselves until the number of total hits of 30 is reached, after which the new game is started with the new genome. We have pre-trained 8 different models for combination of each mode and activation function trained for 20 and 50 generations. In the next section we will try to find the best model out of them and then pair it against the hardcoded paddle to see if it is able to outperform it.

IV. Results

The problem that we experienced when testing our models is that some of them were too good to ever miss the ball. To ensure that someone loses in the end, the velocity along the x axis is increased by 1% with each hit by any paddle. Also, velocity along the y coordinate is increased by the variable that is initially equal to 0.5 but is incremented by 0.1 after each bounce.

A. Against themselves

When pairing the models between each other we designed it in a way that first we test if hardcoded mode resulted in better models, then we test whether 20 or 50 generation model was better and lastly we find the better activation function. The entire bracket with the results can be seen in the Figure 2. We can see that the models that were trained against hardcoded paddle usually dominated the models trained against itself (one exception of 50 generation tanh model). It is quite surprising as we expected them to be poor in terms of generalisation. Besides, these models required significantly less time to train. Next observations is that 20 generations models outperformed 50 generations models. It is worth to note that both games were close, however in the end, models that took less time to train won again. Potential reason for that could be that models that were trained against themselves needed even more generations to advance further. Meanwhile the ones trained against hardcoded paddle were able to pick up skills very quickly but then plateaued. Lastly, we compared the best *relu* model with the best *tanh* model. It was found that *tanh* model performed better with the score 6-15 (average hits of 55.86 per round suggests that it was close). To strengthen our claim we repeated the final game 2 more times with results: 7-15 (avg. hits 48.36) and 14-15 (avg. hits 55.97) for *tanh* model. Each of the models can be tried by the user when going into "Player VS AI" screen.

B. Against hardcoded paddle

In this subsection we will make each model play against the hardcoded paddle and then more extensively research if the best model found in the subsection before is worse/same/better as/than hardcoded paddle.
First we present results of the games in the Table I.

Model	Score	Avg. hits
20gen-relu	0-15	11.46
20gen-relu-hardcoded	7-15	64.41
50gen-relu	1-15	50.81
50gen-relu-hardcoded	2-15	62.06
20gen-tanh	0-15	47.73
20gen-tanh-hardcoded	**7-15**	**65.14**
50gen-tanh	7-15	70.1
50gen-tanh-hardcoded	5-15	65.1

TABLE I: Results of games against hardcoded paddle (in bold the best model found in the tournament previously)

As we can see the results of these games converge with results of our tournament and the models that went further in the competition, achieved better results against hardcoded

paddle. In general, we can also observe that *tanh* side of the table performed better by scoring more points total. Now, we will look into competition between *20gen-tanh-hardcoded* model and hardcoded paddle and determine, using confidence interval, whether the probability of this model winning against hardcoded paddle is 0.5 or less/more. To make our analysis more reliable we played the match against them 5 times and summed up total number of rounds won, with the result being 28-75 for hardcoded paddle.

Let's assume that $X \sim \text{Binom}(n = 103, p = \theta_0)$ and we set

$$H_0 : \theta_0 = 0.5$$
$$H_A : \theta_0 \neq 0.5$$

where $x_i = 1$ when *20gen-tanh-hardcoded* model wins the game and 0 otherwise. According to the data we collected $\hat{\theta} = \frac{28}{103} \approx 0.272$. The 95% confidence interval for parameter θ_0 is given by:

$$\left(\hat{\theta} - 1.96 \times \sqrt{\frac{p \times (1-p)}{n}}, \hat{\theta} + 1.96 \times \sqrt{\frac{p \times (1-p)}{n}}\right)$$

$$\left(0.272 - 1.96\sqrt{\frac{0.272 \times 0.728}{103}}, 0.272 + 1.96\sqrt{\frac{0.272 \times 0.728}{103}}\right)$$

$$(0.186, 0.358)$$

Consequently, 0.5 is not in the confidence interval, hence there is enough evidence to reject null hypothesis and we can conclude that models do not have an equal chance to win.

Owing to the analysis we have evidence that even our best model is not able to play on the same level as hardcoded paddle. It is possible that the bar has been put too high by hardcoded paddle and the models should be compared to the real humans playing. However, due to the lack of representative sample of people we decided to stick with comparison that we conducted. Overall, by playing with the models we can pick the top 3 models (ones that had 7 points against hardcoded paddle in Table I) that were impossible to beat.

V. Discussion

As we have seen, by applying NEAT for training AI in Pong game, we successfully achieved "smart" enough paddles to stand up against, and sometimes even considerably outperform, human player. Potential improvement to our models could be direct extension of NEAT, called HyperNEAT, designed for creating large-scale networks. Alternatively, we could also implement DQN and see how it compares with our models. Due to the hardship of successful integration of this method with visualisation of the process on the screen we decided to pass on this. However, in the future it might be an interesting game feature to add new algorithm for training AI.

VI. Conclusion

To summarise, we have created a Pong game environment using PyGame (the game should look the best on Windows, as when tested on Linux some of the text displayed were in unintended places) which allows user to train its own AI model and then play against it or set it up against other AI model. Alternatively, user can choose out of 8 models that we pretrained. The models that we trained using evolutionary algorithm NEAT showed great skill in the game already after 20 generations. The future improvements to the project might include implementation of another algorithm that would extend the possibilities for comparison of them. Also, potential change in the fitness function that would reward winning more than just hitting back the ball might result in more optimised models for winning. This coupled with training for more generations might result in agents that would be able to beat hardcoded paddle.

VII. Acknowledgements

The code for the base game was inspired by (but design has been customised) https://www.youtube.com/watch?v=vVGTZlnnX3U&ab_channel=TechWithTim. The implementation of the NEAT algorithm was based on https://www.youtube.com/watch?v=2f6TmKm7yx0&ab_channel=TechWithTim with our own penalty/reward system. The idea how to implement main menu in PyGame was taken from https://www.youtube.com/watch?v=GMBqjxcKogA&ab_channel=BaralTech. Even though the implementation of the algorithm was based on some existing code, connecting and operationalising it with other presented ideas required coding knowledge and extensive understanding of the existing implementation.

Libraries used for coding:

- Numpy
- Pandas
- PyGame
- Neat-python

References

[1] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014.

[2] A. Rawat, R. Panday, and A. K. Pandey, "Pong game using ai."

[3] B. He, S. Rammohan, J. Forde, and M. Littman, "Does dqn really learn? exploring adversarial training schemes in pong," *arXiv preprint arXiv:2203.10614*, 2022.

[4] L. Tyler, "Deep q-learning with pong," 2019.

[5] I. Makarov, A. Kashin, and A. Korinevskaya, "Learning to play pong video game via deep reinforcement learning." in *AIST (Supplement)*, 2017, pp. 236–241.

[6] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 2. IEEE, 2002, pp. 1757–1762.