Developing a Hybrid Solution to Point the Exploration of Artificial Intelligence Algorithms Learning Locomotion using Motion-Captured Data

Abstract:

Background: Artificial intelligence learns to walk, with few exceptions, by taking action, observing the outcome, and fine-tuning the decision-making process to choose actions that lead to the best outcome without prior knowledge. ***Goal:*** To develop an artificial intelligence system that reduces exploration time and learns to walk more efficiently and reliably in a two-dimensional virtual environment using a bipedal robot. If successful, this algorithm will be transferable to a three-dimensional robot in the real world. ***Methods:*** From a video of a person walking, the angles of leg joints at each frame were converted to corresponding motor-control input to find the difference between frames to reach the angles in the subsequent frame. I then created a two-part algorithm. The first part evolved a neural network with the Neuro-Evolutionary-Augmented-Topologies (NEAT) algorithm to learn motor outputs resembling human movements. The second part used the Deep Deterministic Policy Gradient (DDPG) algorithm to generalize and fine-tune the learned movement in virtually simulated rough terrain. ***Results:*** The developed algorithm started the second step with a better grasp of walking, but quickly plateaued in learning. However, an algorithm without video which only uses DDPG does not plateau but rather continues to learn given all inputs from the environment. ***Conclusion:*** Although the exploratory phase was optimized, a plateau in learning was observed. I hypothesize that additional input, beyond motor joint angles, could serve as a solution to the plateau by giving the algorithm more inputs to learn from. The research, therefore, concludes that pre-trained AI can learn quicker but needs to mesh with a novel algorithm to incorporate further input from the environment.

Table of Contents

1. Introduction

As early as the 1960s, researchers have been exploring robotics aimed at mimicking human bipedal locomotion[1]. Early attempts at creating walking robots involved programming a strict set of rules for their movement. This approach had limitations in its ability to adapt to new or unfamiliar environments, as it lacked the capacity to generalize beyond the specific conditions it had been programmed to follow. As computing power and hardware capability increased in the early 2010s, the set of rules became more diversified, increasing performance through difficult terrain[2]. However, the generalization of a set of instructions to any environment cannot occur without an artificial intelligence capable of complex decision-making.

The process of accumulating knowledge and experience to walk can be augmented, just as in the case of a child learning to walk, where outside factors play a large role in learning. As a baby learns to walk the parents play a vital role in helping them build skills such as muscle control, balance, and coordination[3]. Parents do this through their own walking, which that babies learn to mimic; rapidly accelerating the learning process. Conversely, by taking away a stimulus for growth (ie. a parenting example) it is reasonable to expect subpar performance during the learning process. Yet, learning without example is how artificial intelligence algorithms currently approach learning locomotion with a few exceptions.

This research aims to fill the gap by providing an example for artificial intelligence to learn locomotion and then apply the system to a virtual environment to generalize the learning. By doing so, I hope to create an AI system that learns locomotion more efficiently and reliably. The following literature review discusses the conceptual understanding necessary to appreciate this hybrid solution.

2. Literature review

2.1 Overview of artificial intelligence

Artificial intelligence (AI) is "a field, which combines computer science and robust datasets, to enable problem-solving"[4] (Figure 1). Artificial intelligence is a term encompassing programming intended for doing tasks requiring human intelligence.

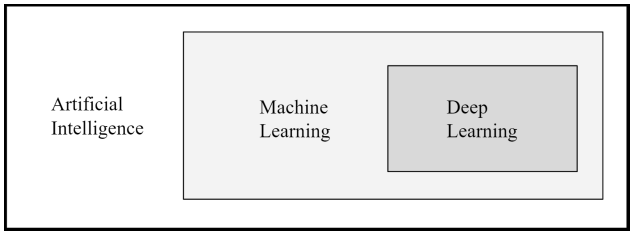

Figure 1: The relationship between the subfields of AI. Artificial Intelligence is a term containing machine learning and deep learning exploring more narrow methods of simulating intelligence.

Under the umbrella term of AI lies the subfield of Machine Learning (Figure 1). "Machine learning is a branch of artificial intelligence which focuses on the use of data and algorithms to imitate the way that

humans learn, gradually improving its accuracy[5]", and under that lays deep learning, which is where my algorithm lays.

"Deep learning attempts to mimic the human brain—albeit far from matching its ability—enabling systems to cluster data and make predictions with incredible accuracy"[7]. Neural networks are the fundamental concept in attempting to replicate a small scale of the brain's neuron topology to mimic complex decision-making.

The next section of this literature review will go more in-depth surrounding how networks work fundamentally, but for now, they can be thought of as a black box (Figure 3). Some data gets passed into the box, and some data gets outputted. In the context of humans, various data can be passed into our brains, while strictly numerical data can be passed into neural networks, and strictly numerical data is outputted.

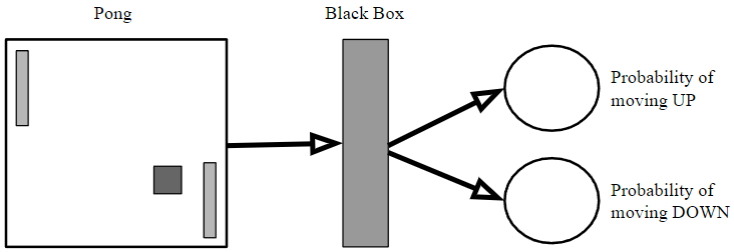

Figure 3: Practical example of network input and output. The network represented by the black box has an input of the raw pixels passed into it, and the probability of moving the paddle up and down gets returned as an output.

Deep learning can be used to choose actions for AI, for example in an environment such as pong. In the game of pong, a ball is bounced between opposing players' paddles, and whichever side allows the ball to pass their paddle loses. In order to be successful at the game the player must comprehend the observable space through pixels and how actions affect the outcome. Due to the large observation space, a neural network is used.

A network can be a very effective tool for increasing decision-making capability by fine-tuning the black box to better output an appropriate action. In the case of pong, the raw pixels on the screen can be passed into the box, and in turn, a move probability is outputted of moving the paddle up or down (Figure 2). Over thousands of games, the correlation between the inputted pixels and more high-performing actions is observed by the network through the use of a chosen algorithm, and the internal connections between neurons are fine-tuned, leading to perpetually increasing performance. Neural networks can be an extremely powerful tool. In the following section, the inner workings will be further discussed.

2.2 Technical explanation of neural networks

A neural network is a flexible tool used to recreate the capability of a small subsection of our brain's neuron topology. Input is passed into the network and processed, with the output being produced. To illustrate how neural networks work better, an in-depth example is described below.

Imagine that you are visiting a country for the first time and you notice that the traffic lights are different than in North America. Instead of a light being used for directing vehicular traffic and a separate one for pedestrian traffic, this country uses a single traffic light to direct all traffic. The lights however do not turn on one at a time, but instead in combinations. Intrigued by this new system you sit down and start to record the combinations of lights you see, and what action the pedestrians subsequently take, assuming they follow the rules.

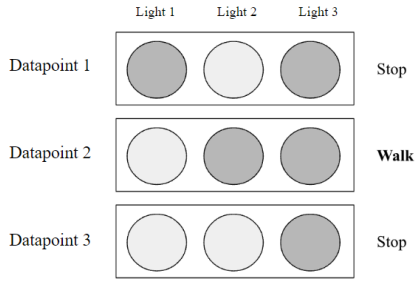

Figure 4: Traffic light configuration and following action recorded. The data shows the recorded configuration and the following action. The darker shade of gray indicates the light being turned on.

It may not be clear initially what the correlation between light configuration and action, so a neural network can be used. By training a neural network it is possible to identify the pattern present within the initial recorded data and then generalize the understanding to new data collected in the future.

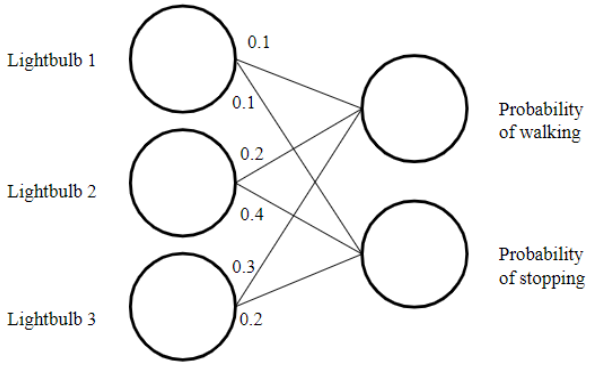

Figure 5: Neural network with an input layer linked to the output layer. This network shows the use of connections and connection weights. The network was initialized with random weights for the example.

A network in its most basic state has an input layer, for data to be passed into, and an output layer. In order for information to be passed between the two layers, each input data point is connected to each output point, with each connection initialized with a random connection weight. Once data is passed in, the connection weights are used to perform computation. This process will be explained further shortly.

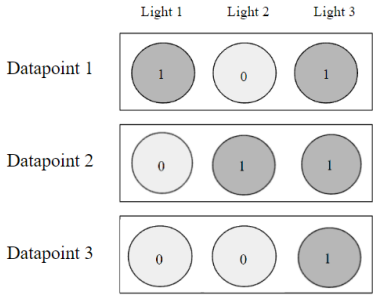

Figure 6: Serializing the data collected. The figure shows the representation of lights turned on as one, and the light turned off as zero.

Due to the connection weights of a neural network being numerical, the data passed into the input layer also needs to be numerical to perform computation. Figure 6 shows the re-representation of data.

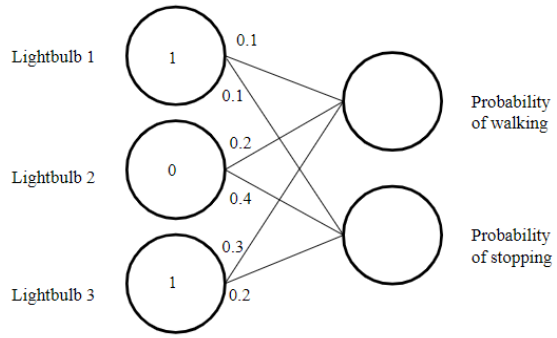

Figure 7: Inputting the data into the network. The first data point of Figure 6 is inputted into the network.

Once data is passed into the input layer of the network as shown in Figure 7, the output can be calculated using the connection weights. By multiplying the connection weight against the value of its origin node and summing the total of all such values flowing into the output node, the value is calculated.

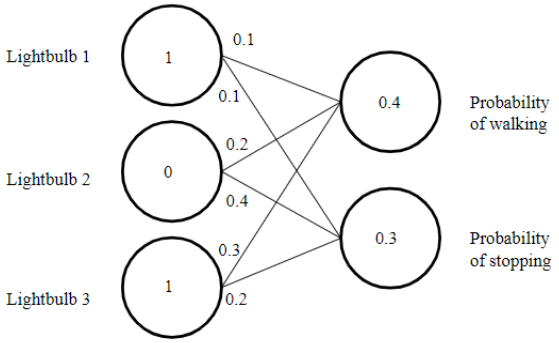

Figure 8: Given the input values of the network, output values are calculated. Connection weights are used to calculate the output.

Using the input data and connection weights, the probability of walking and stopping is computed (Figure 8). The output node with the largest value indicates the action the network believes was completed.

Unfortunately, the calculated values do not match the ground truth observed, so a backward pass of the network is used to increase accuracy.

$$(\text{Ground Truth} - \text{Calculated Output})^2$$

Figure 9: Mean Squared Error (MSE) formula. The MSE formula is used to calculate the network error. Ground truth is defined as the expected output of the data passed into our network from the dataset.

The first step when completing a backward pass is to calculate the error of each output node. This is done using a formula known as the Mean Squared Error shown in Figure 8. The calculated error using this formula can be seen in Figure 10.

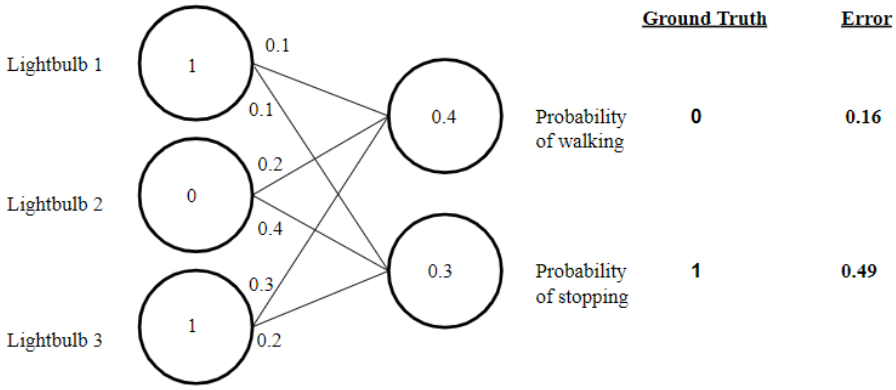

Figure 10: Calculating the error for the nodes in the network. The error is calculated using the Mean Squared Error formula.

The calculated error can then be summed up to indicate the total error of the network. The error computed in this manner however is not used to retrain the network. Instead, the difference between the output node value and ground truth is calculated, known as the delta. The calculated delta indicates how much higher or lower the node output should be to perfectly represent the ground truth.

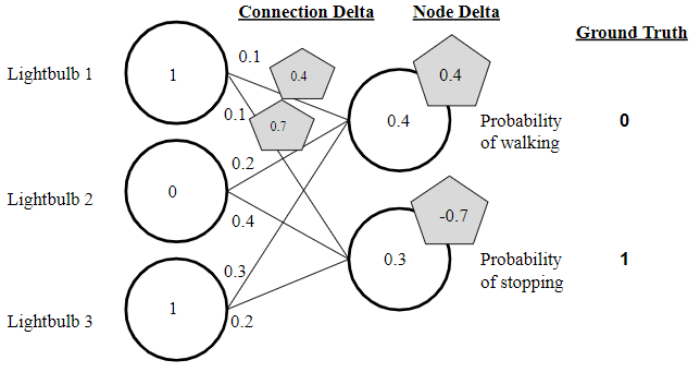

Figure 11: Calculating the node delta and connection delta. The connection delta is calculated by multiplying the node delta by the input node value passed into the connection leading to the output.

Once each output node delta is calculated, the value is multiplied by the input node's value whose connection led to the particular output node. The larger the input value, the more influential the node, as the total calculated with the connections will lead to a larger number, and thus the connection delta calculates the appropriate scaling factor. These computations are shown in Figure 11.

Once the connection deltas are calculated, the connection weight is updated by multiplying the weight against a constant number called the learning rate. A typical learning rate would be 0.01, with a larger number increasing the magnitude of the update to each connection. A larger learning rate can also lead to the network missing the sweet spot of connection weights that lead to the best solution.

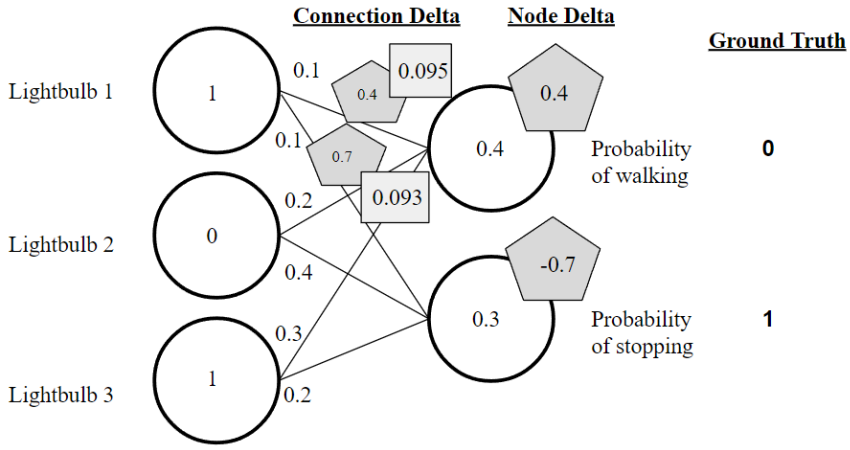

Figure 12: The update to connections is calculated. The number inside of the square icon represents the new value of the connection based on the computational steps discussed. The top number was computed through 0.1 (current weight value) - 0.01 (learning rate) * 0.4 (connection delta) equals 0.095.

The new value to a selected few connections is shown in Figure 11. Once all network connections are updated to their new values, the next set of data can be passed through the network. The steps discussed previously are then followed to pass the errors backward through the process known as backpropagation. By running forward and backward passes over new data points, the network is fine-tuned and starts to accurately predict the future of streetlight combinations.

A limitation of the current network topology is the relatively limited amount of connections. The more connections present, the more fine-tuning of the network can be done to find correlations in more complex data sets. A network topology with more connections is shown in Figure 12.

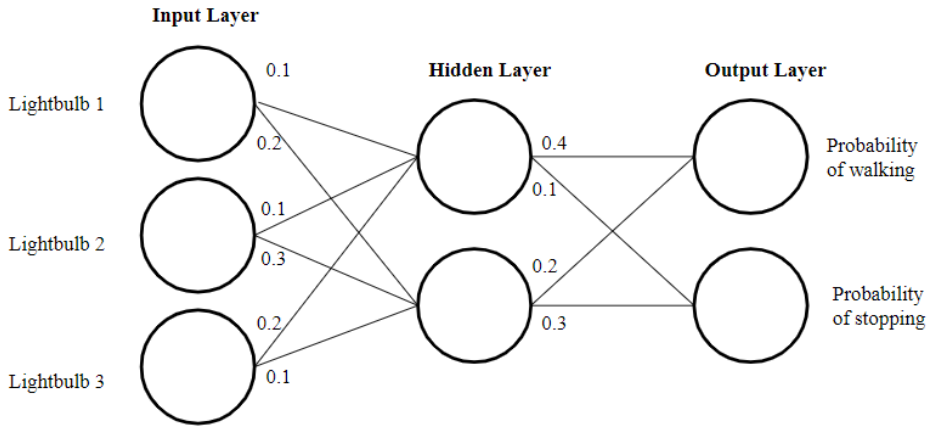

Figure 13: Three-layer neural network topology. Adding a hidden layer between the input and output layer adds more connections and fine-tuning capabilities to be able to find a stronger correlation. The connection weights are randomly initialized.

With more complexity added, the forward pass of the network does not change. The hidden layer nodes are filled with the calculated value from the previous layer as was done in a two-layer network, and then these values are used to calculate the output layer values. This can be seen in Figure 13.

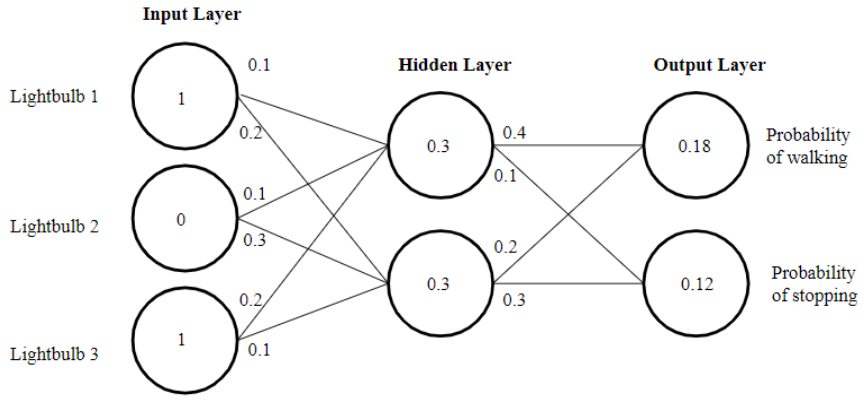

Figure 14: Forward pass of a network with three layers. The figure shows the sequential computational steps taken to reach the values in the output layer.

The values in the output layer, once again do not match the ground truth values, and thus backpropagation must be completed. Just as the delta was calculated in the two-layer network, the same is done for the output layer, however, delta values for the nodes in the hidden layer must also be computed to update connections from the input layer.

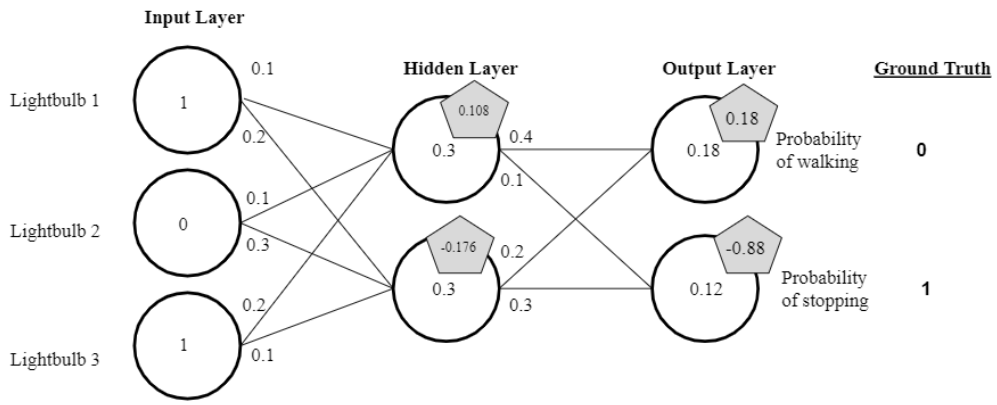

Figure 15: Calculating the delta for the output and hidden layer nodes. The figure shows passing the delta from the output layer backward through the connection weights and summing the totals.

The delta of the hidden layer can be thought of as simply passing through the delta of the output layer through the connections. By multiplying the delta of the node against each connection weight leading into it, and summing the total, we get node deltas for the hidden layer as shown in Figure 14.

A mathematical trick specific to three-layer or larger networks to help the network learn more correlation in the values of the dataset, instead of from which node the data is coming in, which may occur as a byproduct, we turn set the value of the node to zero when it would be below zero. This does not interfere with the computation in Figure 13, however, we need to multiply the node delta in the hidden by the node value if it is larger than zero, and if not we multiply the node delta by zero. This can be seen in Figure 15.

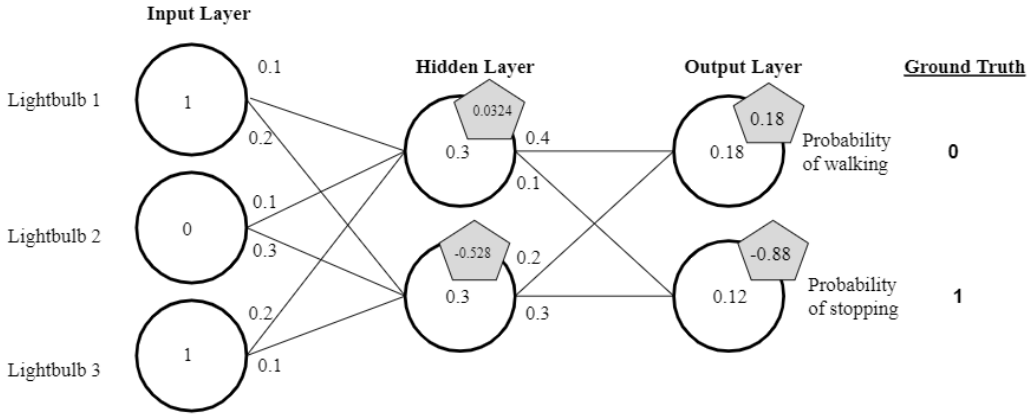

Figure 16: Hidden layer delta computed with the mathematical trick. The trick is in order for the network not to learn the correlation from where the data is coming from, and instead generalize its understanding.

Now that the delta is calculated the connections can be updated by multiplying the delta against the original node value and the learning rate. This value is then subtracted from the connection weight, producing a new more accurate weight.

By completing forward and backward passes iteratively we can increase the accuracy of our network in predicting what action follows what traffic light configuration. A high-accuracy neural network can also be used for locomotion - outputting the number of degrees each joint should rotate given its current position in an environment, or as a prediction tool - given a frame of a person walking, what is their next move? The reinforcement learning algorithms I used specialized in these two cases with some modifications.

2.3 Current approaches towards teaching AI locomotion

The idea of having a bipedal robot learn locomotion to increase its ability to generalize to new environments is not new in itself. My research aims to use a genetic algorithm to learn from motion-captured data to increase the ability to generalize to complex environments.

Research has already been done on using motion-captured data to prompt exploration. A paper published in 2021[8] focuses on the transition between observing the data and training in a virtual environment. The joints the robot had access to was kept as close to the real world as possible. Although this algorithm proved to be effective for the intended environment, it has shortcomings in dealing with environments outside the scope of the parameters of the motor controller.

Another influential study was published in 2022[9] by a team at Berkeley aiming to teach a quadruped robot from scratch how to tackle locomotion in new and unfamiliar environments. The work showed that it was possible to teach locomotion in only twenty minutes. However, a limitation of this study was its basis on having a carefully tuned controller which simplified the process drastically.

My research aims to fill the gap by using motion-captured data of people walking to aid in the learning process of a bipedal walker to reduce time performing actions that may not be feasible in the real world, as well as time exploring, and not use a fine-tuned controller to aid with generalization in any environment.

Methods

As was discussed in the Literature Review, the purpose of the research is to address the gap within the exploratory phase of artificial intelligence algorithms where agents explore sub-optimal ideas for a majority of the starting portion of the training. Without an example to follow, most algorithms complete random actions to observe their outcome and start building a correlation. This study employs the use of a novel two-part algorithm to increase efficiency in the training process by using mimicry as a basis of learning, to reduce the need for a fine-tuned controller as was done in previous literature.

My research aims to answer the question: How can agents be trained using mimicry to point the direction of exploration? I used a two-part algorithm pipeline to address the research question - mimicry and fine-tuning. The first part of the algorithm received a data example of a person walking and tried to predict the next steps. The learning is then continued by having the algorithm generalize its understanding on its own with a self-optimization algorithm.

3.1 Collecting data for training

The first step of my hybrid solution is to collect data on the human joint movement later used as the basis from which the AI learns to replicate the behavior.

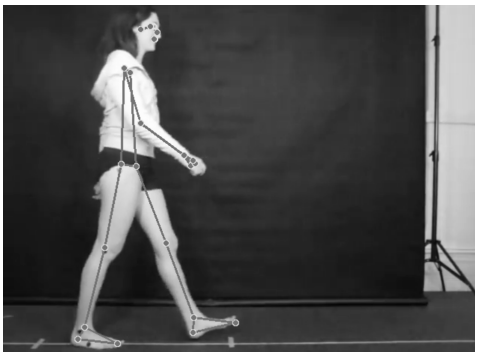

Figure 17: Paused frame of recorded video of walking with Mediapipe software applied. A paused frame is used from the captured video and computer vision is applied to mark all the joints and find the angles between them.

Using an open-source video of a person walking with the camera placed at hip level, multiple steps were captured. Using computer vision software developed by Google called Mediapipe, I was able to impose a mask over the footage with a dot at each major joint in the body (Figure 17). By connecting each dot with a line to the nearest next dot, it is possible to measure the angles of joints in reference to each other. To calculate the angle of the right waist I used the hip and knee as anchor points and the hip and ankle as anchor points for finding the angle of the knee.

With the ability to calculate the angles of joints in one frame of the video, I was able to calculate all the joint angles in each frame of the video. By stacking these calculated values subsequently in the same order as the frames, I was able to create a dataset. By splitting the dataset into pairs of current frame angles and subsequent angles, this became the format that could be passed into the network, with the subsequent angles being used as the target values for the output layer.

```
(91.02138162624868, 165.2067164398796)
(86.1023140225816, 164.33271783768532)
(115.05835592687896, 175.76960573167815)
(118.87746004430463, 176.0381626906006)
```

Left hip, left knee
Next frame: Left hip, left knee
Right hip, right knee
Next frame: right hip, right knee

Figure 18: Excerpt of the dataset of joint positioning. The figure shows the current position of each joint, as well as the angles in the next frame.

Figure 19: OpenAI virtual Bipedal Walker environment. An environment where motor control is inputted into the environment to cause the walker to move steps.

The virtual environment (Figure 19) used to simulate walking on a bipedal walker being used for this research, does not use target angles to move each joint, but instead, a joint is moved given how much power should be applied to the motor. The power applied is a decimal number in the range of negative one to one, with a negative number representing counterclockwise rotation, and positive numbers representing clockwise rotation. To represent the target angles as motor control, I propose to calculate the difference between the initial angle and subsequent angle and divide the result by 90, to squish the result between -1 and 1.

```
[ 91.02138163 165.20671644 115.05835593 175.76960573]   Left hip, left knee, right hip, right knee
[-0.05465631 -0.0097111   0.04243449  0.00298397]       Next frame joints in motor control
```

Figure 20: Current frame angles, with the next frame being represented in motor control. The figure shows the conversion of the next angles into motor control.

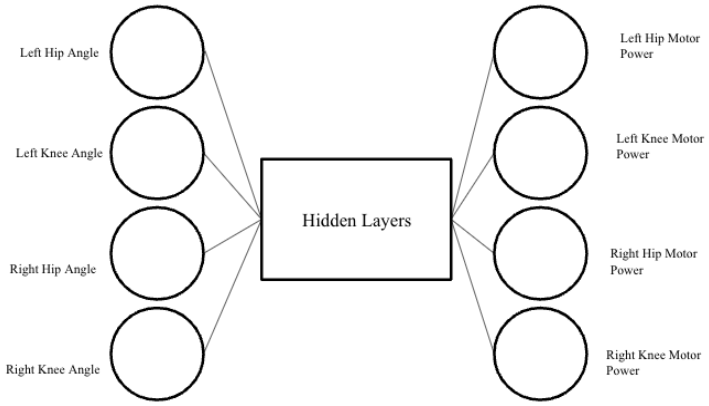

Figure 21: Network topology showing what inputs and outputs are expected. The figure shows what values are passed into the network and what each output node is expected to output.

Figure 21 shows the previously stated information visually represented. The input nodes receive information about the various angles of the person in the current frame, and the network outputs motor power corresponding to how the person chose to move each joint in the next frame. By training a neural

network to make this connection, the network learns to mimic human movement. The algorithms used to learn this correlation will be explained next.

3.2 Implementation of the algorithm

After creating the dataset, I designed a two-part algorithm that utilized two sub-algorithms working in unison. The first was responsible for learning the correlation between human joint movement and motor control, and the second generalized the learning to the virtual environment.

3.2.1 Implementation of NEAT with backpropagation

Choosing the best network topology for the task is important as it can lead to quicker and more efficient learning, which is a core concept of my research. A standard neural network topology with the architecture of an input layer, hidden layer, and output layer, was my first choice to teach the ability to mimic, due to this framework being a common industry standard. My hypothesis was that a simpler network, with only essential nodes, would outperform a more complex network in learning the dataset. This is because a simpler network would not waste time fine-tuning unnecessary connection weights, thereby enabling the AI to learn to walk faster and more accurately.

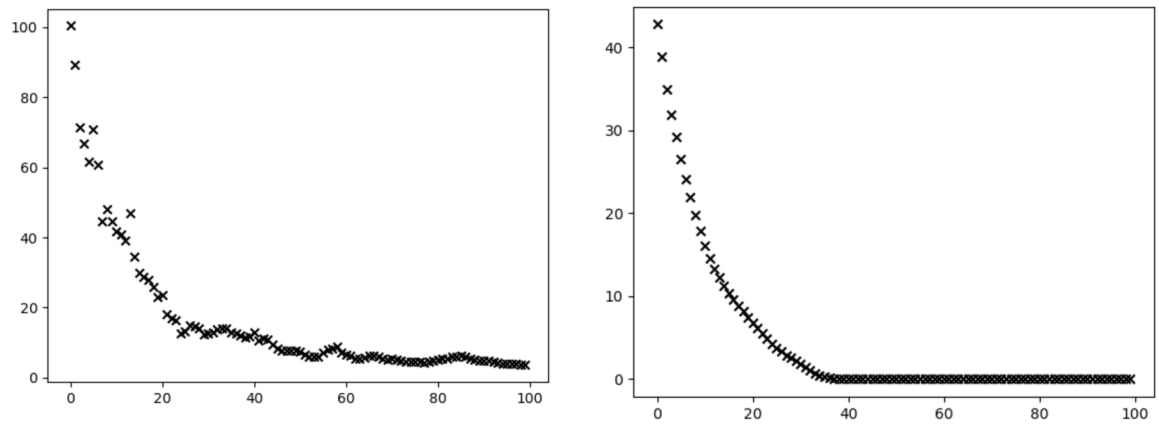

Figure 22: Two graphs showing decreasing error over two different network topologies. The X-axis represents the step taken, and the Y-axis represents the error at that step of learning. The left graph has a network with a fixed topology, and the graph on the right is a network with a minimal topology.

To test this hypothesis, I created two networks and trained them on the open-source MNIST database containing thousands of labeled handwritten digits in the form of 28 by 28 pixels. MNIST is a simplistic problem, so if the theory worked out here, it could work in a more complex environment, and MNIST would not be the bottleneck. The first network I created had an input layer with 784 nodes (28 x 28), 40 hidden layer nodes, and 10 output nodes (10 different digits). I then ran this network and plotted the error with respect to each training step. This graph is shown on the left in Figure 22.

Intrigued by the fact that the graph was not a perfect curve, showing that the learning was not ideal, I decided to generate a network instead. I created a network with an input layer size of 784 nodes (28 x 28) and an output layer of 10 nodes (10 different digits) and connected each input node to each output node. I

created five different types of mutation: adding a node between two connected nodes, connecting two nodes that are not connected, turning a connection on or off, resetting a connection weight, and adding a random number between -1 and 1 to the connection weight. By performing ten randomly chosen mutations to the network, I created my new network. The graph on the right of Figure 20 shows the error of the generated network. The figure shows that the network converges to a low error quicker and the learning is less sporadic compared to the graph on the left.

Since a simple topology led to better performance, I trained a generated network on the joint dataset. However, I implemented the NEAT algorithm to introduce a reward policy into learning. The premise is to use evolution to augment the topologies of neural networks until a solution is discovered that maximizes a reward.

Initially, a number of networks are created and trained on a fixed batch size of data. Before each backward pass, the network error is added to the network's score. Once each network has been run through, the network score is calculated by taking the inverse of the total error (1/total error). Networks are then split up into species based on how similar the networks are, and the bottom half of each species is eliminated and repopulated by breeding top performers of the species. Through completing these steps, a network that is best at mimicking human joints is produced, with a minimal topology.

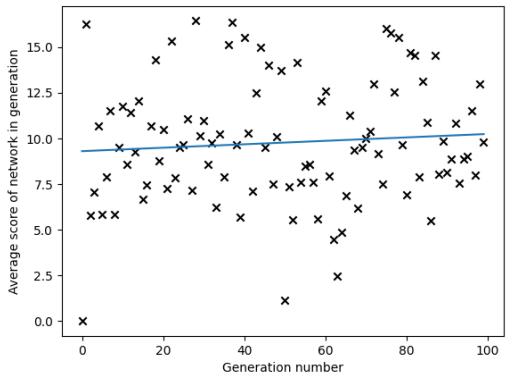

Figure 23: Average score of the network in each generation. The line of best fit shows that there is an upwards trend of score over generations.

Although Figure 23 shows scores to be sporadic, the trend is upward. Only a single best-performer network needs to be isolated as the best at mimicking human movement to then fine-tune the learning in the environment.

The next section will talk about the algorithm used to fine-tune the mimicry learned.

3.2.2 Implementation of DDPG on generated network

Once the network with the best score was selected from the mimicry step, I fine-tuned the network. This was done using the DDPG algorithm, which will be explained next.

Deep Deterministic Policy Gradient, is a standard simplistic algorithm used for environments that do not have binary actions but instead require continual control. In this algorithm, there are two neural networks

working to complement each other; the actor and the critic. Imagine them as two friends - the "critic" and the "probabilistic friend". The critic is good at judging actions as either good or bad, while the probabilistic friend calculates the probability of success for all possible actions in the current environment. Though they may make mistakes, they work together and learn from each other to determine which actions lead to the best rewards in a reinforcement learning environment.

I assigned the network specialized at mimicry to be the actor and used a fixed-size network as the critic due to the critic's output size being a single number, instead of a few nodes used to pick the action. By using these two algorithms in conjunction I saw the network learn to specialize its learning to the environment.

3.3 Data collection

The aim of this research is to reduce the time spent in the exploratory phase of the algorithm and instead direct the learning through using mimicry of a person walking. The hypothesis is that by having a network pre-trained on walking, it will be easier to generalize the learning to a virtual environment than starting from scratch.

The data throughout this experiment came in the form of recording the current learning iteration of the DDPG algorithm in combination with the average reward obtained through the environment. By mapping the time onto the x-axis and the reward onto the y-axis, the reward obtained by the algorithm can be analyzed.

The analysis of these graphs will be done in the next section, the results section.

4.0 Results

The purpose of the research is to address the gap within the exploratory phase of artificial intelligence algorithms where agents explore sub-optimal ideas for a majority of the starting portion of the training. The algorithm was run on a virtual environment used to simulate dropping a robot into an environment and seeing how far it can walk. The environment awards a reward at each step primarily used to teach reinforcement learning algorithms, but by taking the average reward over the entire episode - one full training iteration, we can get an idea of how the algorithm is doing.

To set the context of the effectiveness of the algorithm it is worth looking at a scenario of a network with no prior knowledge of locomotion being set into the virtual environment, with the same four network inputs (the four joints) as the fine-tuned network. The reward is the metric used to determine the effectiveness of algorithms, signifying the score achieved. In the virtual bipedal environment, the reward is roughly determined by adding a point for every step taken and subtracting 200 points when the robot falls over. If the robot reaches the end of the environment 200 will not be subtracted.

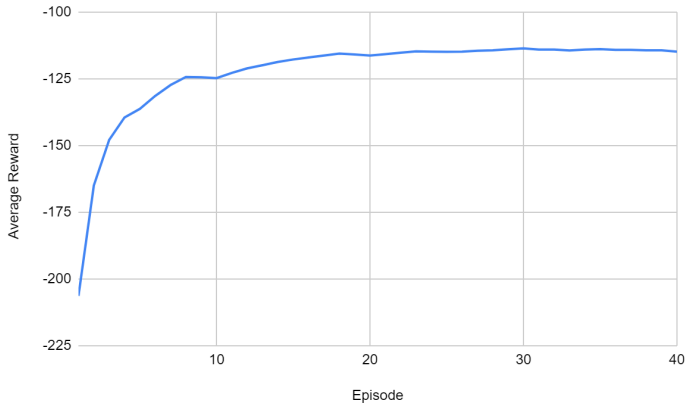

Figure 24: DDPG algorithm learning locomotion with the same network inputs. At each time step the virtual environment outputs a reward, for the reinforcement learning algorithm to learn from.

Figure 24 shows that the network started off with a low reward, learned, and then started to plateau. Having created a base context for how a blank slate network performs, I next fine-tuned my pre-trained network in the virtual environment.

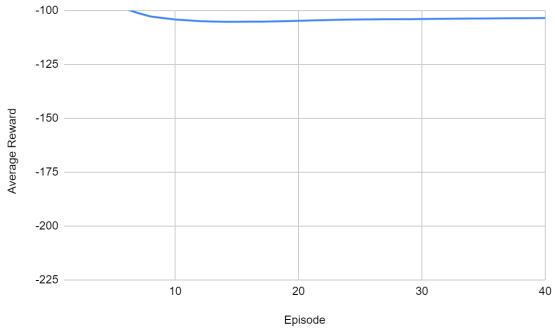

Figure 25: Results after fine-tuning mimicry network on the environment. The graph shows the average reward over each episode of the learning, showing an upwards trend and then a plateau. Figure 25 shows that the average reward plateaued after a number of training steps, following improvement. The difference between the fine-tuned network, and the blank slate network in Figure 24, is that the fine-tuned network started learning from a position with a higher average reward, and plateaued at a larger average reward; demonstrating the effectiveness of the human example.

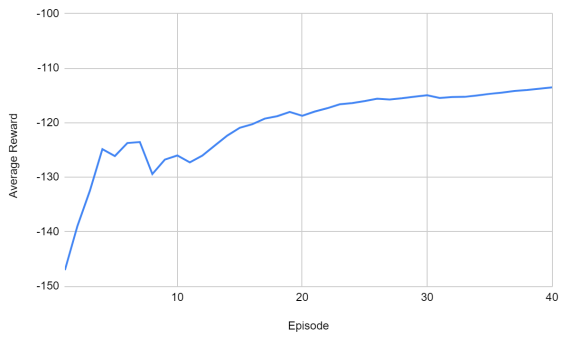

Figure 26: Results of a blank slate network trained with DDPG. The graph shows a continued positive trend in the average reward being achieved.

Intrigued by the plateau in both algorithms learning from a small number of environmental inputs, I decided to run DDPG from scratch, with the network having the input of all the environmental inputs (more than just the four joint angles). Figure 26 shows with more environmental input, the network continues to learn at a steady rate even though the network starts off at a worse place due to the lack of fine-tuning. The next section will discuss why this occurred.

5.0 Discussion

5.1 Regarding research objectives

The purpose of the research is to improve on the exploratory phase of artificial intelligence algorithms where agents explore sub-optimal ideas for a majority of the starting portion of the training. This study employs the use of a novel two-part algorithm to increase efficiency in the training process by using mimicry as a basis of learning, to reduce the need for a fine-tuned controller as was done in previous literature.

The network given time to pre-train on human locomotion outperformed the network starting from a blank slate, during the initial exploration phase, indicating that my research was successful. This can be seen when looking at the starting average reward and where it plateaus in Figure 25 compared to Figure 24. Figure 25 shows the use of a pre-trained network to learn locomotion starts at a higher average reward and plateaus at a larger reward than the network in Figure 24 which starts with a blank-slate network. However, limitations were present which will be discussed next.

5.2 Limitations and future directions

The network starting from a blank slate with more environmental input outperformed the network pre-trained on human locomotion over a longer period of time.

During the pre-training step, only joint positions were fed into the network, and motor control was the output. The same applied when I transferred the learning to the environment, I was only giving input of the biped walkers' four joints.

However, walking is more nuanced, and other input was possible to attain from the network such as the angle of the body, and the distance from the body to the ground. These additional inputs were passed into the comparison algorithm by simply using DDPG. By not having all the inputs present the hybrid solutions as limited by essentially traversing the environment blind, while the base DDPG algorithm could see everything.

During the stage of creating the dataset, these additional inputs were not present, and as such, the mimicry network was also trained blind, and the topology was fixed for only four inputs. Although more input nodes could have been added to the original network, but not have data passed into, I decided against this. By only switching on these nodes during the fine-tuning step, the original learning would be largely overwritten, to encompass the new data that was coming in, which would reduce the impact of the visual data. It is also unclear what nodes the new inputs nodes would be connected to, as they were not accounted for in the evolution of the network, and their optimal topology would not be learned.

A future direction of research is to explore the idea of using another algorithm that acts as a middle step between the NEAT network and fine-tuning step which is able to use additional inputs. In this case, however, the DDPG algorithm will no longer be able to be used, and a novel three-network algorithm will have to be created.

6.0 Conclusion

This research addressed the gap by providing an example of artificial intelligence to learn locomotion from and then apply the system to learn how to generalize the inherited understanding to new environments. By doing so, I hoped to create an AI system that learns locomotion more efficiently and reliably. The results showed that although the pre-trained network starts off at a larger average reward, it plateaus over time. The blank slate network does not encounter this issue and continues to learn, even though it starts at a lower average reward. During the pre-training step, only joint positions were fed into the network, and motor control was the output. The same applied when I transferred the learning to the environment, I was only giving input from the biped walkers' four joints. The number of inputs into the network proved to be a large bottleneck, as the robot was walking blind. A future direction of research is to explore the idea of using another algorithm that acts as a middle step between the NEAT network and fine-tuning step which is able to use additional inputs. In this case, however, the DDPG algorithm will no longer be able to be used, and a novel algorithm will have to be created.

References:

1. Lim, H., & Takanishi, A. (2007). Biped Walking Robots Created at Waseda University: WL and WABIAN Family. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, *365*(1850), 49–64. http://www.jstor.org/stable/25190427

2. S. Feng, E. Whitman, X. Xinjilefu and C. G. Atkeson, "Optimization based full body control for the atlas robot," 2014 IEEE-RAS International Conference on Humanoid Robots, Madrid, Spain, 2014, pp. 120-127, doi: 10.1109/HUMANOIDS.2014.7041347.

3. When do babies walk? Most babies take their first steps sometime between 9 and 15 months and are walking well by the time they're 15 to 18 months old. During your baby's first year, & Burch, K. (n.d.). *When babies walk, and how they learn to take those first steps*. BabyCenter. Retrieved February 28, 2023, from https://www.babycenter.com/baby/baby-development/baby-milestone-walking_6507

4. *What is Artificial Intelligence (AI) ?* IBM. (n.d.). Retrieved February 28, 2023, from https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

5. *What is machine learning?* IBM. (n.d.). Retrieved February 28, 2023, from https://www.ibm.com/cloud/learn/machine-learning

6. James, Steven & Konidaris, George & Rosman, Benjamin. (2017). An Analysis of Monte Carlo Tree Search.

7. *What is deep learning?* IBM. (n.d.). Retrieved February 28, 2023, from https://www.ibm.com/cloud/learn/deep-learning

8. M. Taylor et al., "Learning Bipedal Robot Locomotion from Human Movement," 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021, pp. 2797-2803, doi: 10.1109/ICRA48506.2021.9561591.

9. Smith, Laura & Kostrikov, Ilya & Levine, Sergey. (2022). A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. 10.48550/arXiv.2208.07860.