Applying CNN-LSTM networks and EA for game bot development

Jay Ashworth
Electrical Engineering and Computer Science
University Of Tennessee - Knoxville
washwor@vols.utk.edu

Andrew Mueller
Electrical Engineering and Computer Science
University Of Tennessee - Knoxville
amuell11@vols.utk.edu

***Abstract*—Bot development in games is critical in creating a robust experience for gamers at all levels. Incorporating a well developed bot into a game is not something that comes easily. In the following paper, we explore a way in which bots can be developed, and trained to act on it's own through the implementation of a convolutional neural network, a long short-term memory network, as well as using an evolutionary algorithm. We decided on this approach as each part serves an important purpose, the CNN serves as a way to extract features like game elements, the LSTM provides spatial and time context for the game, and the EA provides the training method in order to push the best individuals over time. The game in which we will be developing this bot with is Diep.io, a fairly intuitive game in which you are a tank, and you need to level up by destroying various game elements, and enemy tanks. Unfortunately, our implementation did not pan out the way we wanted, as there were a fair amount of challenges we had not anticipated, however, we can say we do have a base framework for improving our implementation.**

***Index Terms*—CNN, LSTM, EA, Bot, Game Development**

I. Introduction

In the rapidly evolving realm of artificial intelligence, one area that continues to captivate both developers and gamers alike is the use of sophisticated AI systems in video games, more specifically, creating dynamic and real life bots. By leveraging AI, developers aim to create more immersive, engaging, and challenging experiences for players. This paper delves into our endeavor to develop an AI bot for Diep.io, a multi-player online game where players control tanks with the objective of destroying various game elements and enemy tanks to accrue points and level up.

Our approach employs a hybrid Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) network for the task. The CNN component is utilized to extract game features, such as objects and enemy tanks, while the LSTM network provides spatial and temporal context. This setup allows the bot to remember and predict certain game events, for example, it can still know an enemies general position even if it moves off screen.

A critical aspect of our research is the use of an evolutionary algorithm in which the training of each population will be unsupervised. This method allows us to evolve the AI system over time, without having to train it, gradually improving its abilities by retaining the most effective individuals from each generation and further refining them. By training each individual in the population multiple times, in our case three, and averaging their scores, we ensure a steady progression of improved performance across generations.

However, our ambition stretches beyond the game of Diep.io. We foresee the application of our approach to various game genres, extending to a vast array of more complex and diverse video games. The adaptive and predictive capabilities of our system could significantly enhance the gaming experience by providing more intelligent and responsive opponents, where the dynamic and unpredictable nature of player actions would demand a higher degree of adaptability and predictive accuracy from the AI. The use of a hybrid CNN-LSTM network, combined with evolutionary algorithms, could significantly enhance the game bots ability to analyze and respond to rapidly changing environments and strategies.

Moreover, we acknowledge the potential of this technology in creating advanced simulations for various purposes. Although our focus lies primarily in game development, our methodology could potentially lay ground work for various other technological domains like disaster management simulations, driver less car technology, and even virtual training environments.

Ultimately, this paper endeavors to demonstrate how our AI bot development for Diep.io can pave the way for more advanced game AI systems and simulations. The journey towards creating more intelligent, adaptable, and challenging AI in video games is an exciting, ongoing process and we hope that the following work can be improved upon.

II. Related Work

In this section, we'll delve into the existing landscape of CNN-LSTM networks and the use of evolutionary algorithms for bot creation. Though the body of research in this area is somewhat thin, we'll break it down systematically, examining CNN-LSTM networks, evolutionary algorithms.

CNN-LSTM networks, despite not being used for our exact purpose, have applications with potential relevance. These networks are employed in extracting human action data from video [1], which might be useful for our research if we venture into advanced simulations. For instance, in creating bots for 3D games like first-person shooters, capturing player movement is crucial. CNN-LSTM networks are also used in cybersecurity for detecting cyber attacks with an impressive 98% accuracy

[2]. Although unrelated to gaming, this demonstrates the wide range and effectiveness of these networks. Further, LSTM networks have been used to develop bots for the 5v5 multiplayer online role-playing game, League of Legends [4], and the 3D game, Doom [5]. Further more, in a car racing game, a team of researchers heavily relied on an CNN-LSTM network in order to develop a self driving car bot [8]. In their case, the car was able to learn quite quickly in various driving conditions. These cases, along with others, suggest that CNN-LSTM networks are instrumental in developing effective bots for a variety of games, from simple 2D to more complex 3D games, utilizing the extraction features these networks offer.

Switching gears, let's discuss evolutionary algorithms (EAs), also known as genetic algorithms, and their application in gaming. Ever since the popular mobile game, Flappy Birds, hit the scene in 2013, numerous attempts have been made to use EAs to transform the bird into an unbeatable game character. One team, employing NEAT and reinforcement learning, achieved impressive results [3]. This just goes to show that even games not originally intended for bots can benefit from EAs. A different approach was taken by a team who used EAs to fine-tune parameters for existing bots in the 3D first-person shooter game, Counter Strike [6]. They demonstrated that EAs can help to optimize bot efficiency. In a similar manner, a team used EAs to train a bot to play Super Mario Bros [7], showing marked improvement over generations. In another interesting case, EAs were used in conjunction with a neural network to train bots to fight each other, thereby testing the game's skill balance [9]. Lastly, researchers used an EA to develop a bot for the first-person shooter game, Unreal Tournament, creating three fitness functions similar to ours [10]. The take-home message here is that EAs can provide a robust framework for bot development in games, offering numerous objectives for crafting well-rounded bots for both 2D and 3D games.

In conclusion, the research and applications in the areas of CNN-LSTM networks and evolutionary algorithms demonstrate their potential for game bot development. Although the current body of research isn't extensive, the available work suggests that these tools provide a solid foundation for creating effective and adaptable game bots. Whether it's capturing human movement from videos or optimizing a bot's decision-making parameters, these technologies offer a range of possibilities that can be tailored to specific gaming environments. The goal is not only to create bots that can win games, but to create bots that can adapt, learn, and provide a more engaging and dynamic gaming experience.

III. Diep.io background

Before we move on, we will explain the breakdown of the game we are using, diep.io. The core of the game comes down to leveling up, and being the best player. You start off with being a tank, that has one gun, slow movement, and slow shooting speed. As you explore the map space, you encounter various game elements, which are shown in fig. 1. These game elements score you points if you destroy them, where the game elements increase in difficulty of destroying them, i.e., it takes more bullets to destroy them. As you move through the map, you will also encounter various enemy tanks, which are controlled by other players, and your main objective with them is again to destroy them. As you do, and you gain more points, you can upgrade your tank with various upgrades, such as speed, bullet penetration, and even more specialized types of tanks, for example a sniper tank. As seen in fig 2, we can get a breakdown of how many ways one can upgrade the tank, each coming with different abilities one can use to get an edge over the other players.

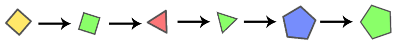


Fig. 1: Game elements in increasing difficulty

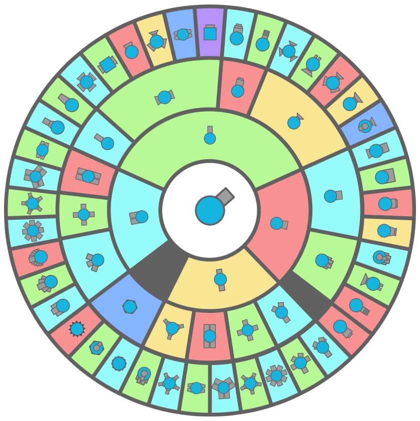


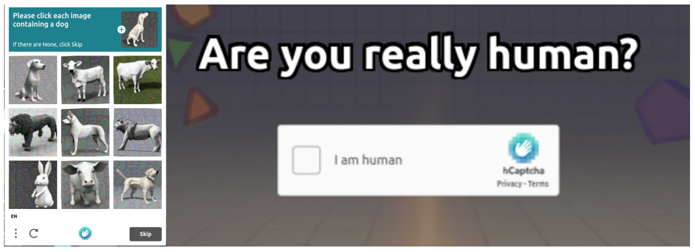
Fig. 2: Tank upgrade map

IV. Challenges

In developing a bot for Diep.io, we encountered various challenges along the way which made it a bit difficult to implement our learning algorithm. Below are a few of the challenges we encountered.

Captcha

Because a lot of games do not want bots playing the game, Diep.io introduced a captcha system in which to deter bots. Below is an image of what this captcha is, and why it is impossible to bypass this without someone monitoring the training.


Please click each image containing a dog
If there are None, click Skip
EN
Skip
Are you really human?
I am human
hCaptcha
Privacy - Terms
Fig. 3: Captcha

Rate limit

Another problem we encountered was connection timing. Currently, Diep.io has a rate limit, in which you can only connect to the game 3 times within a minute. This was problematic as if the bot died more than 3 times in a minute and had to connect for the next training bot, it would crash our program. We had to implement a timing protocol in order to get around this, which slowed down the training time.

Sequential training

Unfortunately, our set up didn't allow for more than one bot to be trained at a time. We had to train each bot individually, which is explained a bit more in the methodology section, but overall, training was slowed down because of this problem.

V. Methodology

In the following section, we will discuss our methodology. As a broad overview, and as stated above, we used various networks in our approach. The first is a convolutional neural network. The CNN's job is to extract features from the game. On top of the CNN, we also used an LSTM network which is responsible for extracting spatial and time context. In order to get the best bot, we utilized a EA. And finally, we will talk about why we are using unsupervised training as opposed to supervised training. At the end of the breakdown, we included a summary section of all the different aspects of our approach.

Fitness function

Below is the fitness function we decided to use. Although this may not be the absolute best, we did find that this fitness function is enough for this game.

$$F(s,t) = w_s \times \left(\frac{s}{S_{\text{max}}}\right) + w_t \times \left(\frac{t}{T_{\text{max}}}\right) \tag{1}$$

where:

s is the score,

t is the time alive,

S_{max}: max possible score: 50000,

T_{max}: max possible time alive: 6000,

w_s: weight assigned to the score, 0.8), and

w_t: weight assigned to the time alive, 0.2).

The weights we decided to assign are .8 and .2, for the score of the bot, and the time the bot spent alive respectively. We decided to use these values as we want to prioritize the score over the time it spent alive, as even if the score is high, and the time spent alive was low, we don't want the fitness score to be based too much on the time alive, as the time spent alive isn't as important as the score the bot obtained.

CNN-LSTM

Our model integrates a Convolutional Neural Network (CNN) with a Long Short-Term Memory (LSTM) network, making it a CNN-LSTM hybrid model. It begins with an input layer that receives an image from which we extracted it from the browser. Following this, the model applies a series of convolutional and max pooling layers to extract features from the image. The CNN section of the model consists of four convolutional layers with 32, 32, 64, and 128 filters respectively, each having a kernel size of 3x3 and using the ReLU (Rectified Linear Unit) activation function. The max pooling layers interspersed between the convolutional layers have a pool size of 2x2 and serve to reduce the spatial dimensions of the output from the preceding convolutional layers, thereby decreasing the amount of parameters and computation in the network. The output from the CNN section, a set of 2D feature maps, feeds into an LSTM layer with 32 units, which analyzes the sequence of these features. The LSTM layer uses the 'tanh' activation function. The final section of the model is a fully connected dense output layer that generates the predictions of the model. The final output of the model is an array of size 15, in which holds the inputs for the bot, for example, move left, right, or perhaps shoot it's gun. We ran into a problem trying to graph our model, however, below is an image of how the model is generally laid out. To reiterate, this model is designed purely feature extraction, and prediction on what the bot should do next. Below are two figures, fig 4 shows what the model looks like visually from what we have outlined above, and fig 5 shows a generalization of the model we are using.

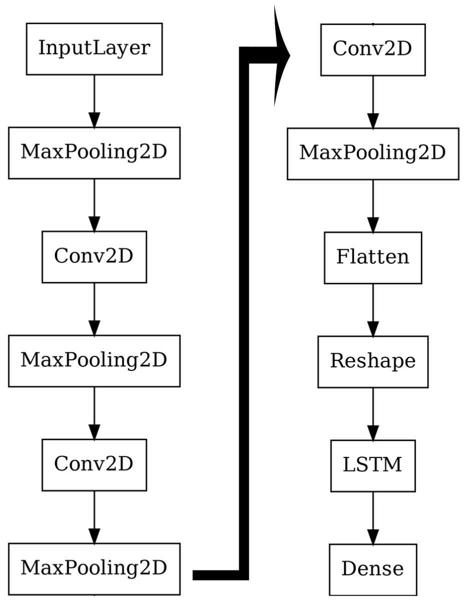


Fig. 4: CNN-LSTM model visual

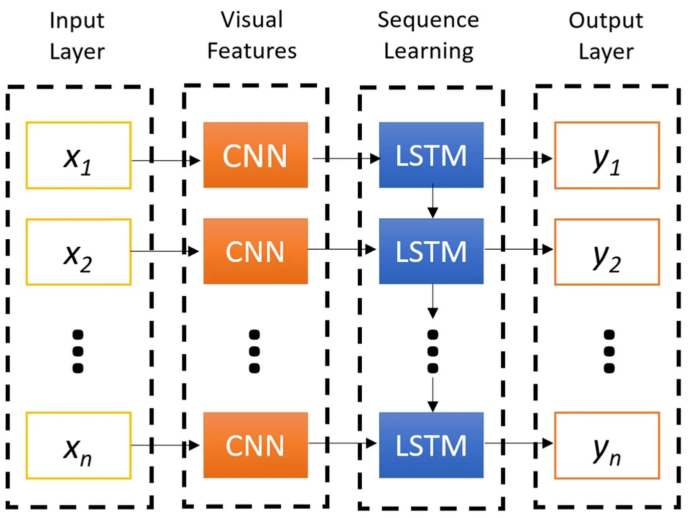


Fig. 5: CNN-LSTM generalization visual [11]

EA

Our evolutionary algorithm comes from the DEAP library in python [12]. We decided on using this library for a couple reasons, one, we had quite a time constraint, so developing our own EA from scratch was infeasible, two, we decided that we would rather use an EA that has already been established. For the EA, we start with an initial population of 10 randomly initialized individuals. We decided to run this for ten generations, as due to time constraints, and how we needed to set our method up, going past ten would take quite some time. For each individual, we run the individual three separate times, and average the score for that individual. We do this as to attempt to remove high score biases, as that one individual could have just gotten extremely lucky. After each generation is ran, the three best individuals from that generation move onto the next generation. In our implementation, we decided to use a Two-point crossover from the DEAP library. Essentially this is picking two points from the parent, and mixing them in with the offspring for each new generation. Furthermore, we decided to use a cross-over rate of 50%, and the mutation probability is set at 10%. fig 3 below shows the rates in which we choose to use in a table, for easier visualization.

Parameter	**Value**
Cross over	50%
Mutation	20%
Initial population	10
Tournament size	3

TABLE I: Parameters for the genetic algorithm

Unsupervised Training

This section will be relatively short, but it explains the reasoning behind using an unsupervised approach to learning. Unsupervised learning allows for us to run this without having to pre-train the model, mainly, the CNN-LSTM network. We decided on this approach because we wanted a robust system, that can pick up patterns, and come to an optimal solution based on purely the patterns it picked up. Another factor into using this approach was the lack of data we could get from the game, as there is very little of it. Due to time constraints, we also were quite limited in the amount of time we could spend recording best movements, and then filtering all the recordings to the ones we wanted to use. With all this being said, it is worth mentioning that implementing a pre-training phase would not be difficult to implement, as various other games that are much more popular, and could use our framework, would potentially have much better data to train the models on.

Summary

Below we summarize the entire implementation as written above. We use an CNN-LSTM model in order to extract features from the game. What we do is feed the model raw images from the browser, and get an array of 15 cells as output. This array is predictions on what the tank bot should do as it's next move, may that be moving across the map, or shooting at a specific location. We then start off with an population of 10, where each individual in the population is trained 3 separate times, and the score is averaged among the three runs. We do this because we want to filter out luck based runs, in order for the correct individuals to cross over to the next generation. Again, the mutation rate is set at 20%, the cross over rate is set at 50%, and the tournament size is set a 3. We have decided to take an unsupervised training approach, as we want it to be able to develop a bot with little to no human interaction.

VI. Results

Due to issues with the training program (and Linux being Linux), we only have qualitative results for our network as the numerical results got deleted on generation 10 in across two runs (that took six hours each). Qualitatively, our model performed acceptably given the nature of the project. In the first generation, nine out of the ten networks had no intelligence meaning they drove into the wall repeatedly or stood still until an enemy killed them. However, we had a single individual that showed slight intelligence. They were able to navigate through the world and even managed to (albeit accidentally) eliminate an enemy tank. This was an amazing result for a randomly generated tank. As expected, this network was very successful throughout subsequent runs. As a result, the average fitness did increase between generations 1 and 10. However, this type of intelligence was not the only type that became common. By generation 10, the AIs evolved into one of three camps. The first were the ones derived from the really good model from the first round. These actually improved slightly through mutation by the end and were able to "un-stick" themselves whenever they ran into a wall. The second group that developed were ones that would find the left side of the wall, drive back and forth, and shoot at the bottom right. This allowed them to rack up a fair amount of points, especially if there were a large number of squares and rectangles in their path. The final type was interesting. It developed itself to do nothing but hide. It would sit in one location and do nothing until it spotted an enemy or a large enough number of bullets. Then, it would drive away. This obviously did not achieve a high score in-game, but it was able to cheat the fitness score by getting a long play time.

VII. Conclusion

As stated in the results section, our AI was able to perform admirably given the setup it had. From nothing, it was able to achieve at least 300 points per round or survive 3+ minutes. This result shows some promise for the idea of using our model architecture to play games. Despite that, our project left a lot to be desired. We came into the project expecting to be able to get a bot that could at least engage in combat with enemy tanks. However, we simply did not have the training time to achieve such a great result. Additionally, its unclear as to whether the model could reach a competitive result within a reasonable amount of tries as similar setups require hundreds of thousands of iterations to get something halfway decent.

VIII. Future Work

Our current model, and over all algorithm has room for improvement. The first thing we would like to implement is a way in which we can have all individuals in a generation train simultaneously. Due to current game limitations, we were unable to do training in parallel, however, we are hoping to develop a way in which it can be done in parallel. Training in parallel would also allow for a much faster evolution, and also develop a way to hyper tune parameters in order to develop a set of parameters that would achieve an optimal solution faster.

Past developing a better training methodology, we would also like to implement this into other games, starting with various 2d games. These games could be more or less advanced than Diep.io. Extending onto 2d games, we would like to implement this into a 3d game, which is far more complex, and would require a much more developed framework. One of the aspects of the game in which we have not involved our design is the upgrading of the tank. Again, due to time constraints, and the limitations of the game, we decided it was best not to implement upgrading the tank, and focus on just getting the tank to do more basic movements.

Introducing supervised learning would also be left up to future work, however, we would like a good mix of both supervised, and unsupervised training, as we still want to keep the robustness of the current implementation, but also implement a way to speed up the training so it isn't always doing it from scratch.

And finally, as mentioned above, we were unable to achieve the desired results, so a lot of the future work that needs to be done is to develop a much better framework for our implementation, as well as allow it to run without having

IX. References

[1] Chuankun Li, Pichao Wang, Shuang Wang, Yonghong Hou and Wanqing Li, ”Skeleton-based action recognition using LSTM and CNN,” 2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW), Hong Kong, 2017, pp. 585-590, doi: 10.1109/ICMEW.2017.8026287.

[2] Pengfei Sun, Pengju Liu, Qi Li, Chenxi Liu, Xiangling Lu, Ruochen Hao, Jinpeng Chen, ”DL-IDS: Extracting Features Using CNN-LSTM Hybrid Network for Intrusion Detection System”, Security and Communication Networks, vol. 2020, Article ID 8890306, 11 pages, 2020. https://doi.org/10.1155/2020/8890306

[3] Selvan, J.P. and Game, P.S., 2022. Playing a 2D Game Indefinitely using NEAT and Reinforcement Learning. arXiv preprint arXiv:2207.14140.

[4] Lohokare, A., Shah, A., Zyda, M. (2020). Deep Learning Bot for League of Legends. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 16(1), 322-324. https://doi.org/10.1609/aiide.v16i1.7449

[5] Lample, G. Chaplot, D. S. (2017) ‘Playing FPS Games with Deep Reinforcement Learning’, in Proceedings of the ... AAAI Conference on Artificial Intelligence. [Online]. 2017 p.

[6] N. Cole, S. J. Louis and C. Miles, ”Using a genetic algorithm to tune first-person shooter bots,” Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 139-145 Vol.1, doi: 10.1109/CEC.2004.1330849.

[7] Hidalgo-Bermúdez, R.M., Rodríguez-Domingo, M.S., Mora, A.M., García-Sánchez, P., Merelo, J.J., Fernández-Leiva, A.J. (2013). Evolutionary FSM-Based Agents for Playing Super Mario Game. In: Nicosia, G., Pardalos, P. (eds) Learning and Intelligent Optimization. LION 2013. Lecture Notes in Computer Science(), vol 7997. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-44973-4_39

[8] Perot, E. et al. (2017) ‘End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning’, in 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). [Online]. 2017 IEEE. pp. 474–475.

[9] Charoenkwan, P. et al. (2010) ‘A Study on Genetic Algorithm and Neural Network for Implementing Mini-Games’, in 2010 International Conference on Technologies and Applications of Artificial Intelligence. [Online]. 2010 IEEE. pp. 158–165.

[10] A. I. Esparcia-Alcázar, A. Martínez-García, A. Mora, J. J. Merelo and P. García-Sánchez, ”Controlling bots in a First Person Shooter game using genetic algorithms,” IEEE Congress on Evolutionary Computation, Barcelona, Spain, 2010, pp. 1-8, doi: 10.1109/CEC.2010.5586059.

[11] Tasdelen, A., Sen, B. A hybrid CNN-LSTM model for pre-miRNA classification. Sci Rep 11, 14125 (2021). https://doi.org/10.1038/s41598-021-93656-0

[12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau and Christian Gagné, “DEAP: Evolutionary Algorithms Made Easy”, Journal of Machine Learning Research, pp. 2171-2175, no 13, jul 2012.

[13] TensorFlow Developers. (2023). TensorFlow (v2.13.0-rc0). Zenodo. https://doi.org/10.5281/zenodo.7916447

Links

(1) Github repo:

(2) Training Video QR code: