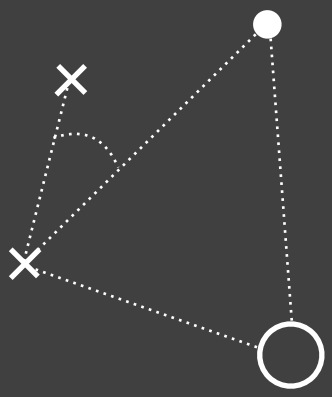


EX_MACHIDOG_

a machine learning research project

rohanpritchard@exmachidog.com

What Does EX_MACHIDOG Do?
EX_MACHIDOG is a machine learning programme which simulates the action of a sheepdog whose aim is to herd sheep into a pen. The algorithm was designed, written and is being developed entirely by myself in Python. Only very basic libraries were used; trig functions, pi, a random number generator, and a graph plotter. The aim of this project is not to create a perfect robotic sheepdog, but rather to model key principles in Machine Learning including Q-Learning, pursuit of the Markov state and exploring credit assignment challenges.

What's The Benefit of The Programme Learning Itself?
A simple - but rejected approach - to programming the sheepdog would have been to utilise the function controlling the reaction of the sheep and instruct the sheepdog to calculate at what angle to approach the sheep for the most efficient outcome.

However by using a reinforcement learning model instead, the sheepdog will be able to adapt to any change in situation, be that herding faster sheep, herding more sheep, herding a completely different animal with different reactions, a different sized field or pen and so on. The sheepdog may even learn to herd with another sheepdog (which may be attempted in later versions).

This same concept could then be put to practical use in many different applications. For instance an autopilot on an airliner, after monitoring and learning the reactions to a certain action the pilot makes, could become progressively more intelligent to the point that it is far safer to use even in extreme conditions than a human pilot. And if planes were to exchange datasets, all autopilots would learn how to overcome certain scenarios be that originally via pilot intervention or by the autopilot working it out itself.

The Reinforcement Learning Model
The programme is based on a reinforcement learning model. Whereby the agent (the sheepdog) has a cumulative reward. In the most basic version, this is that for every move the sheepdog makes, the reward is the sheep moving closer to the pen. For a single move (or single action), it observes the reaction to it's surroundings (i.e. what the sheep does) and learns this action/reaction as a memory.

This model of recording action/reaction is designed to mimic human learning. If a human were to be in the position of the sheepdog, we would soon realise that the sheep will always try to move directly away from us, this is because after seeing this happen enough times, we begin to expect this and we are happy to accept that this will always be the case (however we can never be certain that this is true). EX_MACHIDOG will never make this realisation, it can never decide that it is fact that the sheep will always move away because it hasn't been told this, and even if it were to see this occur a million times, this still doesn't mean that it will happen the million and first time, and actually the reason for the sheep moving the way it moved, as far as EX_MACHIDOG is concerned, could be down to a near infinite different factors of which it would never be able to locate the actual causal factor. What EX_MACHIDOG does instead, is acts upon all of the data that it has recorded from previous moves to mimic that it has developed this insight, this will be explained shortly.

On a brief side note; would an AI ever be able to raise a hypothesis? It seems that in an example like us watching real sheep and sheepdogs, we would happily raise the hypothesis that 'the

sheep will always move away from the sheepdog' by neglecting all factors except the sheepdog and sheep's position, and by completely making up the connection between them because it seems to fit. It seems obvious to us because it feels right, but why would we choose to neglect all other factors for example wind speed, or the steepness of the field at the time? And how would an AI know to neglect all the same factors when they are near infinite? Finally even if it did find the causal factor, how does it make up the rule that connects the action and the reaction? Perhaps it's this ability to create rules on our findings, which allow us to apply what we learn to different environments and in creative ways that could fundamentally differentiate us from AI.

We'll now get to the nuts and bolts of EX_MACHIDOG. The code and a large dataset is also available for download at exmachidog.com. Continued on the next page.

EX_MACHIDOG V1 - Simplest Model

In this model, there will be one sheepdog and one sheep in an infinitely large area. It will have an unlimited number of moves to herd the sheep into a circular area of a radius 0.5 about (0,0). The sheepdog must move 2 units in any direction it chooses, following this the sheep must move 1 unit directly away from the sheepdog (regardless of how far away the sheepdog is). The sheepdog must be able to move faster than the sheep so that in scenarios where the sheep is further away from the pen than the dog, the dog will still be able to go around the sheep and heard it in.

Action/Reaction Learning or Q-Learning

The dog assesses the scenario by generating a triangle with lengths: pen-to-dog, pen-to-sheep, dog-to-sheep. It then looks through it's dataset (or memory) to find the 3 most similar triangles to it's current situation. Out of the 3 most similar triangles, it will choose the one that caused the biggest reduction in the distance between the sheep and the pen (or at least the smallest increase if nothing else). Once it has picked the most suitable triangle from memory, with this memory effectively comes instructions to recreate the same move, this consists of the angle that the dog should move in relation to the sheep to cause the sheep to do the same thing as last time. It will then do this move (and add a tiny angle for reasons explained later), monitor the reaction from the sheep, and save this as a new memory so it can refer to it at another time should it be in a similar situation. This method of saving "I did action A in scenario S and it caused S'" is known as Q-Learning. Please note, taking the 3 most similar triangles and then choosing the one with the best outcome is a very basic model and most definitely not the most efficient means of generating a move, this will be improved later on.

Triangle Generation and Retrieval

Saving memories through triangles are ideal because information can be extracted from a memory even if the dog and sheep aren't in the same positions as the memory at all. For instance if the scenario is as it is in Figure 1, where the red dot is the sheep, the green cross is the sheepdog, and the grey circle is the pen, information can be extracted from Figure 2, providing instructions on what move to make next despite the positions not being similar at all. It is simply a very similar triangle to Figure 1 that has been rotated. The yellow cross on Figure 2 is the move that the dog took next to created the outcome that is saved alongside the triangle in Figure 2, it is this outcome that we want to recreate. The dog will then carry out the instructions in Figure 2 (in this case it would be about 330˚, or 30˚ anti clockwise). Thus producing Figure 3, this will be saved as a new memory along with the outcome (how much closer the sheep got to the pen).

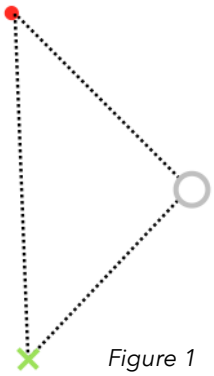


Figure 1

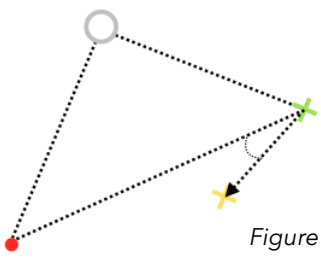


Figure 2

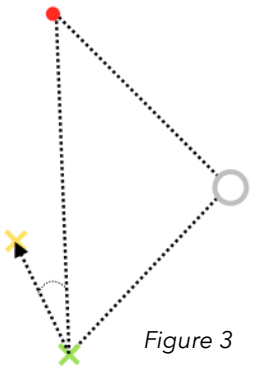


Figure 3

There is one issue to this triangle method that we must address. When EX_MACHIDOG searches for similar triangles, it searches for triangles with the smallest percentage difference in pen-to-dog, pen-to-sheep, and dog-to-sheep lengths between the current triangle and the memories it searches through. This is why it is happy to accept a triangle that is similar but rotated, but it is also happy to accept a triangle that is similar but reflected. Figure 4 represents a new current scenario that is very similar to the memory in Figure 2, however it is reflected. Currently, EX_MACHIDOG would see Figure 2 as similar, and if it were to pick Figure 2 as the most suitable memory to act from, the dog would perform the same 330˚ move as previous. However here, the 330˚ turn would be exactly opposite to the angle the dog should take, and so the sheep would move the wrong way, as demonstrated in Figure 5.

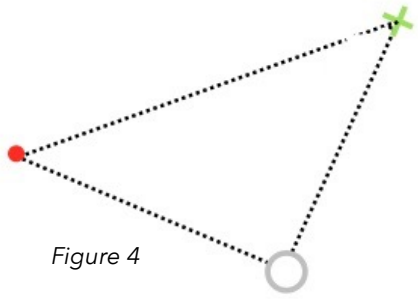


Figure 4

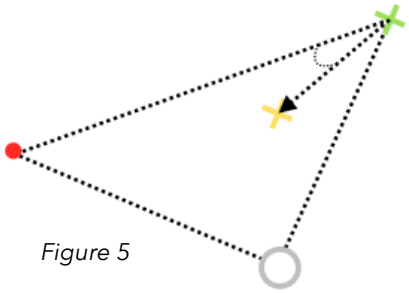


Figure 5

We must identify between similar triangles that are rotated, and similar triangles that are reflected. To do this, triangles will come under 2 categories. Looking at the angle at the pen, if going clockwise, the angle goes from dog to sheep (like in Figures 1 and 2) then it is classed as a 'positive' triangle. However, if going clockwise, the angle goes from sheep to dog (like in Figure 4), it is classed as a 'negative' triangle.

Whilst it would be efficient to tell the dog that in the case of reflections, do the opposite (so in this example go 30˚ clockwise), this would be cheating as it is giving the dog prior information. Instead, if the current scenario generates a negative triangle like in Figure 4, EX_MACHIDOG can only search memories that also contain negative triangles, and vice versa. This stops a triangle like Figure 2 ever being similar to the triangle in Figure 4.

EX_MACHIDOG's Memories in Python

EX_MACHIDOG's memories is a saved as a .txt file containing a single dictionary, this I have called the dataset. Each entry is an individual memory and is stored like so:

key = memory number
value = list containing:
[pen-to-dog, pen-to-sheep, dog-to-sheep, anglemovefromsheep, pentosheepnew, category]

The anglemovefromsheep value is the instruction to be used for next time, the angle that the dog should move along to recreate the same reaction. Pentosheepnew is effectively the outcome (or at least the only part of the outcome that we are interested in storing). And the category is whether the triangle is positive or negative.

Another list called 'current' contains the current dog and sheep coordinates and the 'new' dog and sheep coordinates (or the action and reaction), from this the triangle and all other information to be stored as a memory in the dataset can be calculated.

Memory Selection

As mentioned above, at this stage, the method for finding the most suitable memory is very basic. By calculating the percentage differences between each of the current triangle's lengths and the comparative triangle's lengths (the ones in the dataset), the sum of these becomes the memory's 'simiscore' (the similarity score), where the triangles with the lowest scores (or smallest percentage difference) will be the most similar. Calculating percentage differences are suitable because the difference between a very large triangle and a slightly larger one is less relevant than the difference between a very small triangle and a slightly larger one, this will be the case in the simiscore too.

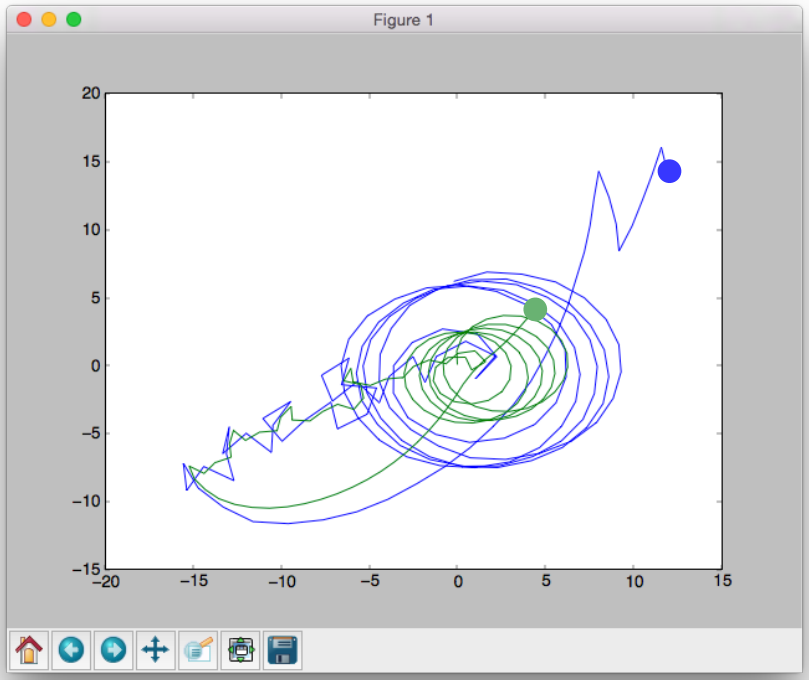
It is worth mentioning that there is a trade off between similarity and outcome, for instance it may be better choosing a very similar triangle that did not have as good an outcome last time than a triangle that had a brilliant outcome (large decrease in pen-to-sheep) but isn't as similar, this is because the same outcome is less likely to be as closely replicated. This trade off is one that will be explored later, and most certainly is not best considered with our current method of choosing the memory with the best outcome out of the closest 3.

Sheepdog Action and Random Angle

As briefly mentioned earlier, once EX_MACHIDOG finds the most suitable memory, it follows the same angle in relation to the sheep's position as it did in the memory to try to replicate the same reaction, however it also adds a small amount of randomness. This is to generate different actions. If the sheepdog were only to copy exactly what it did last time and then save this as a new memory, the dataset would only ever contain memories where the sheepdog does exactly the same thing every time. The sheepdog *must* try new actions so that it can reflect upon what reactions they caused. Initially, the random angle added on was simply a random floating point number between -10 and 10 degrees. However as the dog gets closer to the sheep, the sheep becomes more sensitive to the angles that the dog is moving at, to reflect this, the amount of randomness that the dog moves with is proportional to the distance between sheep and dog. This way the randomness only very slightly effects efficiency.

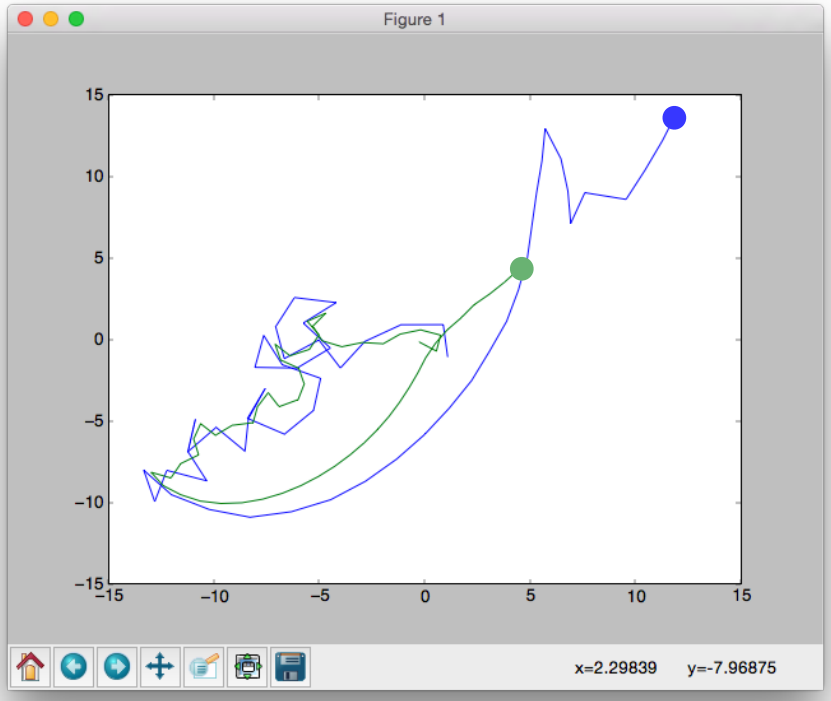
EX_MACHIDOG V1 in Action

The programme will start from a blank dataset. Upon running you input the dog and sheep's coordinates, the radius of the pen, and whether you want EX_MACHIDOG to save memories to the dataset or not (almost always you do want EX_MACHIDOG to save memories or it will not be learning). To test EX_MACHIDOG's rate of improvement, I ran the same initial scenario several times and monitored how EX_MACHIDOG's strategy improved. This scenario was the sheepdog starting at (12,12), the sheep starting at (5,5), and the pen of radius 0.5. The following graphs show the journeys the dog and sheep took. Blue being the dog and green being the sheep. The dots at the start of the line represent the starting points.



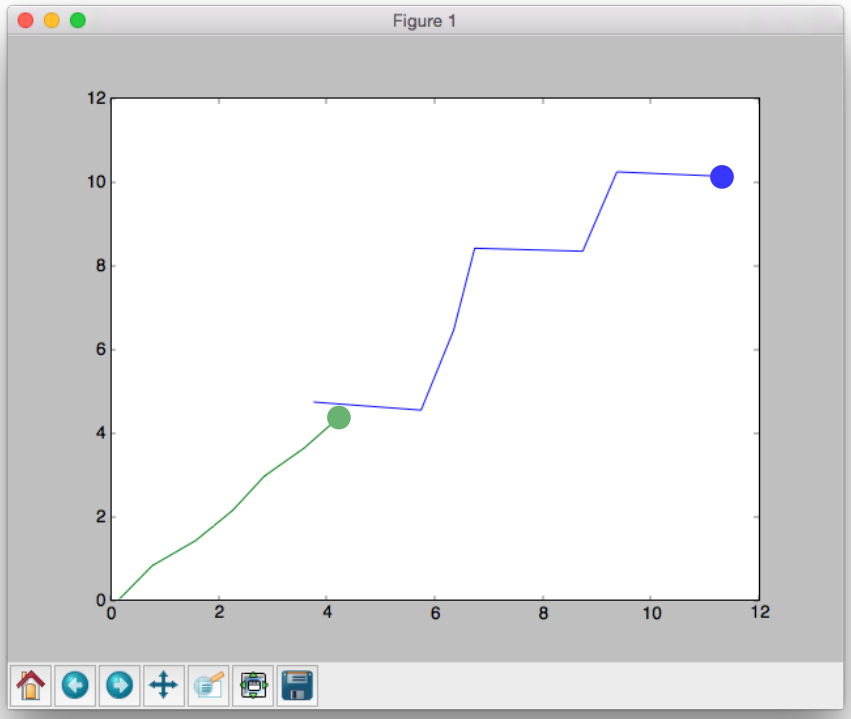
Graph 1
This was the initial run from a blank dataset. The Dog does a few random moves initially to develop a few memories. The dog will zig zag if it gets too close the sheep as it is restricted to moving no less than 2 units, so in order to stop itself from going past the sheep and pushing it the wrong way, it must zig zag.

This journey took a total of 163 moves. Dataset is at 163 memories.



Graph 2
In the second run, the sheepdog has already much improved. A similar strategy occurs here as in Graph 1 but it doesn't enter into the spiral that it does initially. It still fails to herd the sheep directly into the pen from the beginning and has to work out to go round it again to give it another go.

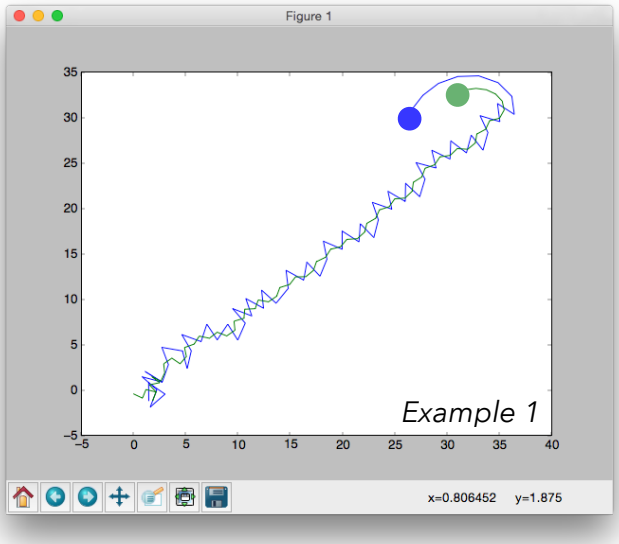
This journey took just 56 moves. Dataset is now at 219.

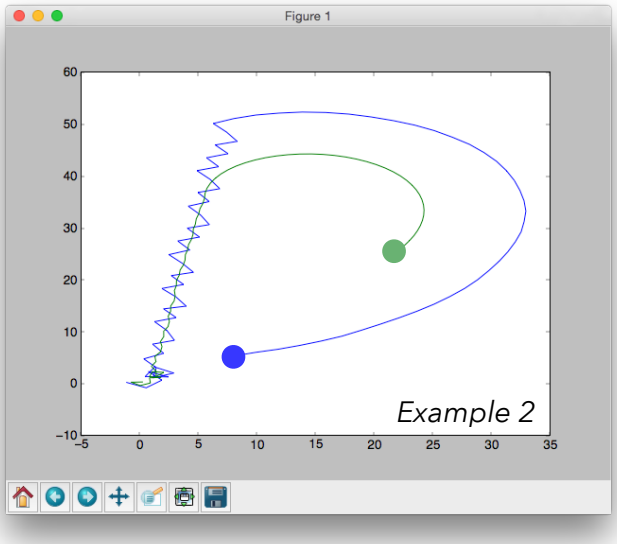


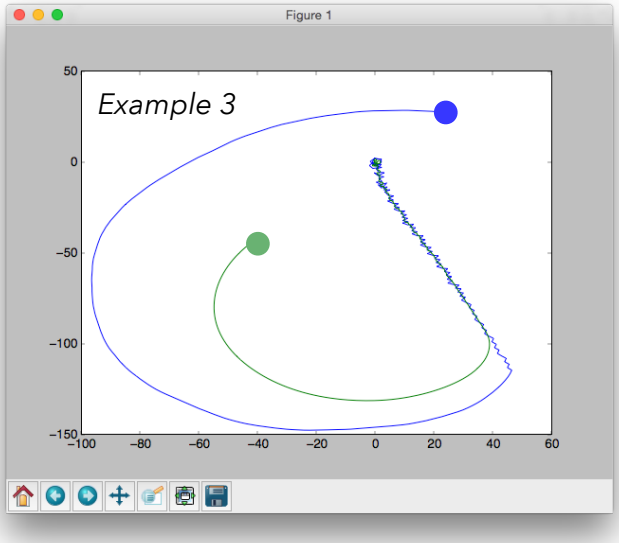
Graph 3
After running a number of more difficult scenarios (sheep starting much further away from the pen and the dog being very close to the pen etc) that took many thousands of moves thus allowing the dataset to grow, I ran the initial scenario again.

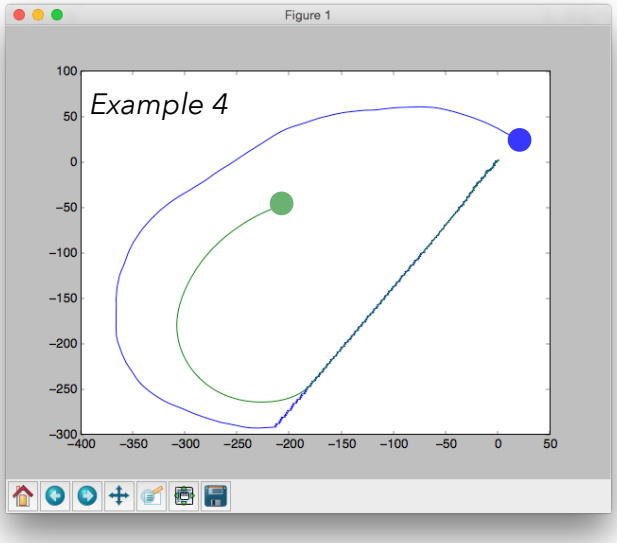
The sheepdog will now heard the sheep straight into the pen in just 6 moves, and it will consistently do this. The dataset is now at 30,827 memories.

With such a large dataset, the sheepdog can now very efficiently cope with far more complicated scenarios, like these examples:

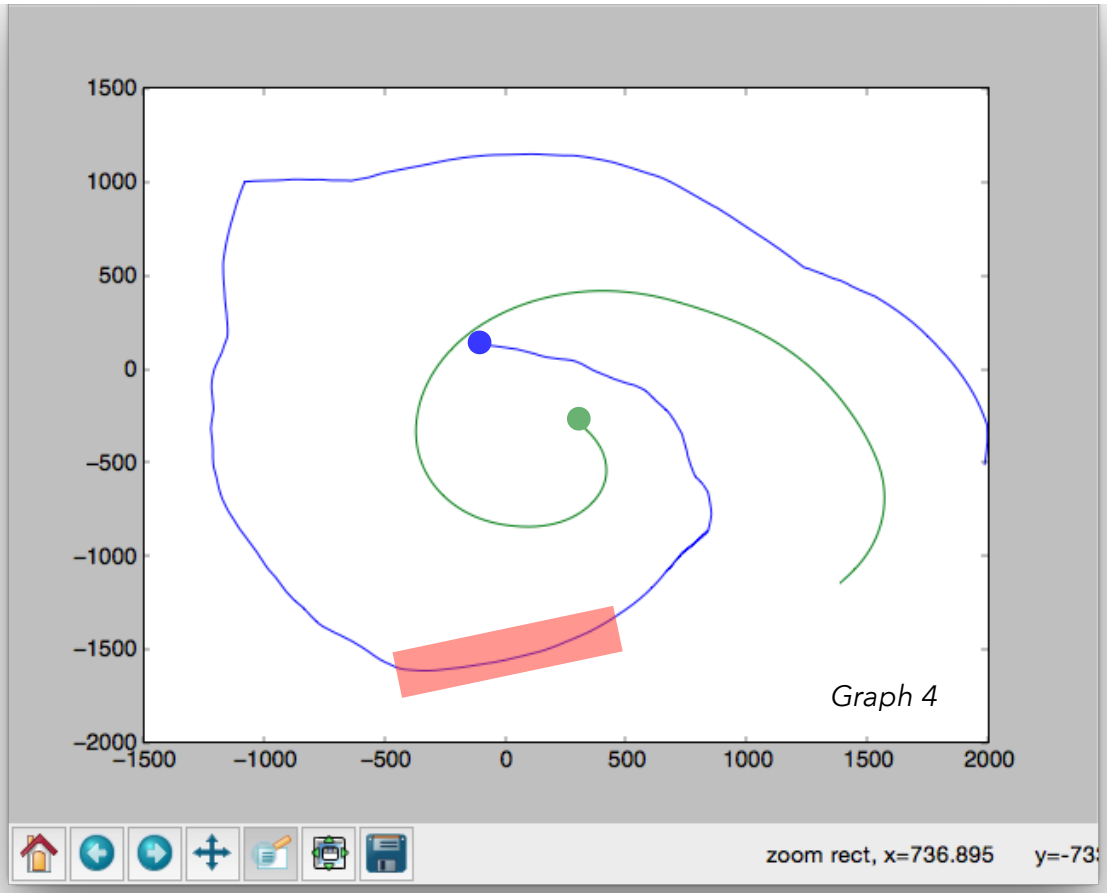








The sheepdog copes impressively well once the dataset is big enough, however there is still the occasional anomaly (these anomalies also become less and less frequent as the dataset grows).

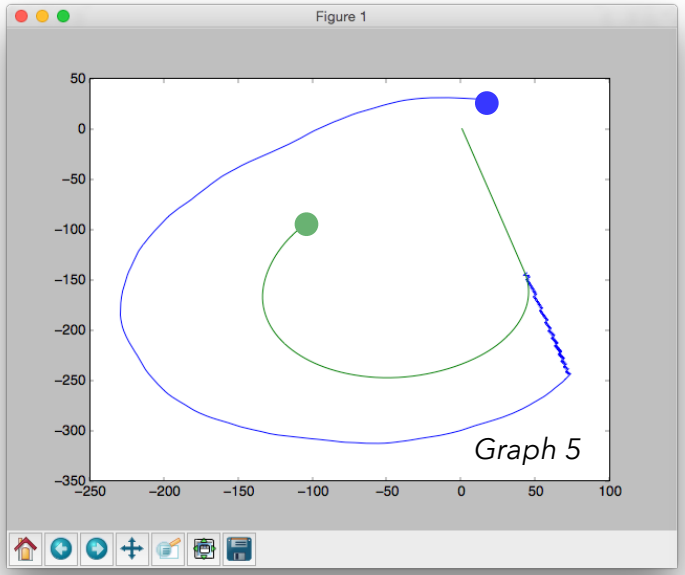


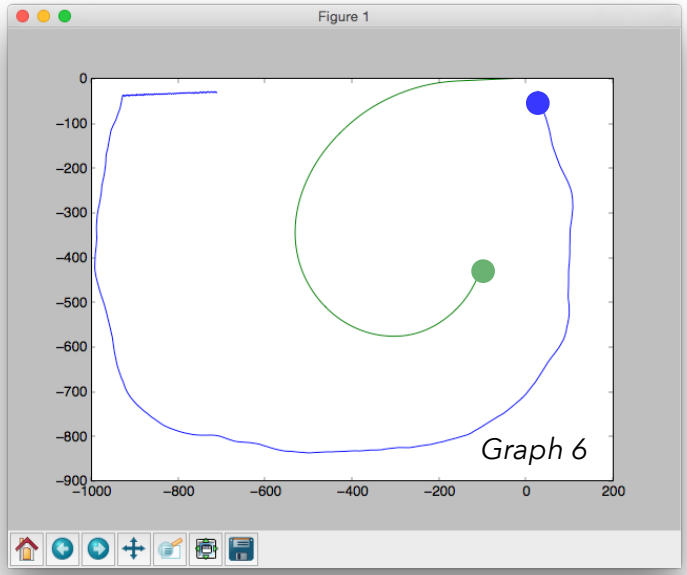
Graph 4

In Graph 4, on this attempt, the dog begins by doing what was to be expected (trying to go round past the sheep to herd it inwards) with a few wobbly moves along the way. But then at a point somewhere in the red area, the dog instead of deciding to herd inwards, continues on a spiral outwards, and eventually EX_MACHIDOG gives up after 5000 moves and the sheep finishes at a position much further away from the pen than the initial position. By looking at the simiscores of the memories it picks around the point where it starts to go wrong, it is clear that there simply weren't triangles similar enough to the position it was in, because the simiscores were so high. I think that, given enough moves, it would eventually have collected similar enough triangles with good outcomes that it would have herded the sheep in but it was forced to give up after 5000 moves.

Infinite Sensitivity Flaw

There is a flaw in the programme that EX_MACHIDOG will sometimes exploit. It is best explained in Graphs 5 and 6. Here, once the dog has gone round the sheep so that it is at the right angle to start to drive it in, the dog is able to direct the sheep into the pen despite being many units of distance away. This is because the sheep, the way it is currently programmed, will always move 1 unit directly away from the dog regardless of how far away the dog is. Graph 6 best demonstrates how this doesn't really seem to be realistic if the sheep was real rather than simulated. In the next version, this will be solved by making the sensitivity of the sheep proportional to the distance it is away from the dog.





Credit Assignment Introduction

A common issue with reinforcement learning models, is the credit assignment problem. Because the programme is always looking for the greatest 'instant' reward at the time of making the decision, it might choose to do something that is good for the short term, but is not the most efficient strategy for the long term. Imagine a real life scenario where a sheepdog would go right around the sheep at a large enough distance away from the sheep that it won't react, before then coming in closer at a much better angle to drive the sheep in. This in total, may be a much more efficient strategy than doing something similar to examples 1, 2, 3 and 4 above. Recognising this 'delayed reward' is something that must be key to EX_MACHIDOG's strategy in future versions once the sheep's sensitivity varies with the proximity of the dog, and it must be able to perform many actions before receiving the reward. However at this stage in Version 1, it is irrelevant because it makes no difference how far away the dog is, so a strategy like the one explained above would not be more efficient. This will be considered soon however.

EX_MACHIDOG V2 - Tackling Credit Assignment

So amongst other improvements that will be made in V2, is the introduction of proportionally sensitive sheep movements, and if the dog is further than a certain distance away, the sheep will not move at all. Programming this into the sheep's behaviour will be very simple, however this new behaviour introduces the credit assignment issue introduced earlier. The simplest way to begin to address this is by looking at combinations of moves. For instance by looking back through the dataset and choosing one move, predicting what will happen based on last time, and then choosing a new memory based on a scenario that it predicts it will be in, the dog is finding a combination of 2 moves without doing anything at all, so then after finding many combinations, it can choose the option with the best total reward no matter how delayed.

A Markov State

To now be looking at all the different combinations of moves before even taking the first move will take a lot more computing power, for instance if it were looking at all the different combinations of 5 moves out of just a 200 memory dataset (200 being extremely small), that would be a total of 320 billion possible moves (200^5). In reality this number would be smaller because the dog will only need to compare moves that are suitable for the scenario (i.e. there's no point comparing a move from a scenario completely different to the one it's in), nevertheless, it is now necessary to do some 'cleaning' of the data as to remove very similar memories with very similar outcomes.

If S_t is the state that EX_MACHIDOG is in (i.e. the scenario it is in as well as the dataset it has), and S_{t+1} is the next state after the dog has performed action A_t then S_t is a *Markov State* if:

$$\mathbf{P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, S_2, \ldots, S_t]}$$

This means that the dog will perform the same action and cause the same outcome when it is provided only with the Markov state as it would if it was provided with all the previous states. So in other words, the dataset that it contains in a Markov state contains all the information it needs to make any decision it needs to (or at least as much information as it can possibly have) without having any extra, unneeded information. This is what we must achieve to make combination calculations a viable option.

THIS IS BEING UPDATED REGULARLY. PLEASE CHECK BACK SOON.
-ROHAN PRITCHARD