A Novel, Supervised Learning Approach to Autonomous Agent Design within a Multi-Agent Game

Undergraduate Capstone Final Report

Ian Whitehouse

December 10th, 2023

Abstract

In this capstone project, I analyze the field of autonomous agents by reviewing the current literature and implementing my own novel framework for learnable autonomy using neural networks.

Originally defined by Stuart Russell and Peter Norvig in *Artificial Intelligence: A Modern Approach*, an autonomous agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [1]. Using this definition, numerous systems we rely on today can be thought of as intelligent agents. This includes systems we entrust with our lives, such as autopilot, lane-keeping assist, and crash-avoidance technologies. Over the next decade, machine learning systems are poised to continue to make breakthroughs in the development of intelligent agents. One area where these agents have already made an impact is the autonomous vehicle space, including autonomous cars, autonomous unmanned aerial vehicles, and autonomous ships.

In this report, I present my research, including a review of the current approaches to intelligent agents, the theory and design of my novel machine-learning system, and a summary of its application within a multi-agent game of tag. While the agent I built achieved only slightly positive results, I am confident in its theoretical underpinning and that I can continue applying the knowledge I gained throughout my academic career.

1 Introduction

This capstone project summarizes my semester-long research project within autonomous systems. Autonomy is a key problem within artificial intelligence, and this research project represents my work to explore and propose a novel solution within that landscape.

When I proposed this project early into the semester, I defined clear goals to ensure that the work could be accomplished in an organized manner. This report includes the deliverables established based on these goals. The first deliverable established was a literature review on the field of autonomy and reinforcement learning, which can be found in Section 2. The next deliverable was the theoretical design of my novel autonomous agent, which can be found in Section 3.1. The following deliverable was an overview of the broader impact of my capstone project, which is located in Section 3.3. Finally, the next-to-last deliverable, my experimental design, can be found in Section 3.2. This paper represents the last deliverable agreed to, which was an academic research paper highlighting my results.

While writing each of these sections, I developed an understanding of the long history of autonomous studies, beginning with the first chess-playing machines in 1951 [2]. This gave me important insight into the development of many historical autonomous agents and helped me plan the development of my own agent. I was also heavily inspired by the paper "A Path Towards Autonomous Machine Intelligence" written by Yann LeCun [3]. This paper lays out several key architectures for LeCun's vision of an autonomous system, which are detailed in Figure 2 and Figure 3. I made use of the main idea behind LeCun's Joint Embedding Predictive Architecture within my project. This architecture uses a predictive model to predict changes to encoded representations instead of raw input, simplifying its task. I combined this idea with a neural-network based cost calculator to create a model capable of choosing an action that will result in the lowest cost. This model was largely successful, and the results of applying it to a multi-agent environment are described in Section 5.

This paper presents the following structure: Section 2 describes the literature surrounding autonomy, intelligent agents, and reinforcement learning. Section 3 describes the design of the agent framework and the multi-agent test environment, as well as the broader impacts of this work. Section 4 describes the final implementation and training of the machine learning agent, and Section 5 describes the results of the agent within the multi-agent test environment. Finally, Section 6 summarizes the project, future research opportunities, and the knowledge gained throughout this semester.

2 Literature Review

2.1 Classic Intelligent Agent Design

The very first autonomous agents were designed in Manchester, England, in 1951 to play chess [2]. Chess was a fascination of the original theorists within autonomy, which can be seen in the fact that many of the original autonomy algorithms were applied. However, after Deep Blue beat Garry Kasparov in their 1997 game, autonomous agents were turned toward other tasks and games, including Go, where the reigning world champion was beaten in 2017 [4, 5]. While the agent that beat Go was developed using reinforcement learning,

the backbone of many autonomous systems, such as Stockfish, which is the current best chess-playing program, continues to be developed from algorithms originating before the development of machine learning [6]. Many of these original algorithms were an inspiration for my project.

Nash Equilibrium

Proposed by John Nash in "Non-Cooperative Games", the Nash Equilibrium is a method to solve non-cooperative, multi-agent games [7]. Numerous problems within autonomy could be considered multi-agent games. For example, multiple studies have viewed autonomous vehicle planning through the lens of game theory [8, 9]. The Nash Equilibrium is also a good way to analyze the success of an autonomous agent. For example, Sections 3.2 and 5 explore how, because the tagger agent in my tag game-based experimental environment is slightly faster than the taggee agent, optimal play will result in the tagger winning every game.

Minmax Planning

One of the original algorithms for decision-making is the Minmax algorithm [1]. The Minmax algorithm is based on the principle of minimizing the opponent's chance of victory while maximizing your own. This idea is a popular solution to numerous games, including tic-tac-toe, checker, and chess [6]. However, the Minmax algorithm requires that the agent can traverse a tree of all possible moves to a reasonable depth. While this was always possible in tic-tac-toe and is now possible in Chess because of modern hardware, it is impossible in games with continuous actions or with a large number of actions, such as Go.

Q-Learning

Q-learning is another important early algorithm for decision-making. In Q-learning, an agent attempts to maximize the expected value of its reward using a table of prior actions and rewards. As an early form of machine learning, Q-learning is trained by repeatedly performing actions and observing the reward of the resulting state, which leads to iterative improvement. The knowledge gained from this process is placed in the Q-table, which is used to choose future actions. It has been shown that, given enough time to train, Q-learning will generally converge on the optimal solution [10]. Similar to the Minmax algorithm, Q-learning suffers when presented with too many actions and does not work when dealing with a continuous set of actions. In addition, Q-learning suffers from many of the flaws that affect other reinforcement learning agents, such as a reliance on the ability to repeatedly replay a scenario.

2.2 Reinforcement Learning

The most popular modern approach to the problem of autonomous agents within machine learning is reinforcement learning. Reinforcement learning has been called the holly grail of machine learning because it has been able to effectively solve problems believed to be impossible for other machine learning disciplines or decision-making algorithms [11]. The most

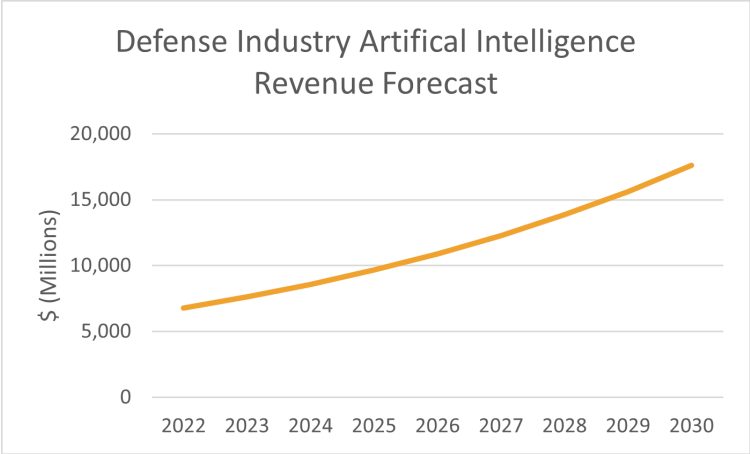

Figure 1: This figure shows the growth of the revenue within the Defense industry for artificial intelligence, which is projected to grow from $6.8 Billion to $17.6 Billion by 2025 [16]

famous of these tasks, the game Go, was famously beaten by Google DeepMind's reinforcement learning agent in 2017 [5]. Go was believed to be unbeatable due to the massive number of possible game states; however, DeepMind's reinforcement learning algorithm was able to beat the top human player through iterative improvements gained through self-play. Reinforcement learning agents are able to perform complex tasks within realistic environments, making them notable for their real-world applications. Reinforcement learning algorithms have been applied to include Atari games [12], stock trading [13], and simulated dogfighting [14].

However, there are known issues with reinforcement learning. Dulac-Arnold et al. lays out nine key challenges to the adoption of reinforcement learning within a real-world system [15]. These include problems that were also a challenge for the development of my model, including reward function design (number 6), and challenges that I attempted to address, such as effectively dealing with stochastic systems (number 5).

A major problem within reinforcement learning stems from the amount of data required to train the model. Yann LeCun argues that reinforcement learning is inherently inefficient because the models require thousands of trials to learn the obvious consequence of their actions, while a human requires many fewer observations [3]. This is a challenge for training a reinforcement learning model because, for them to learn a complex simulation, the agent must repeatedly play through the simulation, which is potentially a very costly operation. I also attempted to remedy this problem when developing my model, which is trained on very little data.

2.3 Applications

Evidently, there are numerous applications of autonomous agents. The largest category of these is autonomous vehicles. As shown in Figure 1, one of the largest customers for these systems is the military, where demand is likely to grow quickly over the next decade. This is also evident through AlphaDogfight, which was a military competition where autonomous

agents successfully beat a human fighter pilot in multiple simulated dogfights. [14]. Within civilian applications, autonomous cars are becoming increasingly common. As the market for both civilian and military autonomy grows, research into autonomous systems will become more essential, as many areas within the field are still in their infancy.

3 Project Design

3.1 Requirements of an Intelligent Agent

According to Russell and Norvig in *Artificial Intelligence: A Modern Approach*, an autonomous agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [1]. They furthermore break down autonomous agents into rational agents that "[act] so as to maximize the expected [utility] of a performance measure based on past experience and knowledge". This is the prime directive of my research: the development of an agent that performs well in a variety of different scenarios simply by learning to take actions that maximize its expected return.

To effectively accomplish this goal within a multi-agent environment, the model must be able to

- Know its current utility
- Estimate how its actions affect its current utility
- Estimate how its opponent's actions affect its current utility
- Combine these estimates to calculate the expected utility of each possible action, and then choose the action to take so as to maximize this utility

In Section 4, I discuss how the model I developed solves this by integrating a predictive neural network and a neural network capable of calculating the current utility.

Inspiration: "A Path Towards Autonomous Machine Intelligence" [3]

In "A Path Towards Autonomous Machine Intelligence", Yann LeCun lays out his vision for a machine learning system, which is shown in Figure 2. His proposal includes a perception module that estimates the current state of the world, a world model that predicts the future states of the world, a cost module that measures the discomfort of the agent at the current state, a memory module that learns to associate different states with their costs, and an actor module, which calculates possible actions that can minimize the agent's discomfort. As discussed in Section 4, I took inspiration from these ideas when developing my model, which includes elements present within the perception, cost, actor, and world model modules.

One key architectural element of "A Path Towards Autonomous Machine Intelligence" is the importance of the Joint-Embedding Predictive Architecture for predictive models. This potentially simplifies the model's prediction task by using an encoder to focus only on the important details from the input. This potentially frees the predictor from wasting its capabilities on predicting unnecessary details within the raw inputs. This architecture was implemented by a paper by Meta [17].

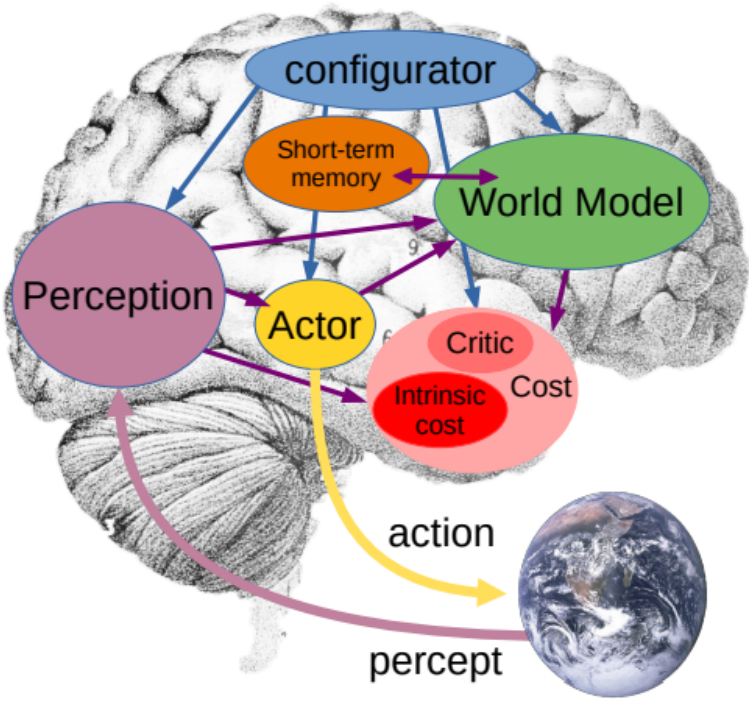

Figure 2: Yann Lecun's proposed architecture for autonomous intelligence [3]. In this project, I have attempted to implement the perception, actor, cost, and world-model modules.

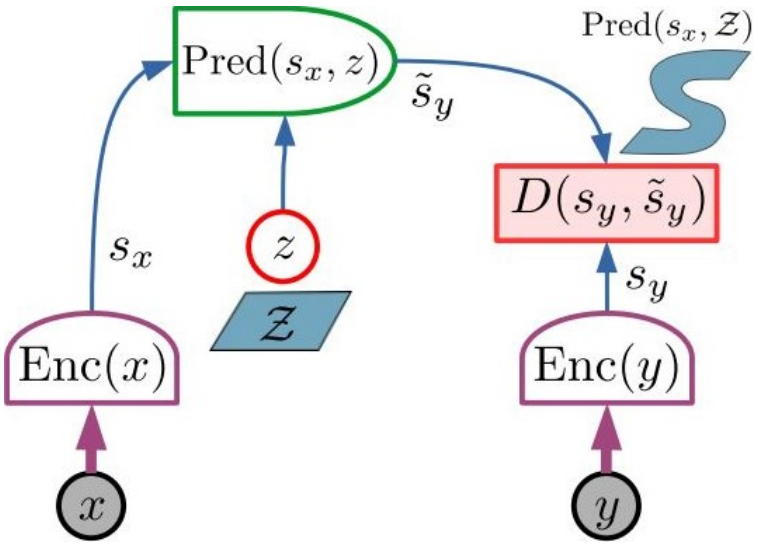

Figure 3: Joint Embedding Predictive Architecture, as proposed by Yann LeCun [3]. In this architecture, the encoder generates latent representations that only include necessary information. The predictor, then, learns to predict these representations instead of the raw stimulus, which potentially contains noise and unnecessary details. This approach simplifies the predictor's job and improves its adaptability

In addition, LeCun proposes three challenges that must be solved for machines to be intelligent. The first, "How can machines learn to represent the world, learn to predict, and learn to act largely by observation?", I propose an answer to within this project. As discussed in Section 4, the model is able to learn to represent the world from visual stimuli alone, and acts based on predictions it makes from this stimulus.

3.2 Experimental Design: Multi-Agent Tag Game

In addition to my autonomous agent, I developed a test environment for the intelligent agent to engage with. When beginning the project, I considered training the autonomous agent to complete a task in a video game, such as level flight within *Ace Combat 7* or driving within *Grand Theft Auto V*. These games provide complex simulated environments, which is why many researchers consider them the perfect environment for studying reinforcement learning. However, these environments lack a simple API I could use to communicate with their world. In addition, they are limited in how fast their simulations can run, which would hamper the data collection required for this project. While these flaws could be fixed by modding the games, I felt that this would be outside the scope of this project and could take valuable time from the development of the intelligent agent.

Other environments, such as the Gymnasium platform [1], exist specifically for reinforcement learning environments. These environments do not have limits on their simulation speed and are built to be controlled by other programs; however, I believed that a custom-built environment would be simpler to work with and provide more flexibility. I look forward to expanding this project after later by adapting it to some of the simulations within the Gymnasium environment.

The simulation environment designed for this project is a multi-agent game of tag, where one player, the tagger, attempts to reach the other player, the taggee. If the tagger reaches the taggee, the taggee hits an obstacle, or the taggee moves out-of-bounds, the tagger wins the game. If the tagger hits an obstacle or runs out-of-bounds, the taggee wins the game. The game includes randomly placed and sized obstacles that the players must avoid. The players start the game in random positions. The tagger is slightly faster than the taggee, which implies that, if each player is playing perfectly, the tagger will win every game.

The simulation is styled like the game *Spacewar*, with simple vector-based graphics and bright colors. This is shown in Figure 4. The graphics shown are passed directly to the visual input of the agent model. I purposely made the graphics simple to reduce computational complexity when collecting data, as the simple graphics require very little time to render. In the future, I plan to apply this agent to more visually complex simulations. An additional benefit of a simple training environment is the ability to represent the graphical output of the game as a sparse array. Without this ability, it would be impossible to store the data required for training on the machine I used.

[1] `https://gymnasium.farama.org/`

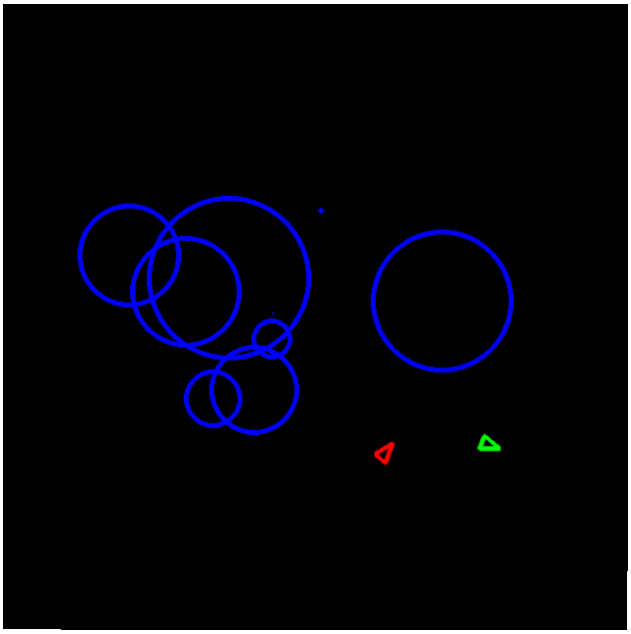

Figure 4: Screenshot the the simulation environment. The tagger is represented by the red triangle, while the taggee is represented by the green triangle. Obstacles are represented by the blue circles

3.3 Feasibility and Broader Impacts

Economic Feasibility and Impact

This research was economically possible because of the technology provided by the American University Computer Science Department. Because of the size of the models, it would have been impossible to train them without access to high-powered and expensive GPUs.

The potential economic impact of autonomous systems is impossible to overstate. While the models I trained were designed to play a simple game of tag, the real-world value of autonomy research is upwards of trillions of dollars. One investment firm, Ark Investment Management, believes that autonomous taxis could affect the total gross domestic product by more than $23 billion [18]. Because autonomous taxis represent just one of the many applications of intelligent agents, this example shows that the potential economic impact of research into autonomous systems is limitless.

Environmental Impacts

There are few environmental impacts to this project. The only environmental impact directly attributable to this project is the energy requirements of the GPUs used to train the model.

As autonomous agents continue to be developed, the energy and water required to train them poses an environmental concern. According to an article from the Associated Press, each query to OpenAi's GPT-4 model uses 500 milliliters of water [19]. While this is an alarming amount, as the technology is developed, it will likely become more efficient, decreasing its environmental impact. In addition, by operating self-driving electric vehicles or

helping manage smart electricity grids, autonomous agents can potentially help make up for this waste by making the systems we rely on more efficient.

Manufacture Feasibility

This project does not rely on manufactured parts, hence this is not applicable when discussing its feasibility.

Technical Feasibility

As discussed above, the main element of the technical feasibility of my project was the access to GPUs provided by the American University Computer Science Department, which was instrumental in the training of my models.

My project was implemented using the TensorFlow[2] framework, and the multi-agent simulation was built within pygame[3]. While the project was very technically complex, these libraries provided helpful abstractions that greatly simplified the project.

Ethical Implications

Unfortunately, there are significant ethical concerns within the study of autonomous systems. The main ethical consideration is the use of autonomous systems within warfare, which is outside of the scope of this project but still merits careful consideration.

When considering autonomous systems in warfare, a central ethical concern is the desensitization of enemy and civilian casualties as it becomes easier to outsource missions to machines, which separates soldiers from the consequences of their actions. This was originally a concern with unmanned aerial vehicles. In his 2002 essay "The Effects of Technology on Our Humanity", former Army Chaplain Keith Shurtleff argued that, because drone operators are safely away from the conflict, "[they] are removed from the horrors of war and see the enemy not as humans but as blips on a screen" which puts them in "a very real danger of losing the deterrent that such horrors provide" [20]. This danger is magnified by autonomous weapons, which further separates their operators from the battlefield. While autonomous systems will inevitably be applied to warfare, it will be necessary that countries strictly define and enforce rules for the identification and execution of targets to minimize civilian casualties and to ensure that enemy combatants are treated with dignity and respect.

Political Impact

There are very few political impacts to this project, short of the political implications of the ethical impacts discussed above.

[2] `https://TensorFlow.org`
[3] `https://pygame.org`

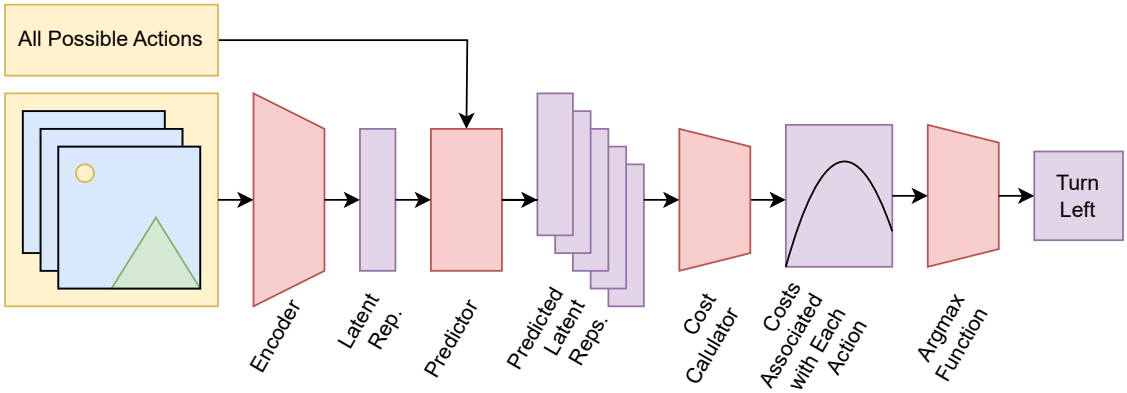

Figure 5: This figure shows the inference pipeline of the model. To estimate the best possible move, the predictor predicts the future cost for each action in a subset of all of the possible actions. Then, the model uses the average of the predicted costs for each of the actions, and the agent takes the action whose moves result in the lowest estimated costs.

4 Implementation and Training of the Intelligent Agent

As discussed in Section 3, this project focused on the development of a supervised machine learning model-based intelligent agent. This section details the design of the model, the training process I used, and the importance of data preprocessing, cost function design, and different hyper-parameters to the model.

4.1 Machine Learning Model Structure

The machine learning model developed for this project includes three neural networks: the encoder, the predictor, and the cost calculator. These models are arranged in a similar manner to the Joint Embedding Predictive Architecture (JEPA), which is shown in Figure 3.

Similar to the JEPA model, the encoder is central to the model and is designed to compress a full-color image into a format that only retains the information required by the cost calculator and predictor. The encoder includes multiple layers of dense convolutional networks, based on the ones discussed by Jégou et al. [21]. Convolutional neural networks have been shown to perform well with images. The outputs of these networks are flattened and used as input to fully connected layers, which decrease in size by factors of two until reaching the size of the latent representation, which is a hyper-parameter. By combining both convolutional and fully connected layers, this structure is ideal for identifying the essential information within an image, and for converting it into a usable format. The size of the latent representation is an area that I have not had enough time to delve further into. When choosing this hyper-parameter, it is essential that it is large enough to hold the information used to predict the future movements of the agent and the other agents, as well as the information required to calculate the agent's current cost accurately. However, the size of this representation should still be minimized to avoid providing unnecessary detail to the other neural networks, which could harm their accuracy during the training process.

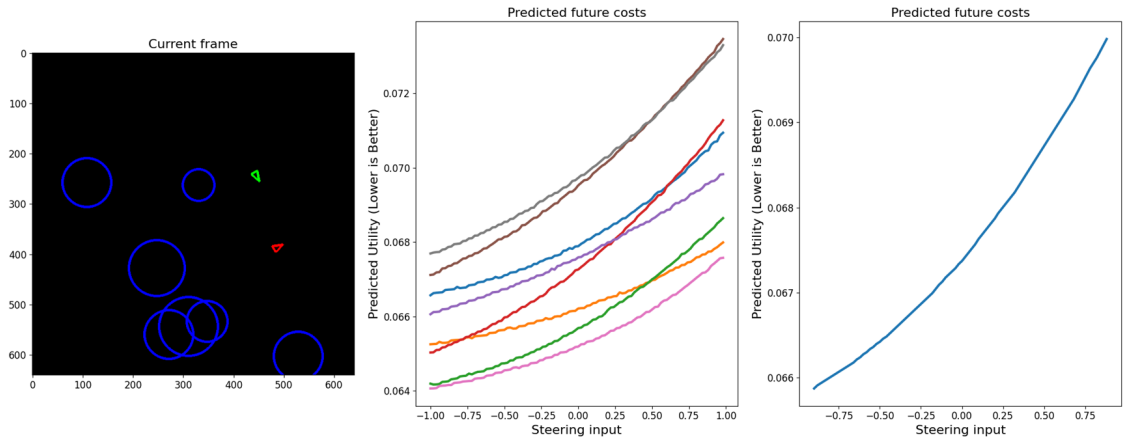

Figure 6: This figure shows the decision-making process of the tagger model. On the far left, the current frame can be seen, with the taggee to the tagger's left. Then, on the center panel, all of the predicted costs, based on the action taken (X-axis, negative numbers represent left turns), are displayed. Finally, the predicted costs are averaged in the far-right column, which shows that the cost is minimized by turning left, which is towards the taggee.

The predictor's role in the autonomous agent is to, when given the current latent representation generated by the encoder, predict the latent representation of a future frame of the scenario. The predictor is a fully connected neural network that takes a single latent representation and outputs multiple latent representations. The number of frames into the future the predictor predicts and the number of predictions it generates are hyper-parameters that massively affect the model's planning and reasoning capabilities. From a theoretical perspective, the agent's performance should improve as it generates more predictions and predicts further into the future. It is also important to note that, as the predictor is tasked with predicting further into the future, the number of predictions it generates should increase exponentially to maintain the fidelity of its predictions. During inference, the predictor is fed a subset of all of the possible moves that the agent could take, along with the current frame, and the costs associated with its predictions are averaged and used to determine the best action. This process is shown in Figure 5, and Figure 6 shows the effects of running this method on an actual frame. A theoretical drawback of this approach is that the predictor must make predictions based on a single action, while the prediction accuracy will be affected by all of the actions taken between when the prediction is made and when the prediction is for. A better structure for this problem could involve the model predicting the outcome of a continuous set of actions instead of a single action.

Finally, the cost calculator uses the latent representations generated by the encoder and predictor to estimate the cost associated with that representation. The real cost at a frame is calculated using the method discussed in section 4.2. The cost calculator is a fully connected layer and must be trained to be highly accurate because it must be able to distinguish between small differences within the latent representation. Inaccuracies caused by the cost calculator have a massive effect on the action chosen during inference, even when those inaccuracies are minor.

4.2 Cost Function Design

Like in reinforcement learning, the design of the cost function is incredibly important for the model's performance. In this model, the cost function is calculated every frame, which is unlike many reinforcement-learning algorithms when the reward is only calculated after a scenario's completion. Therefore, in my scenario, the base of the cost function is the Euclidean distance distance between the tagger and the taggee. For the tagger, it is calculated

$$cost(tagger_x, tagger_y, taggee_x, taggee_y) = \frac{\sqrt{(tagger_x - taggee_x)^2 + (tagger_y - taggee_y)^2}}{\delta} \tag{1}$$

For the taggee it is calculated

$$cost(tagger_x, tagger_y, taggee_x, taggee_y) = \frac{\delta - \sqrt{(tagger_x - taggee_x)^2 + (tagger_y - taggee_y)^2}}{\delta} \tag{2}$$

In these formulas, the Euclidean distance is normalized with δ, which is the maximum possible distance between the agents within the scenario.

However, these equations fail to consider the agent moving out of bounds or encountering an obstacle. Therefore, when training on scenarios that have already occurred, the cost functions of scenarios that end with the agent losing are augmented. To do this, the difference between the 1 and the cost five frames before the negative outcome is taken. This difference is divided by five and added to each of the five frames before the negative outcome as to linearly interpolate it. Therefore, the cost function for this project can be abstracted as

$$cost = cost\ based\ on\ current\ instance + demerit\ for\ negative\ outcome \tag{3}$$

I believe that this design of cost function can be applied to many different problems. For example, a lane-keeping assist system could be thought of through this lens as the combination of the distance to the center of the lane (cost of the current instance) and a demerit for leaving the lane. Similarly, an automatic stock trading portfolio could have a cost based on the return of the portfolio, with demerits for selling too early.

I also want to experiment with cost functions that include only the current instance and cost functions that are linear interpolations of the end state. I believe that this could be a good way to allow for more organic learning to occur within my model.

4.3 Training Process

The process for training my autonomous agent includes three different stages, all of which are designed to ensure that the encoder, cost-calculator, and predictor are trained to work together and be maximally effective. Each agent is trained based on samples from previous generations of the agent, which is discussed more in Section 5.

In the first stage of training, all three neural networks are trained together. During each epoch, two different training processes are run back-to-back. In the first process, the encoder and cost calculator are trained with batches of images to calculate the cost of the current frame. This process is shown in Figure 7. Then, in the second process, the weights of the

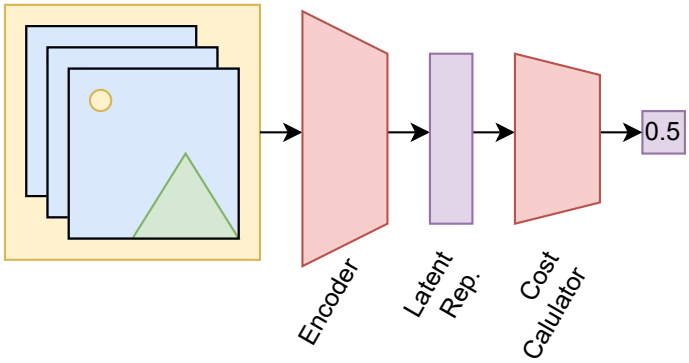

Figure 7: This figure shows the first part of each epoch of the training process. In this step, the encoder is trained to represent the frame input in a useful format for the cost calculator. This step uses MSE to compare the output of the cost calculator to the real cost in each example.

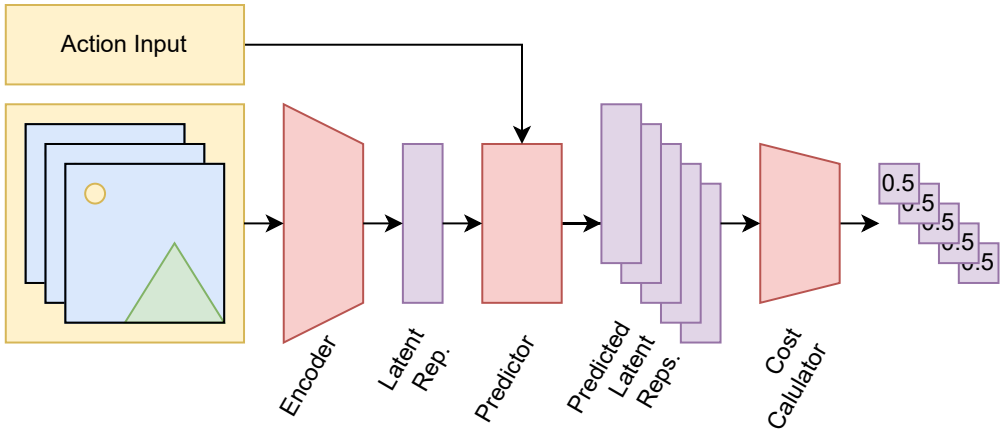

Figure 8: This figure shows the second part of each epoch of the training process. In this step, the predictor uses the encoder's latent representations and the keyboard input at that frame to predict multiple future encodings. The cost calculator uses this to predict the future cost. Because the predictor generates multiple predictions, the minimum mean squared error algorithm discussed in Section 4.3 is used as the loss function.

cost calculator are frozen, and the encoder and predictor are trained on batches of images and actions to calculate the cost of a future frame. This process is shown in Figure 8. In addition, because the predictor is incentivized to make multiple predictions, its loss function is based on Minimum Mean Squared Error, which is discussed in Section 4.3. To prevent either process from overfitting, all three models are saved only when both processes achieve a new lowest loss value, and these saved models are loaded before beginning the next stage of training. After this stage of training, the encoder is considered complete, and its weights are frozen.

During the next stage of training, the cost calculator is further refined. To do this, images are fed through the frozen encoder, and the cost calculator is fit on the latent representation, allowing it to further improve its score. This is effective because, previously, the cost calculator's weights were only saved if the predictor also improved. Without this constraint, the cost calculator is able to fully utilize the latent representation, and it significantly improves compared to when it is trained simultaneously with the predictor. After this stage, the weights of the cost calculator are frozen.

In the final stage of training, the predictor is optimized. Using latent representations from the frozen encoder, the predictor generates predicted latent representations, which are passed to the frozen cost calculator. This stage is enormously rewarding for the predictor's validation loss, as it learns to take advantage of the improvements to the cost calculator. This is the final stage of training, after which the agent's training is considered complete.

Minimum Mean Squared Error

In the development of the predictor model, it became evident that the predictor would need to be able to make a diverse set of predictions for each frame. Without this ability, the predictor's predictions would be too general and would lack the fidelity needed to make small decisions, which is discussed in Section 4.1. However, when training the predictor, I needed a method to reward the predictor for making each prediction different, which led to the development of minimum mean squared error (MMSE). To calculate the MMSE, the mean squared error of the cost of the prediction and the actual future cost are calculated, and then only the minimum MSE is used as the predictor's loss. This means that the predictor would be rewarded by generating a set of predictions accounting for vastly different outcomes, even if some of the predictions are inaccurate.

However, because the future scores ranged between 0 and 1, the predictor began evenly distributing its predictions between these numbers. Because of that, the predictor's loss was updated to include MMSE and MSE, which were added together. The weight of each loss is an additional hyper-parameter, which is discussed at length in Section 4.5.

4.4 Data Preprocessing

The data preprocessing required to train this agent is minimal, which is by design, as the agent is built to engage in simulations using visual input. To allow for training on larger batches, the data was cast to half-precision floats. The scores were normalized, first within a RobustScaler, and then recentered to be between 0 and 1 [22]. Finally, during training, the images presented to the encoder were randomly flipped, helping to ensure generalizability.

This step may not work for many simulations; however, because the experiment discussed in Section 3.2 is symmetrical, flipping does not lead to unrealistic inputs.

4.5 Hyper-parameters

The largest challenge to the development of this model was turning the hyper-parameters that control the model's creation, training, and data preprocessing.

When the model is initialized, the main hyper-parameters include the number of parameters at each layer of the neural network, the number of layers, the number of predictions that the predictor generates, and the dropout strength. The number of predictions and the dropout have the biggest effects on the success of the agent. Allowing the model to make additional predictions seems to increase its effectiveness, but, if the number of predictions is too high, the model doesn't effectively tailor its predictions, instead relying on the sheer number to be accurate. The dropout of the predictor also has a massive effect. When set too low, the fidelity of the predictions is too high, causing the predicted costs to fluctuate over the set of possible actions; however, when too high, the model fails to make accurate predictions.

During the training of the model, the main hyper-parameters are the learning rate at each stage of training and the setup of the combined loss function, discussed in section 4.3. To manage the learning rate, I used a higher one during the initial combined training and then lowered the learning rate when training the cost calculator and predictor separately. This solution seems to be effective, allowing the model to be trained efficiently to a very low loss level. Setting up the combined loss function required one hyper-parameter: the weight given to the regular MSE compared to the minimum MSE. When set too low, the predictor generates a wide range of predicted costs, which shows that it isn't being challenged to generate useful predictions. When set too high, the predictor generates identical predictions, which defeats the purpose of creating multiple predictions.

Finally, the data preprocessing stage has a single hyper-parameter that must be set: the cost associated with losing a game. This part of the cost function, as discussed in Section 4.2, is supposed to prevent the agent from hitting obstacles or running out-of-bounds. However, when loading the training data, it is important to make sure that this cost is not an outlier, causing the agent to care little about its distance to the other agent, while making sure that this cost is still prominent enough for the agent to care to avoid obstacles or boundaries.

The time it takes to train the model makes hyper-parameter tuning strategies too complicated to perform, meaning that they have to be largely set based on previous training runs. This also made it difficult to scientifically measure the effects of each hyper-parameter.

5 Experimental Results

The autonomous agent was tested based on its ability to succeed within the experiment discussed in Section 3.2. The experiments were constructed based on generations, where each generation was the result of an agent trained on the previous generation. The first generation, Generation 0, made entirely random moves.

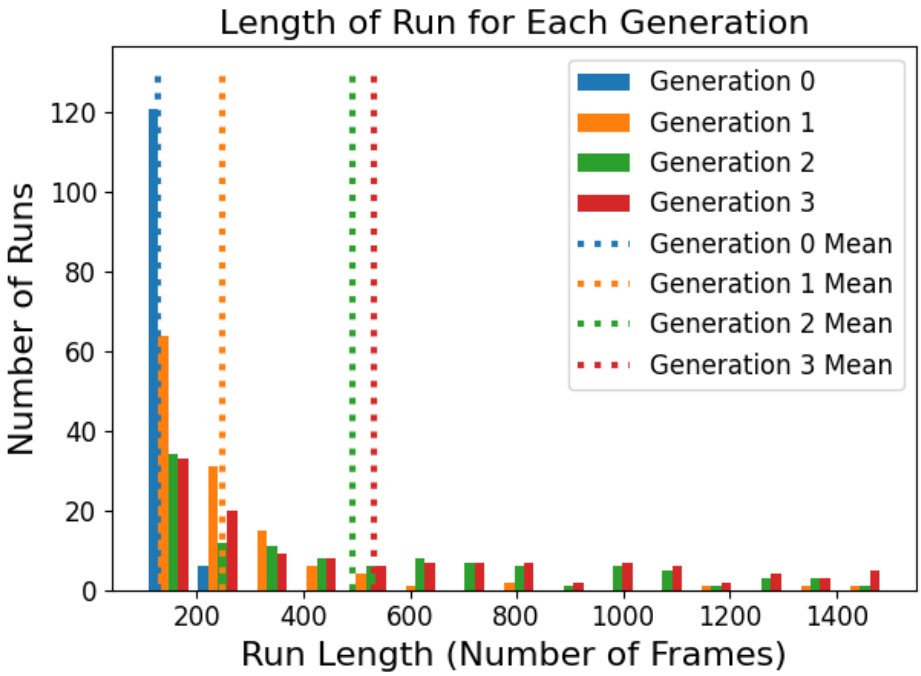

Figure 9: This figure shows a histogram of the lengths of each run completed by the autonomous agents. The dotted line shows the average run for that agent, showing that each generation lives longer than the previous

Because the tagger model was slightly faster than the taggee, if the autonomous agent training is ideal, it would be expected that

- The simulation lasts longer for the current generation than previous generations
- The tagger tags the other player more in the current generation than in previous generations

From Figure 9, it is clear that each generation lived longer than the previous generation, which is a sign of success. However, the results are less clear in Figure 10. While the number of runs that ended with the tagger tagging the taggee increased between Generation 0 and Generation 1 and Generation 2 and Generation 3, it did not increase between Generation 1 and Generation 2. In addition, it remained a minority of the reasons for a run ending. In Generations 0, 1, and 2, the plurality of runs ended from the tagger colliding with an obstacle, which is potentially a sign that it was so aggressive in pursuing the taggee that it ignored the obstacles. if this is the case, it is a cause for concern, as it shows that the cost function discussed in Section 4.2 was not properly weighted.

This agent was also able to learn from fewer training examples than a traditional reinforcement learning agent. For example, Generation 1 was trained on only 9000 frames of gameplay, which is less than 8 minutes of total time. This is important because the amount of data required is a major flaw within reinforcement learning that I wanted to address within my research.

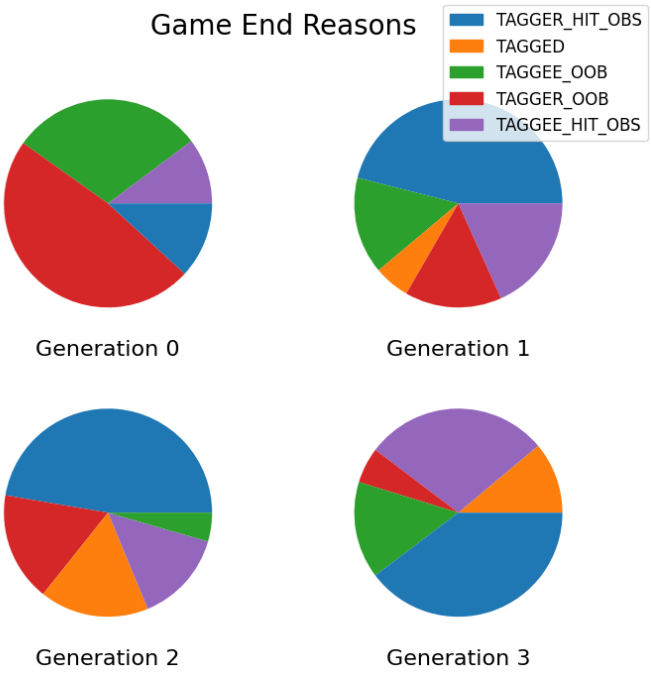

Figure 10: This figure shows what ended each run. OOB stands for out of bounds and was the cause of the majority of run endings in Generation 0. HIT_OBS stands for hit obstacle and was the main cause of runs ending in all other generations

Qualitatively, Generation 1 had the best performance, and, from recordings of the agent's actions, there were clear signs of intelligent behavior displayed by both the tagger and the taggee. However, in later generations, these signs of intelligence were less clear, and the agents would end up stuck going in perpetual circles. To address this, I introduced more randomness into their movements; however, they still never showed the intelligence that Generation 1 showed.

Future research is necessary to continue improving on this model. While this model has a strong theoretical basis, it is currently too complex to be widely used. One pressing area of research is in simplifying the model and analyzing the effect of each hyper-parameter to understand its importance. The model's major flaw is its complexity, which limits how it can be applied and makes it more difficult to train. I also look forward to applying this model to additional problems and training it for longer to see if it reaches the Nash Equilibrium, which would mean that the tagger always wins the scenario by tagging its opponent.

6 Conclusion

In this report, I presented the results of my capstone project, which focused on the development of an autonomous agent, which I applied to a simple, simulated game of tag. This agent builds on much of the current research into autonomy, especially the research by Yann LeCun in "A Path Towards Autonomous Machine Intelligence". The agent is made up of three neural networks: the encoder, the cost calculator, and the predictor. These networks

are trained (See Figures 7 and 8) and inferenced together (See Figure 5). During training, they learn to estimate and predict the cost values associated with different game states in order to choose an action that minimizes their future cost, which builds on much of the literature discussed in Section 2.

Through my experimentation, the results were largely positive, and quantitatively, the model appears to be improving toward optimal decision-making. However, the model is still hard to train and often ends up turning in circles. Within my future work on this project, I hope to make training more efficient by better understanding the hyper-parameters of the model. After doing this, I will be able to run additional experiments, allowing me to further improve the autonomous agent. I am also excited to apply the agent within other domains, especially video games. While the agent is currently a long way away from this point, I believe that it will be an exciting way to prove that it is effective.

Autonomy is a wide and important field within computing, and I am confident that I will continue applying the skills I learned while completing this project to my future work.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2020.

[2] B. J. Copeland, "The Modern History of Computing," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, Winter 2020 ed., 2020.

[3] Y. LeCun, "A path towards autonomous machine intelligence." `https://openreview.net/pdf?id=BZ5a1r-kVsf`, 2022.

[4] L. Greenmeier, "20 years after deep blue: How ai has advanced since conquering chess." `https://www.scientificamerican.com/article/20-years-after-deep-blue-how-ai-has-advanced-since-conquering-chess/`, 2017.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[6] The Stockfish developers. `https://stockfishchess.org/`, 2023. Accessed: 2023-12-10.

[7] J. Nash, "Non-cooperative games," *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[8] N. Smirnov, Y. Liu, A. Validi, W. Morales-Alvarez, and C. Olaverri-Monreal, "A game theory-based approach for modeling autonomous vehicle behavior in congested, urban lane-changing scenarios," *Sensors*, vol. 21, no. 4, 2021.

[9] M. Wang, Z. Wang, J. Talbot, J. C. Gerdes, and M. Schwager, "Game-Theoretic Planning for Self-Driving Cars in Multivehicle Competitive Scenarios," *IEEE Transactions on Robotics*, 2021.

[10] M. T. Regehr and A. Ayoub, "An elementary proof that q-learning converges almost surely," 2021.

[11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," 1996.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[13] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep reinforcement learning for automated stock trading: An ensemble strategy," in *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20, (New York, NY, USA), Association for Computing Machinery, 2021.

[14] Defense Advanced Projects Research Agency. `https://www.darpa.mil/news-events/2020-08-26`, 2020. Accessed: 2023-12-05.

[15] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," 2019.

[16] Vantage Market Research, "Artificial intelligence in military market - global industry assessment & forecast," 2022.

[17] M. Assran, Q. Duval, I. Misra, P. Bojanowski, P. Vincent, M. Rabbat, Y. LeCun, and N. Ballas, "Self-supervised learning from images with a joint-embedding predictive architecture," 2023.

[18] T. Keeney, "Autonomous taxis may have the most impact on gdp of any innovation in history." `https://ark-invest.com/articles/analyst-research/autonomous-taxis-gdp-impact/`, 2023.

[19] M. O'Brien and H. Fingerhut, "Artificial intelligence technology behind chatgpt was built in iowa — with a lot of water," *The Associated Press*.

[20] K. Shurtleff, "The effects of techonology on our humanity." `https://web.archive.org/web/20121224030147/https://www.carlisle.army.mil/usawc/parameters/articles/02summer/shurtlef.htm`, 2002.

[21] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," 2017.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.