Perceptron Based Neural Network for Prometheus AI

Christopher Cullen O'Connor
260619267
Christopher.OConnor@mail.mcgill.ca

December 19, 2018

1 Introduction

The goal of this project was to design a drone artificial intelligence (AI) based on a neural network. The AI should be able to use data from the drone's sensors to make decisions that allow the robot to avoid hitting walls in tis environment. The drone should also be able to identify when it is in a new situation and alter its decision making process to adapt.

1.1 Prometheus

Prometheus is a large long term project with the goal of developing a swarm of drones that are able to interact with and navigate an unknown environment. The project could be used for search and rescue operations and to learn more about the nature of artificial intelligence. The project is broken up into several sections: the simulation team which is developing a 3d world simulation to test the drones in, the swarm team which works with developing the physical drones and making their sensors and intercommunication work, and the AI team which develops the decision making intelligence of the drone.

The AI team is itself broken into four sub-teams, each working on a piece of the Prometheus four part AI. The lowest level is the neural network layer, which this project is focused on. It has the task of interpreting sensor information from the drone and sending that interpretation onto the higher levels. It also has the ability to perform reflex actions, named such due to their similarity with biological reflex actions, such as avoiding dangerous objects or moving towards reward objects. The next layer is the knowledge node network (KNN) which is responsible for the memory of the AI, giving the drones the ability to make future decisions based on past experience. The third layer is the expert system. This layer makes use of the KNN to make a recommendation of what action to make. The final layer is the META reasoning layer. It takes recommendations from the expert system and judges if they are acceptable in the current reality. If the recommendation is chosen, the META layer tells the actuators to function and updates its current reality. If not chosen, the META layer starts a new thought cycle by prompting the KNN again.

1.2 Perceptron

The perceptron is a simple machine learning model used for supervised binary classification. In essence, it takes a labeled dataset and tries to compute a line or hyperplane that separates the dataset into two distinct categories. A perceptron accomplishes this by learning the values for a vector of internal weights, w. When a perceptron receives an input vector x, it takes the dot product of the input and weight vectors to determine the value of that input. Then this value is passed through a function that determines the class of the input based on this value. This function is classically a step function which assigns a positive class to positive values and a negative class to negative values. Alternative functions can be used when dealing with more than two classes or if it would make more sense to make non-linear class decisions. The main learning algorithm behind the perceptron is gradient descent. This enables the perceptron to learn the correct internal weights to make correct classifications in a test environment.

For more complex tasks, multiple layers of perceptrons are assembled to resemble the biological neuron structure. Therefore, these multilayer-perceptron networks are also known as artificial neural networks. Neural networks have a wide variety of uses in the field of machine learning and AI as they can be adapted to solve a wider space of problems than simple linear classifiers.

1.3 Gradient Descent

The algorithm that the perceptron uses to learn its internal weights is known as gradient descent, although gradient descent and similar algorithms are used in many different machine learning applications. Implemented in this project is stochastic gradient descent.

First, the dataset order is randomized. This is done to reduce error caused from a bias in the order of the dataset. Then, the weights of the model are iteratively updated using the following rule :

$$w_{t+1} = w_t - \alpha \nabla \mathcal{L}(w_t, w_{t+1})$$

Where w_t represents the weight vector at a given iteration, $\nabla \mathcal{L}()$ represents the gradient of the error function being used, and α represents the step size.

This step size is a hyper parameter that controls how fast the algorithm converges and how tight that convergence is to the real error value. The weights are iteratively updated until a predefined number of cycles, or epochs, have been done or until the error between two epochs is lower than some predefined threshold. One consideration in gradient descent is choosing the step size, epoch number, and threshold hyper-parameters, as these values can drastically affect the runtime and performance of the model.

Another consideration in using gradient descent is it is not guaranteed to arrive at the most optimal solution. If the error surface of the model is not convex then it is likely that gradient descent will only converge to a local minimum rather than a global minimum. While there are methods to avoid this problem they can either complicate the runtime of the algorithm or are complex to correctly tune.

2 Project Design

2.1 Simulated World

This project required a simulated world environment that a drone could interact with. The implementation of this world was based on work done by another member of the Prometheus team Abdullahi Elmi, with small changes done in this project.

The world that was implemented in this project was a very basic grid world – essentially a two dimensional plane that was divided up into discrete regions and given x and y coordinates. This grid world could be populated by different types of world objects that the drone would have to interact with. In this project three types of world objects were used; null objects that represented free cells in the grid, immovable objects that represented walls, and autonomous objects that represented a drone that existed in the world. For this project, only one autonomous drone object was present in a world at a time.

The world simulation has several functions that it can perform. It is able to add new world objects to itself at specific coordinates. It is able to process an action taken by a drone and update the world accordingly. It is also able to open a graphical interface which displays the current label and location of all objects currently in the world simulation as well as the tiles the drone can currently sense. An example of the interface at two moments in time can be seen in Fig. 2. These functions allow one to easily change and observe a world during testing of a drone.

2.2 Drone

The drone is the main actor in this project, as it is the vessel that is used to test the decision making process of the AI being developed in this project. The drone was built in a way that could be adapted to fit different world environments. In this project, the drone is a two-dimensional actor living in a two-dimensional grid world. Each drone is equipped with a name, an array of actions and an array of sensors, a TheWorld object which represents the drone's view of the world environment, an x, y, and direction value representing the drone's current position and orientation in the world, and a perceptron matrix that is used to make decisions. A class diagram of the drone can be seen in Fig. 1.

The action object has a name as well as three values that tell the drone how its x position, y position, and direction will change when it takes this action. Sensors are very similar. They have a name as well as x and y positions relative to the drone's position and direction. The sensors' main function is to examine the world object or objects in its range and calculate a value to pass to the drone's perceptron. In this project, each sensor is tuned to a specific type of world object; all sensors in this project look for immovable wall objects. If the sensor detects that object it will receive a score of -1, representing an unsafe cell. If the sensor detects an object it is not tuned for it will receive a score of 0, representing an indecisive cell. If the sensor detects a null object it will receive a score of 1, representing a safe cell.

After the drones are initialized, they can be given training data. Each data point is $s + a$ characters long, where s represents the number of sensors the drone has and a represents the number of actions. Given three sensors and five actions, the data would be formed as $\{s, s, s, a, a, a, a, a\}$, where the s values are the sensor values and the a values are the expected

action values the drone will take. These action values are either -1,0, or 1 and like the sensor values they represent unsafe, indecisive, or safe actions.

When a drone is being run in a world it first asks the sensors what grid cells are currently visible in the world. It passed this information to the world's display function so it can show the visible cells to the user. Then drone tries to make a decision. It does this by running its perceptron matrix with the sensor input data. If it finds a valid action, the drone attempts to take the action. If successful, the drone updates its position and orientation in the world and the process repeats. If the drone fails to make a decisive decision about what action to take then it begins the unsupervised learning process. If the drone takes a decision that leads to a collision with a wall object then the world will throw an exception to the drone telling it that it crashed, ending the current run. Ideally, this should never happen given correct training data.

2.3 Perceptron

The perceptron matrix is what drives the decision making process of the drone. When the drone is initialized it creates a matrix of perceptrons. The rows of this matrix are determined by the number of actions the drone is equipped with; each action is learned through its own perceptron. The columns of this matrix are determined by the number of sensors the drone is equipped with. Each perceptron in the matrix also has a bias weight. This bias can be thought of a shift up or down along the decision surface that is not affected by what the inputs to the perceptron are. A visualization of a perceptron matrix can be seen in Fig. 3.

When the drone makes a decision it passes its sensor information to the input of the perceptron matrix. This input vector is then multiplied by the weight matrix. This resulting vector is added with the bias vector to produce a vector of output values. These values are then passed through a logistic sigmoid function having the formula: $sig(x) = \frac{1}{1+e^{-x}}$. This has the effect of scaling the vector to have values between 0 and 1, with $sig(x = 0) = 0.5$, $sig(x > 0) > 0.5$, and $sig(x < 0) < 0.5$. Then, the maximum value from this output vector is chosen, with ties being broken randomly, and the corresponding action is returned for the drone to perform. In the case where no action is able to score above a value of 0.51 then the perceptron throws an indecisive exception to the drone. This tells the drone that it is not confident enough to make a proper decision based on the data it has been trained with. This value of 0.51 was chosen at is some small epsilon amount larger than 0.5 that seemed to work well in practice. This gives the overall decision function of the drone as:

$$decision = argmax(sig(x \cdot w_i + b_i))$$

With x being the input vector, w_i and b_i being the weights and bias of a particular perceptron i, and the $argmax()$ function selecting the action corresponding the the perceptron with the largest output.

For the perceptron to function it must be able to learn the correct weights. In this project, gradient descent was used to minimize the mean square error of the distance between a data point and the hyperplane decision boundary. This gives the equation for gradient descent of:

$$w_{t+1} = w_t - \alpha \sum_{i=0}^{n} (x_i w_t - \hat{y}_i) * x_i$$

Where w_t is the weights (and bias) value of the perceptron at a given iteration, x_i represents the input vector of a given data point, and $\hat{y}$ is the expected classification of that given data point. The step size chosen was 1E-4, the threshold value was 1E-20, and the epoch number was 1E+7. These values were chosen as the step size was low enough to closely approach the true minimum value for the error function without the algorithm taking much time to compute. Generally the correct weights are learned almost instantly.

2.4 Unsupervised Learning

It is expected that, in its final iteration, the drone will interact with a realistic world where there may be many different types of objects and constant changes to the drone's interpretation of the world. This could lead to situations where the drone is faced with information that it does not know how to use to make a decision. Therefore, the need for some unsupervised learning–the ability for the drone to generate its own training data on the fly–is needed.

The approach taken in this project was when the drone receives an indecisive exception from its perceptron matrix it creates a copy of its current view of the world and runs a training simulation. The purpose of this simulation is to generate new labeled data to further tune the drone without human intervention. This data is generated by simulating each action that could be performed and labeling the results of each action. If the action resulted in the drone moving to a cell that was not being sensed then it is given a score of 0, as the drone is not able to tell if the action is safe or not. If the action lead to the drone receiving a crash exception then it is given a score of -1, as that action was unsafe. If the action lead to the drone moving onto an empty cell then it is given a score of 1, as that action was safe. If the action lead to the drone simply rotating but not moving to a new cell then it is given a score of 0.5, as the action was safe but not productive. The newly produced training data is then fed back into the perceptron matrix to adjust the weights to tech the drone about the new environment.

3 Results and Conclusion

The main goal of the project, for the drone to be able to use its sensor data to avoid walls, was completely successful. When given good training data the drone's perceptron matrix was able to learn the exact decision boundary which separated the the actions of the drone perfectly. The drone was always able to take the correct action for a given sensor input, and it was able to randomly navigate through the basic environments it was placed in without crashing into walls.

The only part of this project that did not work as intended was the unsupervised learning method. It would seem that the approach to learning in this project was too naive. When the drone was initialized with a partially complete training set, the unsupervised learning method was able to fill in the gaps and be able to explore the world environment properly. For example, if given labels of what to do in every situation except when it is looking at three walls then it was able to learn that a rotation is the correct action in that scenario. However, when the drone was initiated with no training data, and expected to learn about the world

only through unsupervised learning, problems with this method arise. In this scenario the drone often learns that the best action in every scenario is just to take a rotation action as it will always be safe and give some positive score. Changing the reward weight for a rotation from 0.5 to some smaller value still results in this behavior as any positive reward is desirable. Changing the reward for these actions to 0 results in an infinite loop of the drone being indecisive as it is not able to make any progress if there are no safe actions it can take. Changing to a negative reward value gives similar results. It is clear that a more complex algorithm is needed for the drone to collect and weigh new unsupervised data to properly adapt to new environments and situations.

Overall, the project was mostly a success. The main goal of avoiding walls was met, but the additional goal of developing an unsupervised learning method still has much room for improvement.

4 Future Work

While the initial goals of this project have been met with some success, there is still more that can be done to improve it. The main goal for the future should be to improve upon the unsupervised learning method. One possible improvement to the current method would be to identify situations where the drone currently has any safe, productive actions. If this is the case it could give rotations a score of 0 as they are worse than already explored actions, but if there are no safe actions it could give rotations a positive score as they are the only option. In these cases it would be able to give rotation action a positive score only as a last resort to avoid the drone only learning to rotate.

Another goal for the future is testing the drone in more complex environments. This project only tested the drone in worlds that only contained walls, empty cells, and the drone itself. But one could imagine a world simulation filled with many different objects; other drones making their own decisions, goal objects that actively reward a drone for navigating to them, or dynamic objects that can be pushed when in contact with the drone. Any way to approximate the real world will give more insight into the nature of AI.

A possible future task could also be experimenting with a different AI model from the perceptron. The main benefits of the perceptron are that they are easy to train, can be updated easily in an online fashion, and overall not very computationally complex. However, they do have downsides. If the problem that a perceptron is trying to solve is not linearly separable it will produce poor results. One classic example is the perceptron cannot solve the XOR problem, when the inputs and expected output of the decision look like a logical exclusive-or operation then the perceptron cannot draw a linear boundary to properly separate the classes. Also, perceptrons are mainly meant for binary classification, when there are only two classes of data. However, since the goal of this project requires the drone to be able to identify when it does not have a good action, three classes are needed for data points. A 1 representing a safe action, -1 representing a negative action, and 0 representing an indecisive or unproductive action. While the current perceptron implementation is still able to separate three classes well, as future actions and sensors become more complex it may struggle to correctly make decisions.

One suggestion would be the decision tree. These classify inputs and make decisions in a

way that models programing if-then statements. At each branch of the tree a different input feature is analyzed. Depending on the value of that feature a different branch is followed to the next level of the tree. The nodes of the tree then represent the action that should be taken based on the path that was traversed in the tree. An example decision tree can be seen in Fig. 4. This model avoids the problems with a perceptron as it is not limited to making linear classifications. It is also very intuitive to follow the drone's line of reasoning by examining what features are looked at at each level of the tree. The downside to this model is tree structures can be slower to train than perceptrons, and are not as easily adapted to being trained in an online fashion. Still, a decision tree model could be worth pursuing in the future.

4.1 How to Use Codebase

The code base is written entirely in Java using only standard libraries. A repository can be found at https://github.com/CypherElmi/Prometheus-Neural-Network. To run a demo or a test, run the DroneMain.java file. To change from the default demo configuration you can alter the characteristics of the drone or of the world the drone will be placed in. First, a drone must be initialized with a sensor array, an action array, a TheWorld object, a perceptron matrix, x and y positions, and a direction feature. To initialize an action, call new Action(String name, int x, int y, double direction). Then group each action created into an array. To initialize a sensor, call new Basic_Sensor(WorldObject signal, int x, int y). Then group the created sensors into an array. To initialize a world for the drone, call buildSquareWorld(int size) which will create a (size x size) world surrounded by immovable WorldObjects. To populate the world use the helper methods in the file. The drawline function takes in a WorldObject and starting and ending coordinates and draws a line of that object in the world over the given coordinates. Additionally a plot function is given that is used for drawing single points. To initialize a drone call new Drone(String name, Action[] actions, Sensor[] sensors, TheWorld world, int x, int y, double direction) using the previously initiated objects as inputs. Then to train the drone call trainDrone(double[][] trainingData). Finally, when you are ready to test the drone in the world call run(int iterations, boolean visible) with iterations being the number of decisions the drone will make in the runtime, and the visible attribute determining whether to display the world's graphical interface. Further questions on how to initialize the various objects (actions, sensors, TheWorld ect.), or how functions of those objects work, should be answered in the comments included on those objects' respective java files.

5 Figures

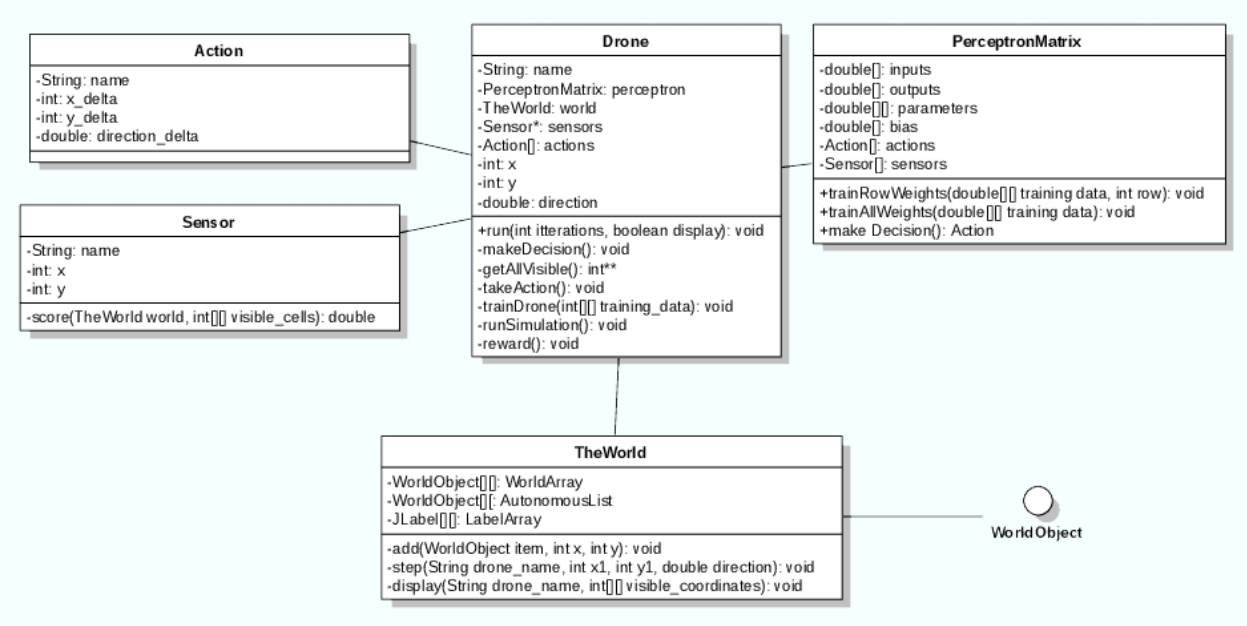
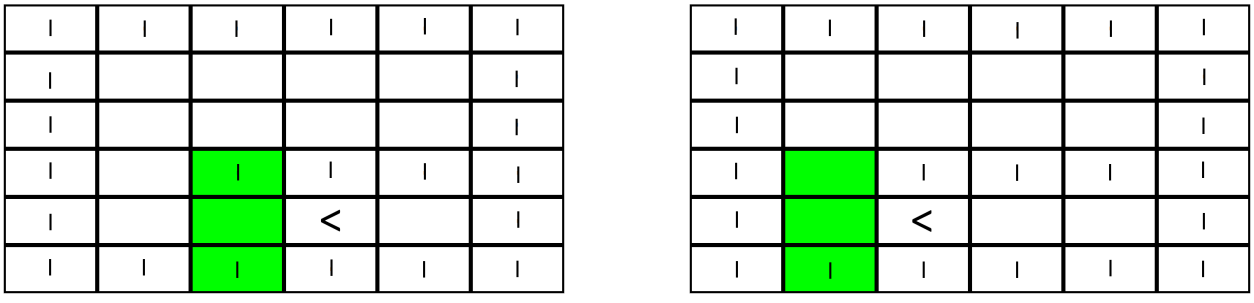


Figure 1: A class diagram of the drone class and related objects used in this project.



(a) Grid world interface for a run at time t=1

(b) Grid world interface for a run at time t=2

Figure 2: "<" represents a drone object (facing left), "I" represents an immovable object, unlabeled tiles are empty cells, and green tiles are being probed by the drone's sensors. (Sizes altered for visibility.)

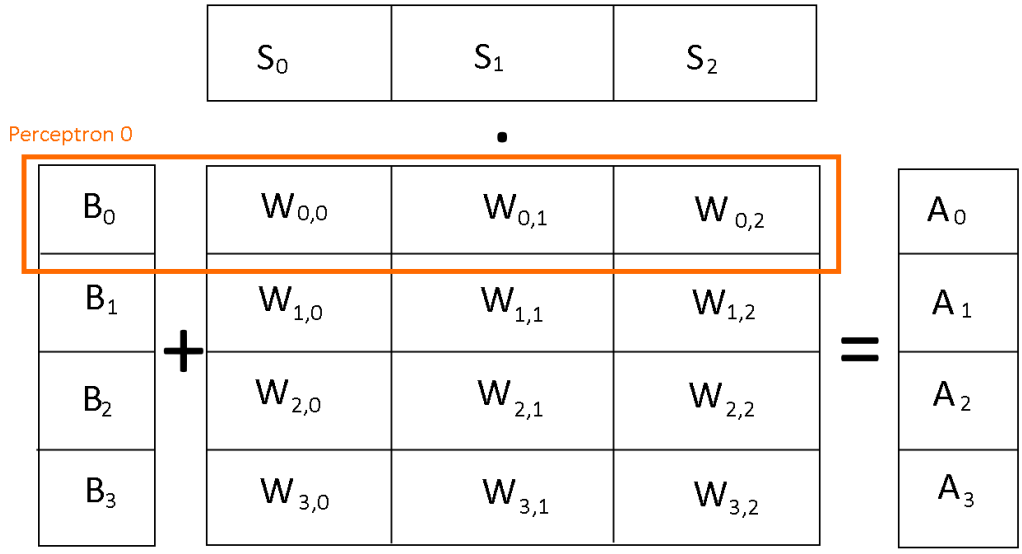


Figure 3: A visual representation for the perceptron matrix. S = sensor value, B = bias value, W = weight value, A = action value.

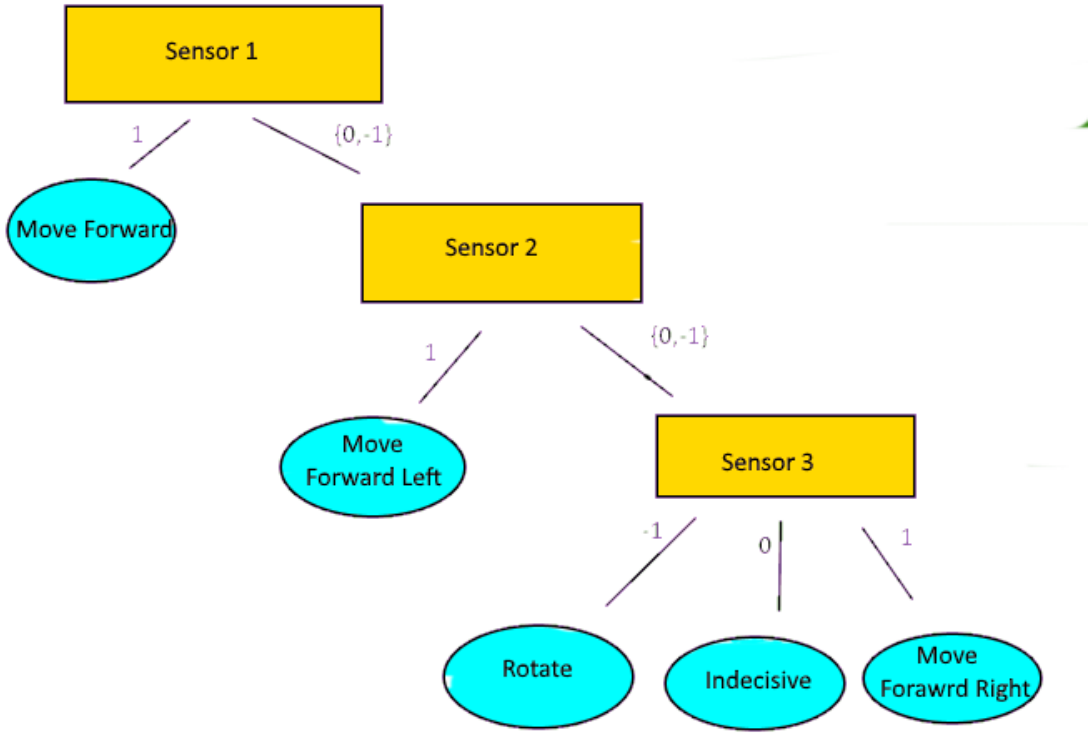


Figure 4: An example decision tree structure. Each yellow node represents a different input feature, in this case sensor values. The next level of the tree is reached by following the branch with the value of that feature. Blue leaf nodes represent actions that the drone will take based on the traversal through the tree.