Contents

1. 2018

1.1 August

Introductory post (2018-08-03 00:38)

Aim

This Wordpress site will be used to track progress of my research project on the shepherding problem. This post will provide an overview of the problem and the first steps taken in understanding it.

Conduct

The supervisor for my project is Professor Hussein Abbass, who is an expert in the field of machine learning and has research foci on autonomous systems, swarm behaviour and human-AI trust and interactions.

After meeting with Hussein to discuss a suitable project, we settled on the shepherding problem. This problem is about an AI being able to herd a number of flocking agents to a particular location and is based on the classic sheep-sheepdog interactions seen on sheep farms daily around the world.

To begin with, Hussein has provided me with an 11 week guide on how to attack the project, commencing with literature review of the classic problem. The literature is based on a paper by Strombom et al: ”Solving the shepherding problem: heuristics for herding autonomous, interacting agents”.

At the end of next week, I am to compare the literature to a set of (5) Matlab scripts that seek to reproduce the findings of the paper. This comparison is to be provided in a concise report back to Hussein.

Further research requirements will be discussed in future posts as they become relevant.

Results

I have now read through and understand the original paper in its entirety and am now looking through Hussein’s Matlab code to see how closely it matches the description in the paper. The paper itself provides no code so

that others can verify the findings but there is room to interpret the discussion so that a reproduction is very possible.

Current step: 8 hours

Running total: 8 hours

Strömbom and Matlab (2018-08-07 10:39)

Aim

Provide details on the report made to compare the paper by Strömbom et al. and Prof. Abbass's Matlab code implementation based on the discussion of the same paper.

Conduct

I have completed the report (3 pages) and submitted it to Hussein for review. I found a number of key similarities and differences between the two pieces of work and these are detailed in the report. The report is labelled Week3Report.pdf and can be found here for future reference: https://github.com/jakemyork/Shepherding-Problem.

Results

Both model and emulated code do not use machine learning but I am sure that this is just a step towards being able

to use machine learning to deal with the same problem.

Next Steps

My next task is to read the first chapter of Machine Learning, the 1997 textbook by Tom Mitchell.

Current step: 11 hours

Running total: 19 hours

Machine Learning: Chapter 1 **(2018-08-10 00:09)**

Aim

Provide a summary of the key points taken away from the first chapter of the textbook, ”Machine Learning” by Tom Mitchell, 1997.

Conduct

I have read through the chapter and the key points taken are summarised below.

Results

Although the textbook is now 21 years old, machine learning is based on principles and these are important to learn more. Therefore, despite all of the developments since 1997, the lessons still hold true.

A program is said to learn from experience with respect to a class of tasks and one or more performance measures if its performance in those tasks (as measured) improves with experience. For example:

- task: autonomous car driving on public road
- performance measure: average distance travelled before error
- experience: images, video and human commands recorded while observing a human driver

An important step in designing a machine learning tool to learn how to complete a task, is determining what kind of experience to provide for it to learn from. This experience can be of the following examples:

- a teacher can provide example situations and the best possible solution (or solutions);
- the learner can propose situations and ask the teacher for the correct solution;
- the learner can test different methods without guidance from a teacher to see if they lead to a correct solution;
    - this can then mean that the learner becomes more comfortable and reacts better to certain situations or it may create a range of novel situations to test against.

An outcome from reading the first chapter is that the book seeks to raise and then answer a number of important questions about the field:

- what algorithms exist to learn a task and in what settings will those algorithms achieve a successful solution, given a sufficient amount of training data;
- which algorithms perform better at certain types of tasks;
- how much training data is sufficient and how can we confidently characterise it as effective training data;
- how can prior experience by the learner assist it in finding a solution to the current problem;
- how do we determine the most effective next training scenario.

Chapter Four will cover artificial neural networks (the back-propagation algorithm specifically) and the general approach of gradient descent, including an example relating to facial recognition.

A well-defined learning problem requires a well-specified task, performance measure and source of training experience.

Next Steps

My next task is to read the fourth chapter of Machine Learning, the 1997 textbook by Tom Mitchell.

Current step: 6 hours

Running total: 25 hours

Artificial Neural Networks (2018-08-23 10:34)

Aim

Provide a summary of the key points taken away from the fourth chapter of the textbook, ”Machine Learning” by Tom Mitchell, 1997. The chapter covers the concepts of artificial neural networks and was supplemented by online teachings by 3Blue1Brown and the online textbook ”Neural Networks and Deep Learning” by Michael Nielsen, 2017.

Conduct

I have gone through all of the above material (only limited elements of Nielsen's textbook, despite it being very well written) and a summary is below. I will give Hussein a "lesson" on the subject on Monday, 27 Aug 18.

Results

An artificial neural network is explained by its name quite concisely. It is a network made up of artificial elements we call neurons. The network itself is divided into relevant layers, which make up one of three type of layer:

- input layer
- output layer
- intermediate decision or "hidden" layers (called hidden for the fact that their outputs are not typically observed).

A neuron as a variable that holds a number. In the classic multilayer perceptron model, a neuron is binary. In reality, each neuron that is not an input takes a function of inputs from the previous network layer and will output its binary answer depending on whether or not that function reaches some determined threshold ("bias"). In later models, there is a neuron called the sigmoid neuron which takes any value between 0 and 1, so that it is not necessarily switched fully on or off. Each neuron in each layer that is not the output layer has some weight that is allocated to it (more on this later). The neuron function is effectively the weighted sum of all of the neurons in the preceding layer minus a bias variable. The function output, the number between 0 and 1, can be described as the neuron "activation", which is a determination of how turned on that neuron is and how much it will affect following layers. No matter what the weight is, a neuron that is close to 0 will have very little effect on following layers (anything times 0 is 0).

Mathematical expressions for the above are written down but will not be reproduced in this blog - they will likely be present in the first seminar and final report.

The weights and biases of each neuron are initialised to random numbers as we don't know exactly what values given to neurons at each layer will do to the final answer that the network gives. As might be expected, the first answer will generally be pure garbage. However, this introduces the cost function. This is the sum of all of the

squared differences between the actual answers the output neurons should produce and those that the network does produce. When the difference sum is close to zero, then the cost is small. When it is not, then the cost is large comparatively. This is what the sum of squares gives us. In effect, the way we find the weight and bias changes for the cost function is very easy. We want to find the minimum cost of the function, which tells us that the network has given the most correct answer (that it can). To find the minimum of a single variable function, we take the derivative and just set it to zero. As we adjust the weights and biases, we eventually find a minimum of the function. This can be extrapolated over a large multi-variate function by taking the negative grad of the entire function. The only drawback with this is that it will give only a local minimum and will not necessarily find the global minimum. There are ways to mitigate this such as:

- adding a momentum element to move the output past small or minor minima
- creating multiple networks copies that all start learning from a different point and taking the highest performance network to be the ”averaged global minimum”.

At this point, I now have the mathematical tools and understanding of the structure of a neural network. I am not fully confident that I know how to implement such a structure in code to do... anything... yet, but I am confident that I would be able to analyse how some relevant code operates and compare it to my knowledge of the structure.

Next Steps

My next task is to ”teach” this subject to Hussein on Monday, 27 Aug 18, and then commence data generation to teach a neural network to learn how to shepherd the sheep in the original problem.

Current step: 12 hours

Running total: 37 hours

1.2 September

Neural Network Design (2018-09-05 05:09)

Aim

Provide an update on the latest progress in designing an artficial neural network.

Conduct

It has been some time since my last post, but Hussein and I have been discussing the different ways a network could be designed and how each layer can be structured. In line with this, I have done a fair amount of work on my presentation for Seminar 1 (10 Sep), which will be a brief lesson on neural network architecture.

Results

As per my last post, the mathematics behind neural network computation will be presented in Seminar 1. The network that I have designed was originally as per the following:

Pre-processing:

Direction from shepherd to goal position.

Direction from shepherd to global centre of mass (GCM) of sheep flock.

Direction from GCM to goal position.

Distance from GCM to goal position.

Determination of if the flock is aggregated enough.

If not, direction to furthest sheep from GCM.

Layer 0: choice of behaviour.

Inputs: determination of if the flock is aggregated enough.

Output: choice of driving or collecting behaviour.

Cost function: whether the shepherd made the correct choice.

Layer 1a: driving behaviour.

Inputs: direction to goal, direction to GCM, distance and direction from GCM to goal.

Output: velocity vector of shepherd.

Cost function: whether the shepherd reduced the angle difference between all inputs and reduced GCM distance to the goal.

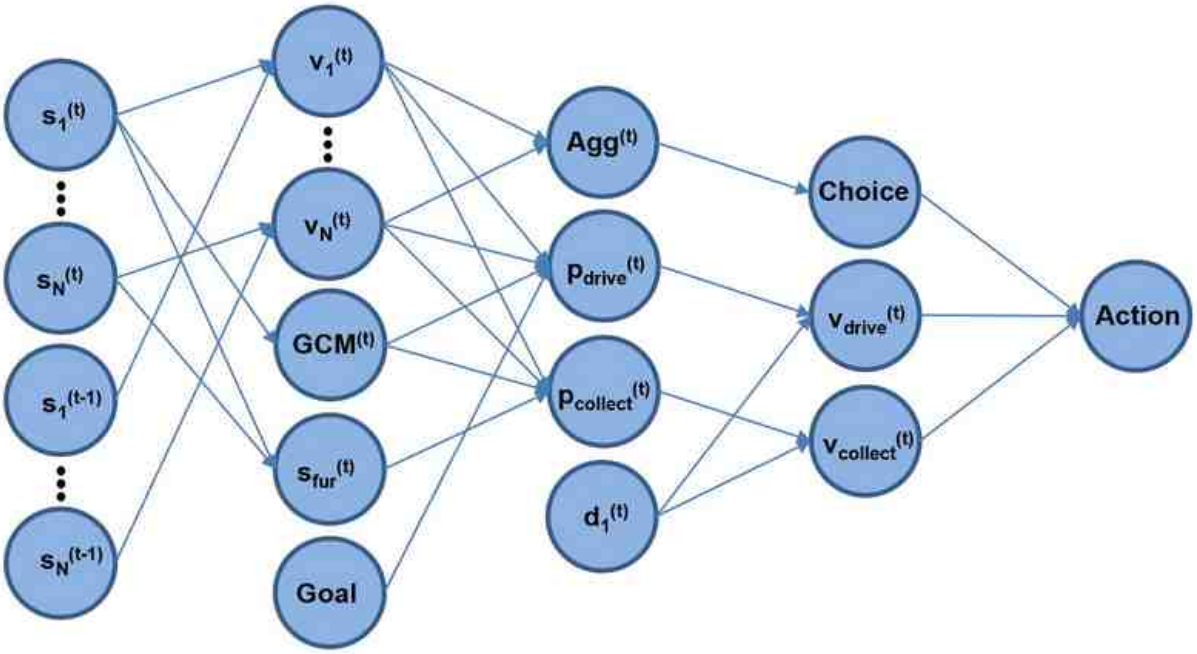
Layer 1b: collecting behaviour.

Inputs: direction to furthest sheep, direction to GCM, determination of aggregation.

Output: velocity vector of shepherd.

Cost function: whether the shepherd reduced the angle difference between the two inputs and caused aggregation.

Hussein thought this needed some expanding over several layers, so I have done that, presented in the following image:



Here, the following conventions apply:

- s = sheep location
- v = velocity vector
- p = position
- d = dog location

The other notation is fairly explanatory. It is my feeling that the network designed will be a sigmoid neuron network. For my presentation, this and multilayer perceptron networks will be discussed.

Next Steps

My next task will be to implement this network in code and then generate some data for the network to be able to learn how to do its job. It will also be to teach the other CDF students about neural networks and my project.

Current step: 13 hours

Running total: 50 hours

Neural Update (2018-09-10 00:28)

Aim

Provide an update on the latest progress in neural network design and presentation completion.

Conduct

I have constructed a set of code (below) to determine the shepherd collecting position programmatically. Using data from this, I will feed a neural network so that it can learn the collecting position from limited input.

Results

The code to compute the GCM and collecting position is pasted below:

```
height = 150; % length of arena
width = 150; % width of arena
outerMod = 0.9; % to scale the limits on where sheep can spawn
innerMod = 0.6;
x = 1; % easy for row indexing
y = 2;
numSheep = 20; % the number of sheep being simulated
sheepRadius = 2; % this value is how far the sheep need to be from other sheep

% calculate random values for the (x,y) coordinates for each sheep
for s = 1:numSheep
sheepMatrix(x,s) = (outerMod*width - innerMod*width)*rand(1)+ innerMod*width;
sheepMatrix(y,s) = (outerMod*height - innerMod*height)*rand(1)+ innerMod*height;
end
% calculate the GCM of the flock
GCM(x) = mean(sheepMatrix(x,:)); GCM(y) = mean(sheepMatrix(y,:));
% calculate the direction vectors from each sheep to the GCM
for s = 1:numSheep
directions(x,s) = sheepMatrix(x,s) - GCM(x);
directions(y,s) = sheepMatrix(y,s) - GCM(y);
end
% calculate the magnitude of each direction vector
furthestNorm = 0; % store the largest distance from GCM
furthestIndex = 0; % store the index of furthestNorm
```

```
for s = 1:numSheep
temp = [directions(x,s) directions(y,s)];
norms(s) = norm(temp);
if (norms(s) > furthestNorm)
furthestNorm = norms(s);
furthestIndex = s;
end
end
% calculate position of collecting position based on furthestIndex and GCM
% location
scalingFactor = (furthestNorm + sheepRadius)/furthestNorm; % the shepherd collecting position is only 2m away from the furthest sheep
collectingPos(x) = (directions(x,furthestIndex)*scalingFact or+GCM(x));
collectingPos(y) = (directions(y,furthestIndex)*scalingFact or+GCM(y));
% specify an extra index to hold the origin so that this is also fed to the
% neural network

circleSize = 10;
scatter(sheepMatrix(x,:),sheepMatrix(y,:),circleSize,'filled'); hold on; scatter(GCM(x),GCM(y),circleSize,'filled');
hold on; scatter(collectingPos(x),collectingPos(y),circleSize,'filled'); xlim([0 width]); ylim([0 height]);
```

This code creates N sheep in random locations in the upper right quadrant of the arena. It then calculates the GCM and the shepherd collecting positions separately and plots all points of interest. Right now, I am working on being able to replicate this on a large scale to provide training data to the network. The inputs will be the sheep locations and the target output will be the collecting position. The issue right now is an error with input and output array sizes being different but Hussein has suggested a fix for that. I will try to implement that before the next post.

Hussein has discussed with me that my previous post's architecture actually represents a fusion network - a network of neural networks - in which all tasks can be individual neural network tasks that can be trained separately. This is not an issue but we are taking first steps in trying to train a single outcome first before moving on to other tasks.

Speaking of other tasks, I have completed and fully rehearsed my presentation for this evening. It should be fairly informative and educational for those with limited experience with neural networks.

Next Steps

My next task is to generate the data and confirm that using Matlab's Neural Network Toolbox, the network can learn where collecting positions are best situated.

Current step: 7 hours

Running total: 57 hours

Network Training Completion **(2018-09-11 07:30)**

Aim

Provide an update and some insights into the completion of four neural networks as part of solving the shepherding problem.

Conduct

I have amended the previous post's code to store 20,000 sets of training, validation and testing data for the teaching of four neural networks to be able to determine the GCM, driving position, collecting position and whether or not the flock is clustered enough.

Results

As per the previous code example, the sheep spawn locations were set between the bounds of (0.6,0.6) and (0.9,0.9) (times the length and height of the arena respectively). It turns out that this condition for training the network is

actually very restrictive. As soon as the network sees any other level of clustering and a flock of sheep that doesn't sit inside those particular bounds inside the arena, its accuracy is drastically reduced. This was found after some simple and quick testing.

Therefore, I retrained the networks on much more varied and randomised data, by altering the limits of the sheep spawn locations. The networks are therefore trained on all manner of clustering and flocks that are located all around the arena. In doing this, the networks have been able to perform much better when looking at the distribution of sheep and when provided the right inputs, they have learned with high degrees of accuracy what their outputs should be.

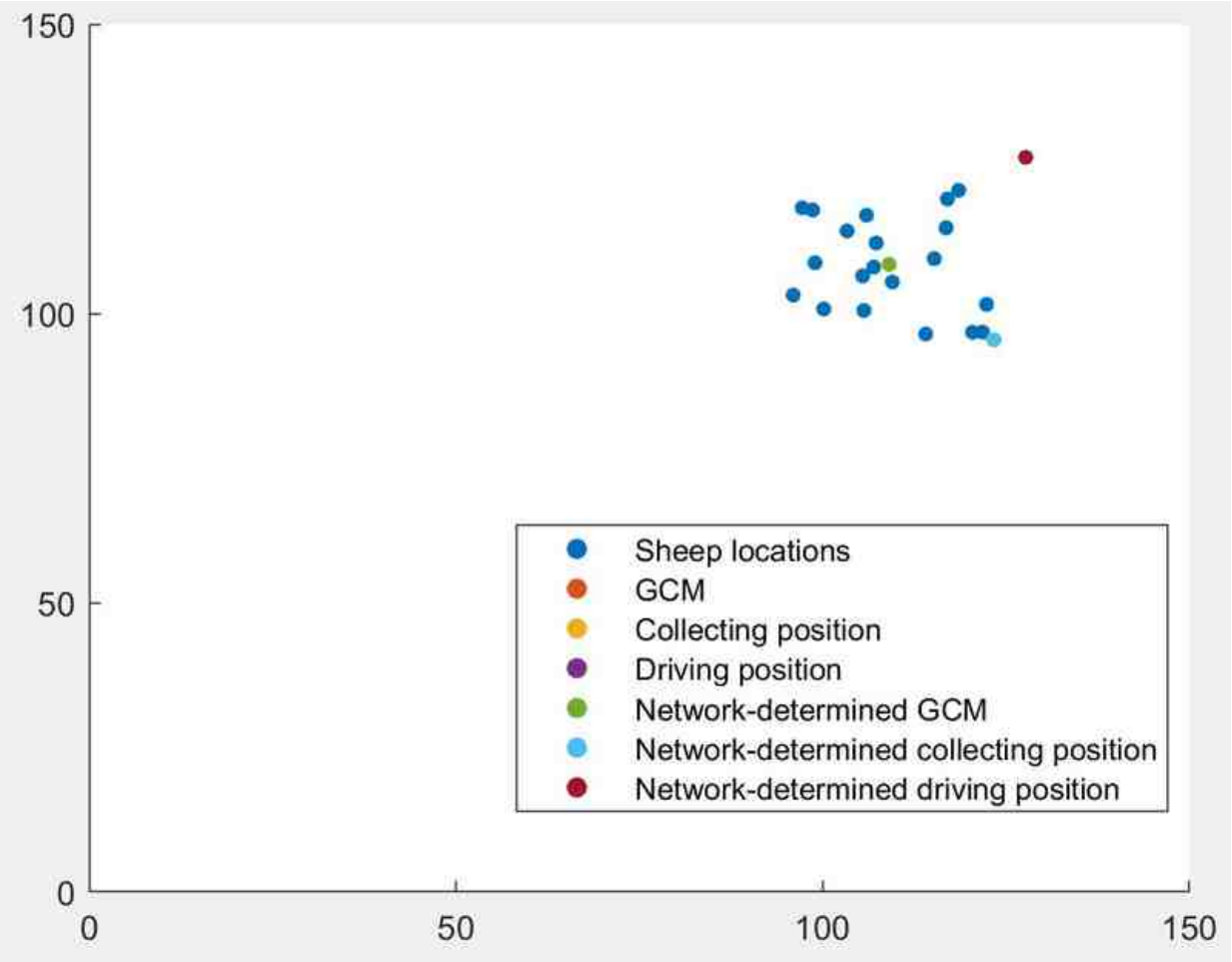
The GCM network takes all sheep locations as a 1-row 40-column vector input and outputs a 1-row 2-column GCM location.

The collecting position network takes the GCM and the furthest sheep coordinates as a 1-row 4-column vector input and outputs a 1-row 2-column collecting position location.

The driving position network takes the GCM and a "back of flock" position coordinates as a 1-row 4-column vector input and outputs a 1-row 2-column driving position location.

The isClustered boolean network takes the same inputs as the collecting position network, but instead outputs a single logical output.

The proof of this working is shown in the below figure. In fact, it is not possible to see any of the actual points of interest, only those generated by the networks. This is due to the high level of accuracy in the network outputs. Upon zooming in to a very high level, differences between the "correct" outputs and the network outputs can be observed.



Next Steps

Next, I will seek feedback from Hussein about this progress. I will then either continue to train networks on this problem and start looking at integrating the shepherd and vector movements, or potentially I will start looking at a reinforcement learning model to compare.

Current step: 8 hours

Running total: 65 hours

Full Simulation Testing **(2018-09-16 10:21)**

Aim

Provide an update on the testing of the neural networks against a scripted output from start to finish, as well as statistics accumulation of mean and standard deviation differences.

Conduct

Over the weekend, I implemented 10,000 test cases of the shepherding problem simulation, calculating means and standard deviations of each network against its relevant target output. This was done over a full range of possible starting positions, with 20 sheep in a 150x150 arena. Following that, I wrote code to run simulation from start to finish using both the target outputs and the neural network outputs, and having the shepherd and sheep behave as per the original problem statement. Comparisons of these simulations were then done over 1,000 test cases.

Results

In the mean and standard deviation testing, it was found that differences between target and neural outputs were extremely low. The collecting position network performed the worst of all by far, with an absolute mean difference being approximately 0.2, while the other networks showed mean differences between 0.0001 and 0.01 over 10,000 random test cases. Having said that, although the absolute difference may seem significant, when shown as a scale

of the entire arena width or height (150 m), then this becomes statistically insignificant. Therefore, on independent random test cases, the neural networks perform very closely to their target outputs.

The code to run a full simulation using target outputs or neural outputs went smoothly. I will show this to Hussein in our next meeting, but the shepherd herds the flock of sheep successfully at an average of 310 time steps. More testing is required in this space as, after some analysis, it was found that a number of test cases (for both target output and neural network) failed to herd the flock at all as the shepherd became pinned on the other side of the flock from the driving position. This meant that the shepherd would halt for this time step and if nothing changed significantly, would halt for the next time step and so on. I have considered implementing a random perpendicular vector output if the shepherd halts for more than one consecutive time step to prevent this from occurring. As stated, I will discuss this with Hussein.

In the testing, the cases were limited to a maximum of 1000 time steps before determining a simulation failure. When any failures were discounted, the averages were comparable at 310 time steps, but in counting the failures, over 3 successive tests, the neural networks failed 6 % of the time, compared to 4 % of the time for the scripted outputs. I will discuss this with Hussein as well and look at running more testing to confirm the failure bias.

Next Steps

Next, I will meet with Hussein to discuss my latest achievements and what my next goal should be.

Current step: 14 hours

Running total: 79 hours

1.3 October

Code Revision (2018-10-02 23:15)

Aim

Describe changes made to the various scripts after discussion with Hussein on where to from here.

Conduct

Hussein asked me to give a presentation to him and his 4th year thesis students on my project last week. Resulting from this presentation was a discussion with Hussein, detailing where we should go from here. The goal is now to see how the neural networks perform under non-ideal conditions by introducing noise into the inputs, reducing visibility of the shepherd and changing other variables. In order to conduct this testing, I have spent some time cleaning up the code of my various scripts to ensure they are readable and fit for the final report (as appendices). I am no longer going to compare the neural networks to a reinforcement learning model.

Results

With the exception of the four network scripts and the three testing scripts I have, there are three non-testing scripts that have been revised to provide ample information about the computation contained within.

As a sidenote, after seeing the presentations of the 4th year thesis students, it appears that I have made comparable progress with the notable exception of having completely reconstructed the problem from scratch after seeing the issues with Daniel Baxter's (4th year thesis student last year) code. All of the thesis students have used his code and although our goals and methods differ by a lot, their results are far less conclusive.

I'm confident that testing the neural networks will yield some really interesting results as they've been specifically trained to an incredibly high resolution with 20 sheep. Changing this number or doing a calculation on a subset related to the range of the shepherd's vision could be really interesting.

Next Steps

Start testing the networks for robustness under non-ideal conditions.

Current step: 8 hours

Running total: 87 hours

Introducing Noise **(2018-10-06 00:57)**

Aim

Provide details on code update to introduce noise into simulations.

Conduct

I have implemented a script that introduces noise in terms of the positions of the sheep that is scaled by how far the shepherd is from the GCM of the flock. If the flock are far away, then the inputs to the neural networks are more corrupted. If within a certain range (40 m), then the shepherd has full clarity and all noise is removed from the inputs.

Results

Effectively, for the inputs to the neural networks, all sheep positions have been recalculated with an added random x and y coordinate. The sheep positions themselves do not change, only the inputs to the networks. This way, new sheep positions every time step are uncorrupted. The random x and y coordinate addition is scaled by a factor that is influenced by the distance from the shepherd to the GCM. The logic here is that the further away the shepherd is, the less accurate the network determinations will be. As the shepherd approached the flock, the scaling factor reduces (linearly with shepherd distance) until the shepherd is 40 m away. When the shepherd gets inside this range, the scaling factor reduces to zero and inputs to the networks are no longer corrupted.

I found that placing a limit on the noise introduction distance (40 m) reduces a lot of computation time for the simulations, but does not change much in the outcomes for each simulation. By the time the shepherd has gotten to 40 m from the GCM of the flock, the noise has reduced to a point that the position calculations for the shepherd's next move are only varied minimally. This is true even when the scaling factor is large.

As I write this, I am running 20,000 simulations, with a linearly incremented scaling factor every 1,000 sims. This way, I can measure to what degree noise influences the shepherd. Results will be included in the final seminar and report.

Next Steps

Analyse results from noise testing. Introduce partial visibility testing.

Current step: 7 hours

Running total: 94 hours

Noise Testing **(2018-10-08 03:30)**

Aim

Provide details on simulation noise testing.

Conduct

Having implemented position-based noise into individual simulations, I have now run testing over a broad range to assess the impact the noise has on the performance of the shepherd.

I have also started writing the final report and am at approximately 3 pages.

Results

I have run 100,000 simulations, iterating over 25 different noise levels. Unfortunately, the noise appears to have a linear effect on the shepherd performance in simulation; that is to say that increasing the noise directly increases the time taken for the shepherd to complete its task.

I was hoping that there might be diminishing returns or that noise above a certain level would yield a flat response for the shepherd but it seems not to be the case. As mentioned in the last post, once the shepherd gets within a certain range of the flock GCM, the noise is removed entirely. Therefore, even at the upper limits of noise applied, because the shepherd still approaches the general direction of the herd, eventually it is able to reduce the noise to zero and complete the task thereafter.

Although mildly interesting, this is not a significant result for testing the limits of the shepherd.

Next Steps

Introduce partial visibility testing. Also considering testing of arena size differences.

Current step: 6 hours

Running total: 100 hours

Changing Sheep Numbers **(2018-10-11 09:53)**

Aim

Provide update on success with changing the number of sheep in the simulations.

Conduct

I have implemented a way to change the number of sheep that are used as inputs to the neural networks. This is interesting because the networks have been so rigorously trained on 20 sheep that their performance with different numbers of inputs could be greatly varied.

Results

Testing has just been completed on 40,000 simulations, looking at increments of 5 sheep for 1,000 simulations. That is, the increments go from 5 sheep up to 200 sheep in steps of 5.

The neural networks have fixed limitations on how they can accept inputs. For example, the GCM determining network must have a single row vector of 40 inputs in order to run its algorithm. These inputs are the x and y coordinates of 20 sheep. In order to allow for other input sizes, I provided the networks with extrapolating sizes under 20 and averaging sizes above 20.

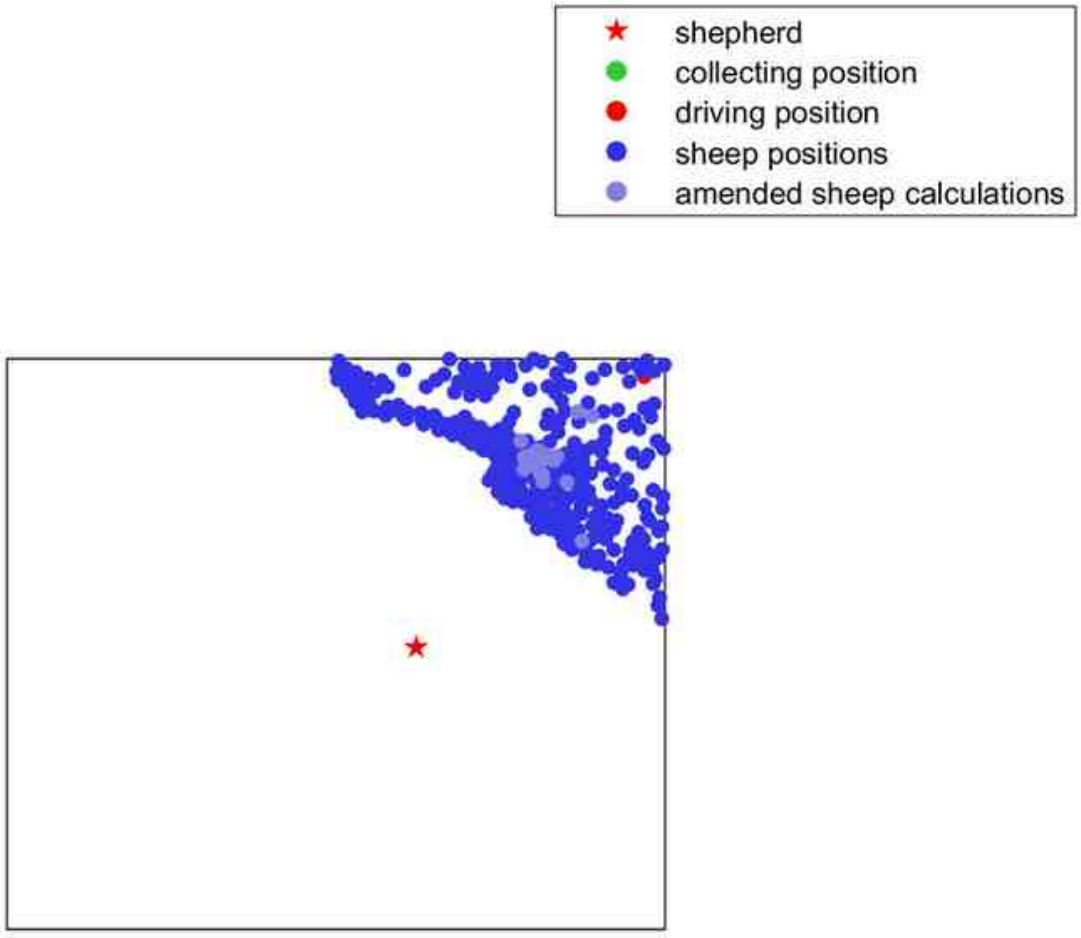
For numbers less than 20, effectively, all I have done is keep those numbers, take the average and add some small randomisation to fill the gaps. It has little overall effect on the outcome. For example, if there were 19 sheep, the 20th fictional sheep that is fed to the networks as input would be generated as the mean value of all sheep positions plus a small randomisation value for its x and y coordinates. The randomisation is small enough to have minimal effect on the overall distribution but large enough that the fictional input sheep are not overlaid on top of each other. To be clear, these are simple calculations that are fed to the networks, they are not newly generated sheep that cause repulsion from other sheep or exhibit any other behaviours.

For numbers over 20, I have effectively taken the average of a set of sheep indices to generate 20 new fictional positions that are fed to the network. This is fully scaleable (up to 400) and works like the following examples, which are more succinctly implemented in Matlab code:

If there are 38 sheep, then the average of the 1st and 21st is taken - this is input 1. Same for the 2nd and 22nd - input 2, and so on up until 18 and 38 - input 18. After that, input 19 and 20 are just the 19th and 20th original sheep.

If there are 58 sheep, then the average of the 1st, 21st and 41st is taken as input 1. And so on up until a possible maximum limit of 400 sheep. Technically, this could be unlimited but given the size of the arena, 400 is already too high for realistic purposes.

It's important to note that although the networks receive vector inputs from the 20 altered inputs to make decisions of where to place various points (collecting position etc), and the shepherd moves to these points, the shepherd does not directly interact with the 20 altered input sheep. It still interacts with the actual sheep in the arena and vice versa. If the shepherd is trying to go to a collecting position that is in the middle of two sub-groups of sheep, this can cause issues - this has been observed at high numbers of sheep. Conversely, because the shepherd still causes all of the sheep to move, at high numbers, there is a large cascading effect that is beneficial to the shepherd as per the below image:



As can be seen, the sheep at the edge of influence of the shepherd are pushing the adjacent sheep and so on. This means that at high numbers, the shepherd is more efficient at herding sheep due to their intrinsic behaviour. Although not an effect of the neural networks, I thought it was still an interesting phenomenon to note.

Next Steps

Testing of arena size differences. Analysis of numbers changing data that was just captured.

Current step: 6 hours

Running total: 106 hours

Noise Revisited **(2018-10-13 11:50)**

Aim

Provide update on change to noise dynamic in simulations.

Conduct

I wasn't happy that the noise introduced repeatedly changed position at every time step. Therefore, I have revised the noise input to be more appropriate to what I feel a real situation for a shepherding dog (in a foggy paddock, for example) would be like.

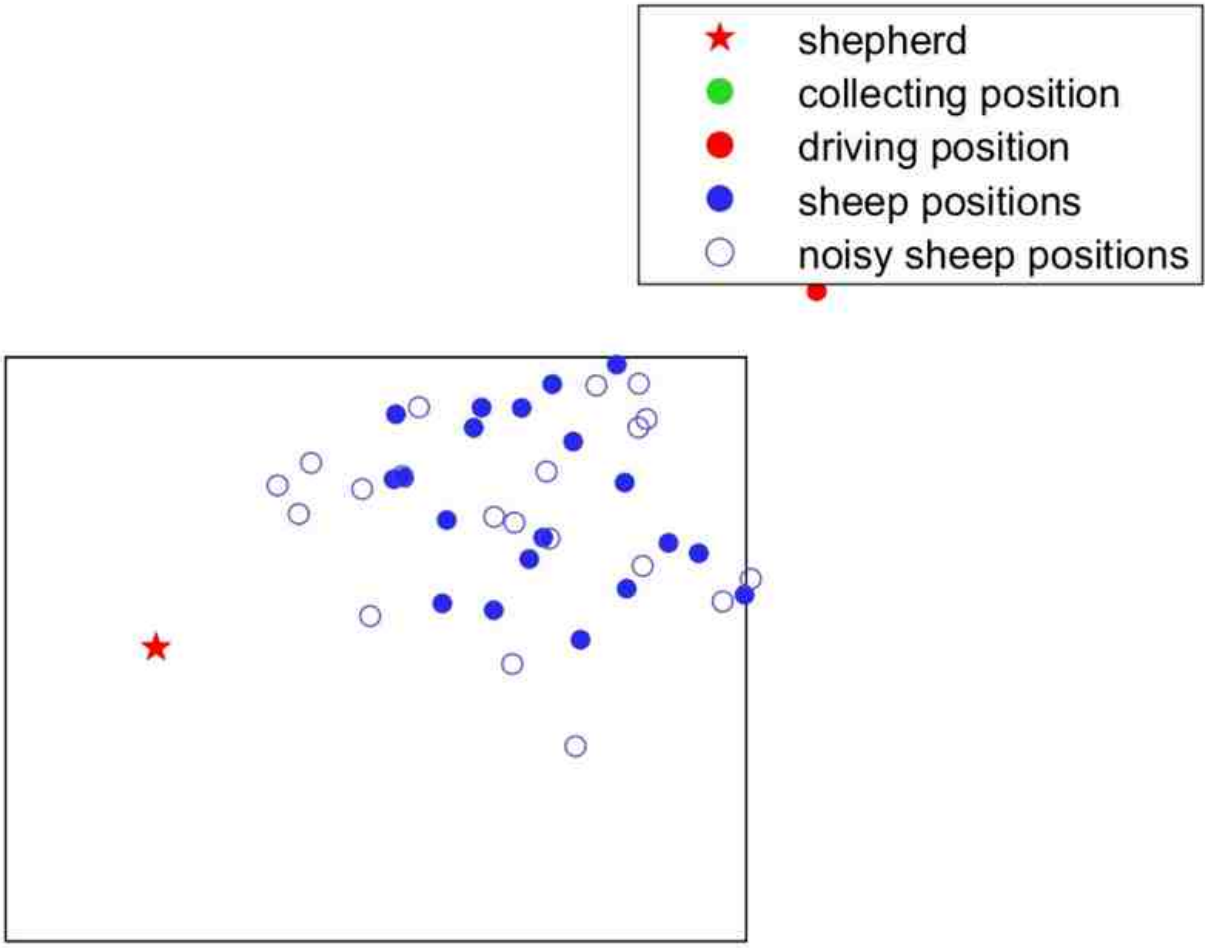
Results

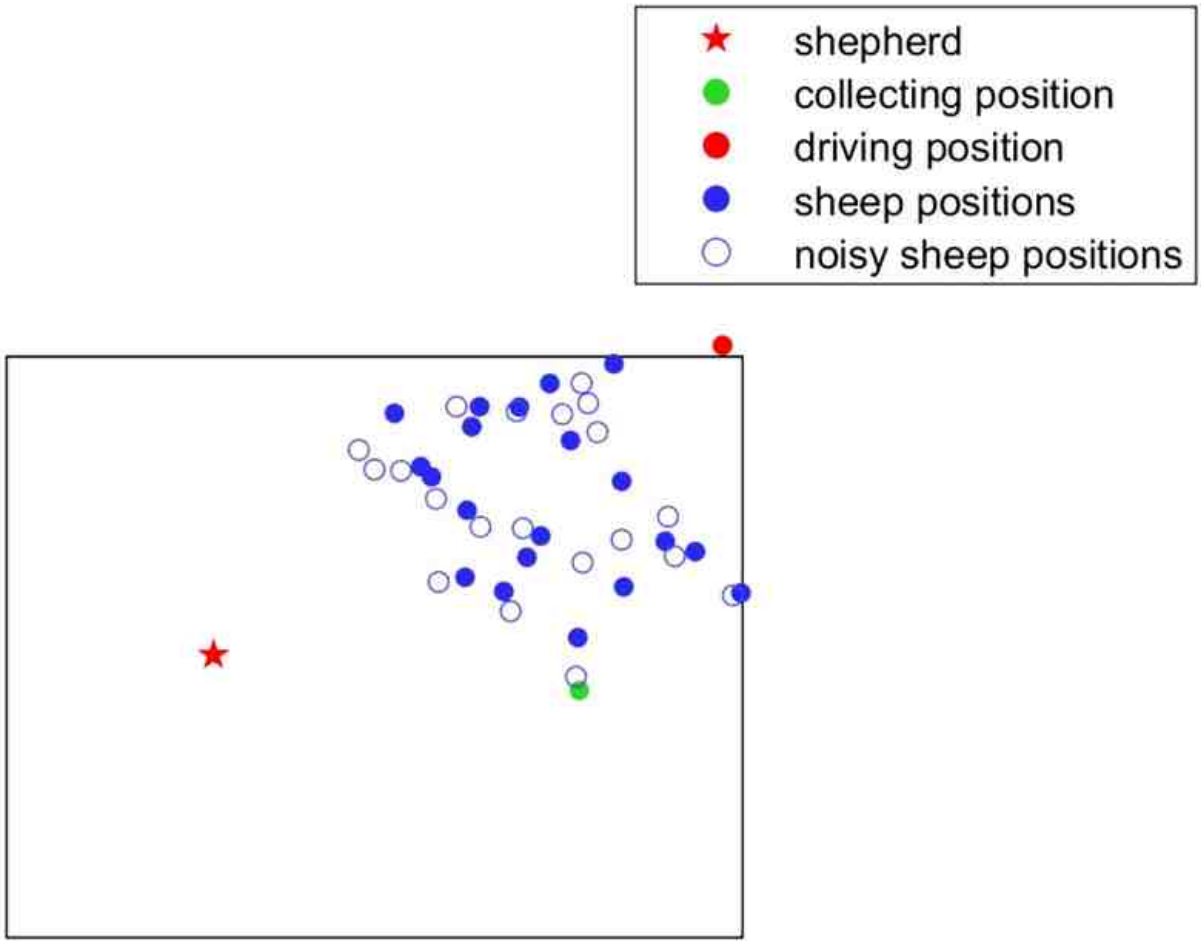
Testing is now running on my new implementation. Effectively, instead of the shepherd seeing a new random position for a sheep every timestep, what I have now done is as per the below:

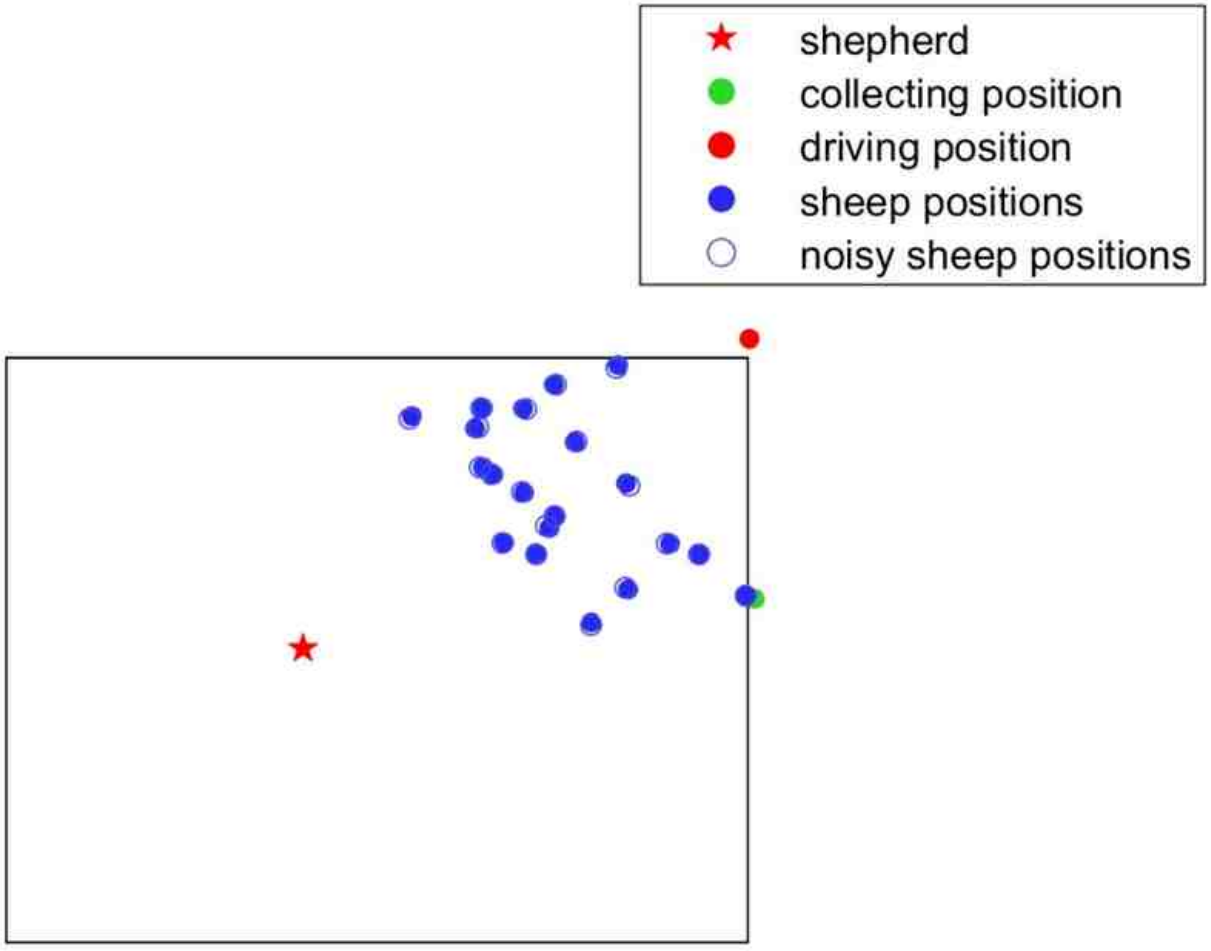
- matrix of sheep positions is generated

- shepherd is generated and distance from shepherd to the GCM is calculated
- a noise modifier is generated based on that distance (if the shepherd is further away, the noise is greater - it can't see as clearly)
- a new matrix of noisy sheep positions is generated based on the original sheep positions plus a random variable scaled by the noise modifier
- The shepherd actions and decisions are based off these noisy inputs
- For each time step, the positions of the sheep are updated as normal; however, the noisy positions are updated relative to the new sheep positions and how far the shepherd is from the GCM
- if the shepherd has come closer than the previous time step, the noise is reduced and so the new noisy position is closer to the true sheep position than the last time step

This is shown in the following consecutive screen grabs, where the noise reduces as the shepherd approaches the flock:




shepherd
collecting position
driving position
sheep positions
noisy sheep positions



The effective outcome of this new noise dynamic is that the shepherd shows much greater inaccuracy and uncertainty on positions to adopt at the start of a simulation. However, as the shepherd approaches the flock as the simulation develops, this situation resolves itself.

Next Steps

Testing of arena size differences. Analysis of data captured.

Current step: 5 hours

Running total: 111 hours

Final report completion **(2018-10-20 17:40)**

Aim

Provide update on report completion

Conduct

I have written the final report (pending final proofreading). It is 11 pages and 8 sections (not including appendices).

Results

The report details training and development of the neural networks, testing under ideal conditions, non-ideal conditions and analysis. Overall, I'm very happy with how it has come out, with proofreading to tie up any loose ends.

I have completed my final seminar slides as well, with the speech still to come.

Next Steps

Complete speech for final seminar.

Current step: 21 hours

Running total: 132 hours

All Work Complete **(2018-10-25 17:51)**

Aim

Provide final update.

Conduct

The final report and seminar are now fully complete. The final report is 7500 words.

Results

All work complete.

Next Steps

Submit report and journal.

Current step: 6 hours

Running total: 138 hours

BlogBook v1.0,
LaTeX 2$_\varepsilon$ & GNU/Linux.
https://www.blogbooker.com

Edited: October 25, 2018