Physics Representation and Search in Reinforcement Learning

Chris Dipple
Stanford University, XCS229ii Milestone 4, final paper
chris@dipple.org

Abstract

Reinforcement Learning is critically dependent on two things. How it represents the problem space and how it searches that problem space. The representation can be tabular, or a function, typically learned with a neural net. Tabular representations of continuous state spaces can get very large and accuracy depends on the granularity. Neural networks require a lot of training and are inherently expensive in machine cycles. In both cases sample efficiency can be poor. Sample efficiency can be improved with better search strategies, and/or modelling. We propose exploring these two issues with some classic Reinforcement Learning problems.

We used a number of methods to model our problem space and found results are generally fragile and difficult to repeat. There are a number of improvements that could be made to reinforcement learning simulators and approaches can be standardised more to encourage repeatability. Some elements of Learning Theory, such as variants of training, validation and test sets could be introduced to improve results too, we describe how this might be done. We discuss the use of derived/emergent features and how these might be discovered and how declarative function approximation might be used. We limit our research to Physics based problems and exploit inherent constraints in that domain, and how those discovered constraints can be 'transferred' to other Physics based problems. We put forward proposals for how this can all be achieved in future work.

Introduction

This research is motivated by the need for edge computing on low cost devices such as IoT, and in particular autonomous vehicles and robots for space exploration. In particular physics based environments. Environments where some degree of learning is required locally because the environment is not entirely static or fully observable and/or centralised learning resources are not immediately available.

Tabular representations of Reinforcement Learning state spaces are simple when the state space is small, both in terms of discrete values modelled and the number of dimensions required. Unfortunately they get very large when dimensionality is high and/or each dimension has many distinct values (or value ranges in the case of discretised continuous state spaces). However often a state space is only 'sparely' explored, especially in higher dimensions, when a Reinforcement Learning algorithm is learning a policy. Potentially a great deal of space can be saved by not pre-allocating space for all possible states, especially in higher dimensional spaces. Also, granularity of the state space may only need to be low in certain places but of higher granularity elsewhere, can it be adaptive?

Another characteristic of Reinforcement Learning problems is the exploration/exploitation trade off is poorly managed and much time and energy is often wasted with simple search strategies such as epsilon greedy which can thrash about locally and not do a great deal of useful exploring. The credit assignment problem cannot be qualified in many problems until some reward (not necessarily the best reward) has been identified, so efficient initial search is desirable. There are well known techniques for regret reduction, which can be helpful, but again only once some reward has been identified. Statistics based intrinsic rewards can be very helpful, various error bound confidence based methods, such as upper confidence bound or Thompson sampling can be useful but will be inferior to model based methods [this requires some justification, the basis of another paper]. Are there better techniques, in our case domain specific to Physics, simple techniques, for exploring state space and finding rewards in a sample efficient manner that can help bootstrap the learning process and reduce thrashing?

- The system will use OpenAI Gym as a simulator and source of data.
- The system will output a model for running CartPole and MountainCar, with discrete actions for now. Later to be generalised to more complex models, many with continuous action spaces.
- Success criteria will be efficient and stable learning with minimal resources (no GPU or TPU required or excessive memory). Above all it should be as sample efficient as possible given it will likely be model free initially.
- There are many challenges. RL is a huge area of study and we are addressing some of the fundamental issues. This project is a beginning not an end.
- There are aspects of RL problems that might be used to characterise them, over and above the

common characterisations (Continuous or not, stochastic or not and others). For example CartPole receives rewards on every iteration, until it fails to do so. While MountainCar receives negative any rewards until it achieves it's objective, or the number of negative rewards exceeds a threshold. To succeed it cannot be greedy all the time as sequences of coordinated actions between 'high order' features is required for a problem solution. How to identify different search space characteristics and tackle different problems automatically with a common framework is the objective here.
- Search strategies and making the most of a simple and common 'functional' representation (namely discretisation) is the limited objective of this project. It is theorised that it is possible that a crude solution, achieved quickly could be reused to train a more complex solution which could then be fine tuned. Linking sequences of states/actions together is necessary.

There are five primary references supplied below. A mix of discretisation schemes and state space search strategies. Some papers can be simply implemented, such as [Sutton, 2018] for tile coding is referenced in other papers as it acts as a building block. Other papers are aspirational for the authors in this project, such as [Ecoffet, 2021] and [Badia, 2020] and might be the subject of future work.

The general problem area covered is functional representation of policies and search space principles to quickly uncover rewards, and to bootstrap useful policy learning. Different papers take different approaches and there is no definitive, single approach.

The primary objective of this study is to explore relatively simple methods, that perform well across multiple criteria while efficiently traversing the problem space and mapping it's contours to quickly discover a reliable policy. Different problems carry different challenges, so any scheme must be adaptable and flexible. The Agent7 paper [Badia, 2020] suggests taking a range of policies and exploration strategies that adapt to the problem space. Similarly [Ecoffet, 2021]. This would seem to be the way forward, hence the papers chosen. Often the problems can look very different, yet it would seem that the brain uses essentially that same subcomponent, the cortical column, thousands of times over, utilising different types of input while processing a a generic way. This would seem a sensible way forward with Reinforcement Learning too. An adaptable generic component, that can operate and cooperate with similar units to solve a bigger problem while tuning it's own representation and problem exploration strategies. See [Hawkins, 2021] as a background reference.

There is a problem currently when policies are transferred from simulated environments in which they are trained and the real world. The real world is noisier. Inserting Gaussian noise helps, but better, declarative functional approximation would make these issues more explicit and hence easier to compensate for. We hope to have more to say about this later.

This is important because many potential applications of RL in edge computing situations where constant automated retraining is indicated are often prohibited currently given the limited resources available. In particular applications such as autonomous vehicles and robots in space.

Prior work often concentrates on specific solutions to narrow problems and often seeks to obtain state of the art results, that might be impressive in their own right but are often not fully understood and often difficult to replicate. Academic reputations are at stake of course. Robust results will likely come from a deeper understanding of when and why a solution does or does not work well. More research on fundamentals is needed, with wide applicability to other similar problems, in an explainable (and causal) way. Initial conditions are one things, but if these change then counterfactual reasoning is required to frame new solutions, that are equally valid.

In the specific domain of primary interest, which is physics based problems. There are a number of higher level features that can be discovered, such as turning points, zero crossings and others. When finding these points, the current and previous value must be known. The system must learn the relationship between features and how to exploit these relationships.

The use of Kalman filters, later, would be sensible and this will be explored in future work. A Kalman filter can also be used to suggest 'missing' forces (gravity, possibly from multiple sources), when errors are systematically biased, and could be used to improve the model automatically.

Unresolved Issues

The objective of the paper is to explore issues and suggest several ways forward for future research. So, the objective here is to do exploratory research, to inform knowledge about how to later proceed to obtain the desired objective. The desired objective is to have a tractable, physics model 'aware' solution that is optimally sample efficient, tolerant to change, and generally applicable to Physics problems (initially autonomous vehicles in space). This is of course a much longer and broader scope.

Overview of future work

The central hypothesis is that using the constraints in Physics, initially supplied as meta-data by an improved simulator environment. We can then search the problem space in an principled way, gathering an understanding of the problem space and various landmarks in that space to find and later navigate and maximise the reward(s) defined by that space.

Methods

Task Definition

As previously stated, the purpose of this paper is to explore physics based Reinforcement Learning problems. We have examples of two problems, how to stay in the same place with CartPole, a sort of dynamic balance, and how to explore efficiently when sequences of actions are cumulative as with MountainCar. In both cases a simulator is used and provides a reward on every iteration in an episode.

Given this environment, with particular emphasis on physics, we can add some constraints to the environment, for example it's a linear environment, actions are forces that act on objects and change their positions. A problem is that the simulator supplies state space attributes with values, but those values, and any relationships between them are not given (such as position, and velocity are given, but velocity could easily between derived, in fact we would rather that were done). The relationships between, state space attributes, and actions would be very helpful information given the inherent constraints in a physics based environment like this. Sequences of states are important, and significant states, where sequences change are of interest. By discovering the correct sequences, we can navigate and determine the optimal actions to navigate quickly to rewards.

We are hoping ultimately to discover a naive, machine declarative. physics based model underlying the data and use that for future inference, hopefully between different experiments too, as well as parametrised variants of the same problem. Said model will evolve as it must be capable of negation. This is the essence of Science and the likely direction that must be taken for AGI.

Datasets

This a Reinforcement Learning project so a simulator was used, namely the OpenAI gym simulator. Of immediate interest is the CartPole-v0 (and v1) and the MountainCar-v0 environments. If there is enough time then the MountainCarContinuous-v0 and possibly Pendulum-v0 may be used, they however have continuous action spaces and hence require Policy Gradient methods.

Both CartPole and MountainCar have discrete action spaces and with appropriate discretisation can, in principle, be solved by relatively simple value based methods such as Q-Learning.

CartPole State consists of 4 items:

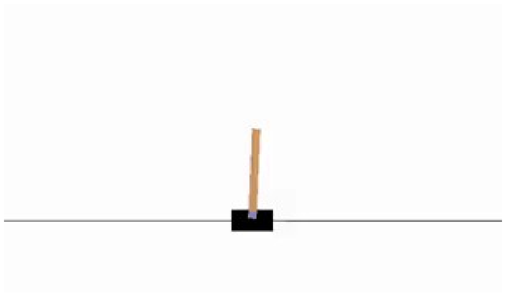
Type	Low Value	High Value
X Position	-4.8	4.8
X Velocity	-Infinity	Infinity
Pole Angle	-0.418 Rad	0.418 Rad
Pole Angular Velocity	-Infinity	Infinity

The CartPole Action space is discrete with the following values:

Action Value from gym	Meaning
0	Move cart left
1	Move cart right

Now, the CartPole state space contains infinities. For the purposes of discretisation and with experience of running CaretPole more realistic ranges for discretisation of those infinities are +/- 4.0 in both cases. Again, could this be automated?

A visualisation of the CartPole looks like this:



Note that there are several inconsistencies with this. The velocity never actually gets very high, let alone to infinity. This causes some difficulty for discretisation and a more sensible value range needs to be supplied. The data returned by the simulator is limited and can be misleading, this is not helpful for experimentation.

For each time slot when both the cart and the pole remains within their bounds, a reward of 1 is given. When an upper bound, depending on the version of CartPole (v0 = 200, v1 = 500), is reached then the system terminates. So, there are two conditions under which the system terminates, with the 'done' flag set, namely when position bounds are broken or the maximum iterations is achieved. So, the number of iterations must be taken into account to determine which case is true. This is poor simulator design, which encourages human intervention to correct. It is suggested it all be made clear and explicit as meta data, it can then removed as a problem solution evolves to make minimal assumptions.

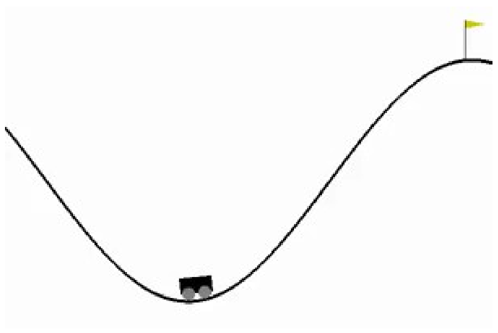
In the case of MountainCar the State space is:

Type	Low Value	High Value
Car Position	-1.2	0.6
Car Velocity	-0.07	0.07

The Action space is again discrete (for our purposes, though there is a continuous version, which presents some additional challenges):

Action Value from gym	Meaning
0	Move car left
1	Do Nothing (no force)
2	Move car right

A visualisation of MountainCar is:



MountainCar will succeed when the Position reaches 0.5, setting the done flag and terminating the episode. For every other time slot when the car does not reach 0.5 then a reward of -1 is given. When the cumulative reward is -200 then the system will terminate with done set. So, again there are two situations under which termination occurs. Which is not ideal.

In this problem the force imparted on the car by an action is not enough in itself to push the car up the 'hill', so the car must build up momentum to achieve success. Essentially it must swing back and forth, which requires 'insertion' of an intermediate goal (to climb up the opposite 'hill' to the reward, to take advantage of the implicit gravitational force.

These two simulations are both based on physics. However, they are very different. One receives positive rewards, the other negative for each timeslot. The objective for one is around a single state, which must be maintained for a count. However MountainCar has a single objective, which is, when the goal state is achieved stop the count. The generic problem, which is what we would like to solve, is to identify the reward state (or a stable state in the case of positive rewards). For both problems, and any similar problems and to discover new scenarios.

Motivation

The datasets considered above are well known and well understood. They are relatively constrained, yet have sufficient richness to be challenging. They can also be rapidly run, making extensive experimentation practical.

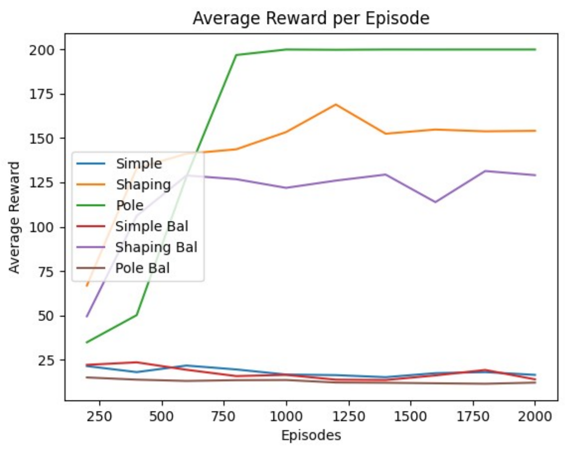
Model description

For both problems we explore 3 separate baselines using standard Q learning. There is also an interesting tweak to Q learning, which is referred to as 'balanced' Q learning below. The balance term is motivated by reducing the accumulated reward as well as adding in new reward, it does seem to make a significant difference, sometimes positive, sometimes negative.

Cartpole Baselines

Parameters used by the Cartpole experiments are summarised here.

episodes	2000
alpha	0.2
gamma	0.9
epsilon	0.9 to 0.1
discretisation	8, 8, 6, 12 or 1,1,6,12
search	See discussion



A summary of resources used is shown in tabular form below. Note the order is the same as the legend above. A single CPU core was used, so elapsed and CPU time are similar.

Time	Memory	Episodes	Steps
3.0128	85.608	2000	35955
23.7299	74.018	2000	284446
26.9906	117.774	2000	321964
3.029	68.362	2000	34591
19.4651	75.884	2000	232582
2.2659	125.792	2000	25463

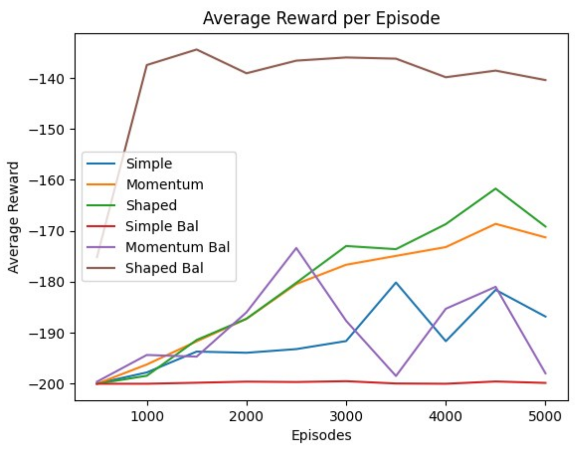
In both cases here Pole works well, by Pole we mean a discretisation where both the Cart Position and Velocity are effectively ignored (by putting all values in a single bucket).

Shaping means setting the episode reward to -200 when the pole is balanced for less than 200 iterations in an episode.

MountainCar baselines

These were parametrised as follows.

episodes	5000
alpha	0.2 to 0.1
gamma	0.9
epsilon	0.9 to 0.1
discretisation	25, 25
search	See discussion



Again, we supply a summary of resources used in the table below.

Time	Memory	Episodes	Steps
125.4895	193.626	5000	955159
123.2842	177.341	5000	910109
127.0325	171.852	5000	901668
140.917	145.346	5000	998939
132.1484	171.476	5000	949153
101.9572	171.863	5000	706841

For MountainCar Momentum means rather than using epsilon greedy, we use epsilon ‘the same action as in the last iteration’. This provides a longer sequence of the same action, enabling better exploration, crude but effective.

Shaped runs in this context mean adding the difference in velocity between this time slot and the previous time slot to the reward supplied by the simulator, biasing the reward to the ‘direction’ of the action.

Balanced

In both cases above, the first 3 runs (the non-’balanced’ runs used the standard Q learning update. Where $Qo=Q(s',a)$ however, the ‘balanced’ runs use $Qo=(1-\alpha)*Q(s',a)$. This makes a big difference. The exact reason why this is so needs further investigation to fully understand. The resulting Q-Learning update now becoming :

$$Q(s',a)=Qo+\alpha*(r+\gamma*max(Q(s))-Q(s',a))$$

Infrastructure and Assumptions

Minimal infrastructure was used for these experiments. In all cases the tests were run on a single CPU core on a midrange desktop under Ubuntu. Application memory use was modest too. All experiments were written in Python. The authors preferred language is Rust, and with a Rust based simulator too (see closing remarks), resource usage should reduce to a minimum.

We have assumed standard parametrisation of the Greek variables, alpha, gamma, and epsilon remain constant (or decay at the same rate) for each baseline.

Results

Metrics

For our purposes an obvious baseline is to solve each of the above problems within the problem constraints defined in the previous section. To do this in a discretised environment in a similar way for both problems. Then, and only then can we hope to incrementally improve the scores by seeking more efficient discretisation schemes or better search space navigation.

Success criteria to measure performance are:

- How generic is the solution and does it work for similar problems
- How sensitive to hyper parameter settings is the solution? A robust stable solution that is readily understandable and explainable is desired.
- Sample efficiency
- Elapsed time
- Minimise Memory
- Minimise CPU

These performance criteria can be measured readily in a Reinforcement Learning setting. A generic solution would allow different but similar problems, in a generic category such as physics, to use very similar code bases. Minimisation of the differences (or self tuning) hyper parameters should be a common objective too. The number of samples required is readily measured and is an obvious metric. Measurement of elapsed time, memory and CPU are readily automated too).

The second criteria above is of crucial interest. The better understood an algorithm/parametrisation is then the more robust and repeatable it is likely to be, and hence easier to extrapolate to similar problems.

An objective/heuristic would seem to be, for these problems in a physics domain:

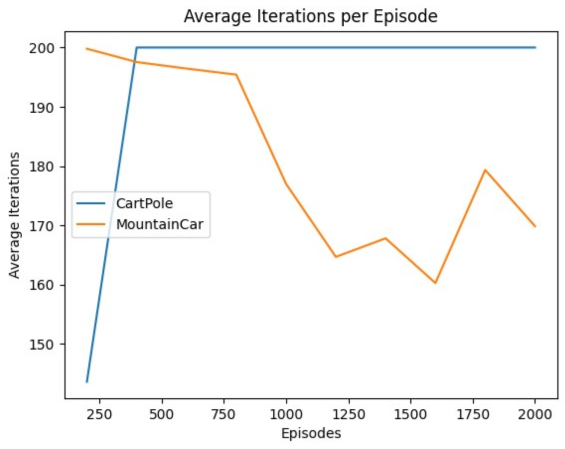
> If we receive a positive reward each time step then identify a 'turning point' and take the action that gets us nearer to that turning point on each iteration. Given that said stable/turning point is likely to be associated with success if we stay near it.
>
> Otherwise, search the environment as rapidly as possible for a reward signal. As rapidly as possible might mean using momentum when applying an action from a stationary position means we cannot achieve the endpoint of the range of possibilities for a state. A sequence of states must be considered to observe an emergent behaviour such as the momentum gained on a slope to help climb another slope.
>
> Some of the challenges are. What is a turning point? What is a stationary position? What is stability or high/low velocity? Which states indicate momentum, as we would assume zero knowledge about the state values as the start of the learning run. How does an action correlate with a change in state? How do we track change over a set of time steps / episodes. How do we model 'external' forces, such as the gravity implicit in both problems?

Are these heuristics general enough to solve other similar problems? Probably not, this is a journey of discovery.

These proposed heuristics are an attempt to uncover declarative information in the problem space that can be used later by symbolic/causal methods to declaratively understand the problem space, increase sample efficiency, make changes (such as the move from simulation to the real world) smoother, more principled and faster. We see below, that even simple reward shaping and better modelling of correlations between the state space and the actions space can greatly improve sample efficiency and speed up learning. How to do this in a principled and generic way is the challenge?

Tiling

Tiling was next used instead of explicit hand tailored discretisation. Findings were similar to the hand tailored discretisation runs, only discretisation was simpler. Some tuning was still required. CartPole worked best again when the cart position and velocity were ignored. So, some effort was saved, but there are not dramatic benefits, except for simpler code. Run times are similar too.



In both cases we used the optimisations learned in the baseline cases above, to inform these tiling solutions.

PSO optimisation

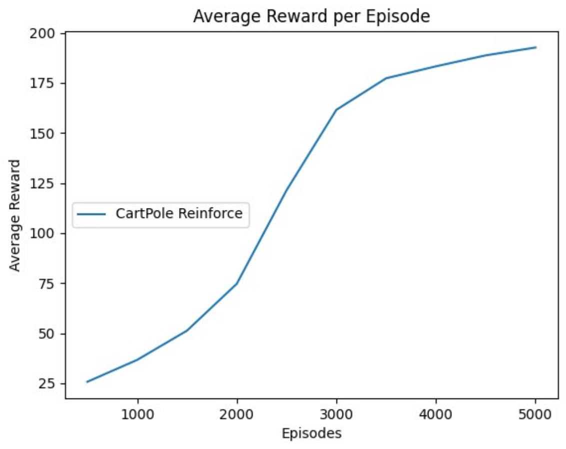
It is not possible to produce a nice linear learning graph for PSO. However, the basic metrics do tell a story. For a simple domain like CartPole a perfect score of 200 time steps per episode was produced 100% of the time. For MountainCar of the test runs done, they succeeded 100% of the time too with an average score of -127 and a best score of -86.

PSO works well for small search spaces (and can easily be parallelised). It is not necessarily as sample efficient, but using a simulator helps mitigate this problem.

Experiment	Time	Memory	Episodes	Steps
CartPole	0.4726	34.658	200	26417
MountainCar	67.5237	174.063	4800	957836

Policy Gradient

Finally to demonstrate a different approach we used Reinforce for CartPole, which gave nice steady learning. However this did not work for MountainCar as expected, there are many innovations to make the likes of MountainCar work for Policy gradient methods to tame the variance of simple Reinforce. This work is fascinating but does not impact greatly on the objective in this paper. We highlight this fact here for completeness. Like PSO, Reinforce may work well in narrow and well behaved sub search spaces and is a useful tool as such. This is the graph:



Again, machine resource usage is light (we use a simple linear function). These further results confirm that there are many other degrees of freedom available to solve problems with with model free methods. However, there is only so far we can go. Exploring modelling, given the constraints inherent in physics based problem domain would seem to be the way forward.

Applicability of ML Theory

Being an Reinforcement learning project Learning Theory developed for supervised learning is not obviously applicable. However, there are compensating things that could be done. The lack of Training, Validation and Test sets is an obvious case in point. Multiple runs against a problem with averaging is an obvious way forward. However, this has limitation, it was observed that some solutions worked very well much of the time, but on some runs performed badly. This may be due to the exact initial setting of the observed parameters and/or the exact sequence of random number generated by epsilon greedy or other exploration scheme. Likely both. So an obvious way forward would be to isolate these likely causes (by initialising the random number generator) and determine the effect of initial conditions on outcomes. Conversely, being able to set the seed in the simulator, to generate consistent sequences would help too. Isolate and understand individual variables. Having better mete-data from the simulator would help too, for example this variable is the velocity of this variable between this and the previous time slot. Also, this action is a force acting on this variable in this direction. This metadata should be configurable, when we start investigating a problem more can be supplied, later this can be reduced and the missing data inferred by the system. At the moment a position and velocity might be supplied, if we had meta-data we could understand this relationship, then remove the velocity feature and derive it ourselves (acceleration/jerk too potentially). Full disclosure then flexible removal of data from a simulator , builds insight, one is not overwhelmed by complexity initially and solutions, probably model based, can then be honed to let the system fill in gaps in a principled way that later might be used on other problems in the same domain. Transfer learning of a new kind.

The proposal is essentially, understand the features coming from the simulator. Learning the relationships, fill in gaps, take small steps and test with many variants. Learn what can be learned, use counterfactuals, and test against variants that obey the same rules at a certain level of abstraction.

Comparision of Results

In this work we created a baseline. In fact a number of successive baselines, to explore the problem space. Now the question is how to improve these solutions, producing more generic solutions and more robust learning. With a view to simplicity, declarative knowledge if possible and reusability (such as transfer learning).

We explored discretisation. The granularity makes a difference, and a single granularity is not always appropriate as the 'contour line' of the problem space vary over the space. Tiling can work well, and is easier to set up. However, knowledge of the variables in the observation/state space of the problem and there correlation with the actions is important information and tiling might be counter productive and hide that if used naively. We found this with CartPole, regardless of the fact that actions moved the cart back and forth, the indirect effect on the pole angle is more significant to the problem solution than the cart position (most of the time). So, understanding the relationships between cart position, or more importantly it's acceleration in one direct or another and the equal and opposite force this induces in the pole, would help immensely.

To Do:

1. Develop derived, multi-step features as outlined in the Reasoning section. The common features in physics domains can be learnt too.
2. Explore the benefits of biologically inspired search, such as Particle Swarm Optimisation. This could inform further work and allow us to compare and contrast results.
3. Outline further next steps. It is likely that the physics features will require much more work, in particular autodiscovery. Then modelling this in a more declarative, explainable, manner using symbolic and causal reasoning is likely to be the future challenge.

Discussion

In this paper we have explored some variants of solutions to a couple of well know 'Classical' Open AI gym problems. Both physics based. These simple physics domains are well understood, we can use that understanding to more deeply explore the solutions given by standard methods from the RL literature, with some variations. This was done to highlight limitations and opportunities for improvement that might be made if search and problem space are explored in a way that is informed by what is known about physics. This is early days, but we hypothesis that using higher order attributes might dramatically improve search efficiency, provide for higher order learning which can be 'transferred' to similar problems and dramatically speed up sample efficiency and learning. This is how humans view these problems. If we take baby steps with machines, consolidate learning as we progress, then perhaps we can solve a whole, and very important, domain of problems, sub-domain by sub-domain. Rather than looking for a single methodology to solve everything. This implies a sort of hierarchical structure to problem solving,

Some higher level features (meta-features) available in a physics based problem space are:

Observation/State space meta-features

- Determine positional/spatial points and how they relate to velocity and other time derived data. For example Velocity can be derived from position (in each orthogonal dimension)
- Derive those higher order features and add to the model if not supplied
- Characterise higher order feature where something changes as, turning/inflection points, stationary/saddle points (or sequences of the same), zero crossings (by direction) and others
- Mark states where rewards are high
- Derive sequencing of sequences of states between above and how transitions between them are affected by actions

Action space

- if discrete, characterise by effect on above
- if continuous, distinguish different parts (zero crossing) and then characterise by effect on above with sign and magnitude

Sequences of states between the above higher order features will likely have the same action applied to them. Join up the states between meta-features. This is then an opportunity to ‘fill in the gaps’ from the discovered structure of the problem given the linear nature of Newtonian physics.

Significance of Results

The results of the experiments so far hint at better solutions. We have hypothesised what those solutions might entail above.

How to improve results

There are a number of future research directions outlined here. It is likely that a domain specific model based approach will work best. One way to do this, using a discretisation scheme would be to chain similar states together and identify sequences of such states and the ‘best’ action to navigate through them quickly. We can make some predictions about what then might happen under some new circumstance. For example, if in a discrete state we transition to another state given an action then that transition is probably sound. However if we transition to a different state to previous transitions then the discrete state, at this location, can be assumed to have more structure than the current discretisation level can capture. Hence locally (and possibly in adjoining states) we might wish to increase the level of discretisation to capture the difference. Of course we are assuming actions here are ‘optimal’ in some sense and it needs to be further clarified how this is determined.

Classically state translations given actions would be done with statistical methods such as Upper Confidence Bounds or Thompson Sampling (depending on our allegiance to a statistical sub-species). This does not work so well with a discretised, value based solution so we suggest the alternative opportunity for improvement above. These statistical methods work better with Policy Gradient methods where discretisation is not a thing and we do not have (or cannot observe) the constraints imposed on our problem space by physics. This said, this statistical approach might help inform the above and hence might be done in parallel.

As previously stated, individual time steps need to be consolidated/sequenced over time between higher level features we discover.

Ultimately could we have a single discretisation bucket for each fundamental feature and then split (and later possible join) discretisation granularity to model a problem and sequences are build and tested. CartPole might be a good test case.

Limitations

The major limitation on this work was time available. However, it does lay the ground for many other possible projects, which new now intend to pursue.

The project started with a relative simple premise. That representation and search could be improved. As we got more deeply into these problems and considered their application for physics based problems more and more directions for future research opened up, as the clock ticked away. Many of these direction appear to be unconventional, yet we cannot find obvious fault with them so they are documented here. This project has essentially turned into a launch vehicle for many follow on projects. The author would like to pursue these ideas and further weed out good and bad ideas. Ultimately we would like to see a standard methodology, like that developed for supervised learning, for RL too. We also envisage ‘standard’ solutions for problems, that may or may not stand the test of time. At the moment we are still in the Wild West.

Thought Experiments

Writing this paper triggered a number of though experiments, some of which were ultimately not productive, others may or may not lead to interesting results, but further work needs to be done. Most of these are represented below in the suggested future work. There are the seeds of a number of papers here, many of which will be failures. However, much can be leant from failure. Some will be controversial (hopefully). Reading over the above, there is no mention of MDPs or similar. In fact, much of what has been done ‘stretches’ the ideal of an MDP. This is to be expected, it’s a useful unifying concept and breaking or extending it requires explanation and more work is required here. The next step is outlining a prioritised plan for addressing future work.

It would seem many problems assume certain initial conditions, which might limit the exploration of problems and ultimately limit knowledge creation. Scientists must be more open to counterfactual thinking. Actively assuming different initial (but possible) conditions and following through the logical consequences should be encouraged. Reinforcement Learning is a vehicle to do this.

Better simulators for RL is mentioned below. This would work too for other areas such as supervised learning especially when data is skewed. If the data can be simulated and counterfactuals used to address skew, then resulting models (such as face recognition) might be less biased. Synthetic data is used, but tends to be superficially generated.

Future Work

- Further investigation of some of the relatively simple changes to the results above that had a large impact. Such as discretisation schemes, reward shaping and action momentum.
- With further understanding, it might be possible to declaratively tune learning runs based on problem space insights.
- Better simulators, that provide variants on the problem space, such as different gravitational constants, problem space ranges and boundaries, magnitude of actions (when forces in a physics setting). Meta-data on parameters (after all, as a human one can quickly tell that a force to the left, will create an acceleration to the left and vice versa). Indeed if it were known that a parameter were a position (or angular position), then the velocity and other such parameters can be derived. Also, we need more problems in a class, such as more physics based problems, to explore the space better.
- With better, parametrised simulators producing say 100 variants on a problem, we could split these into Test, Validation and Test sets. The result would be more robust and generic/generalisable results. Also, an incentive to keep it simple to reduce runtime.
- Implement simulator in Rust for 'Classic' problem in gym with better meta-data, parametrisation of observation space and actions. Then add some additional problems.
- Much more research, hardly touched on here because of time constraints, of learning higher order features of the problem space and using those declaratively, in context, and the use of counterfactuals and causal inference. Neural Nets have a place (for unstructured data), but certainly in a physics domain (which is by it's very nature is linear), instinctively should not be necessary and may obscure real understanding.
- Explainable solutions.
- Tactical use of biologically inspired search schemes, such as PSO. Probably appropriate in small sub-spaces of the larger problem space. Which begs the question, how do we partition the global problem space.
- Model based solutions, linking the sequence of states traversed at a certain discretisation level and using failed predictions to increase local discretisation granularity. Using constraints on physical space to predict paths between discrete states, and link states together to better predict appropriate action, but understanding the relationship between state variables and sequences of actions.
- Robust learning on single, parametrised problems. This is to avoid 'over-fitting' to a single generic problem and it's specific parametrisation.
- 'Transfer' learning from one physics based problem to another.
- Counterfactual reasoning. We can train on what is, if we can understand the degrees of freedom in a problem space we might be able to combine them in counterfactual ways, not yet seen for better and broader solutions. In a word, causal reasoning in a limited domain.
- A Kalman filter can also be used to suggest 'missing' forces (gravity, possibly from multiple sources), when the errors it predicts are systematically biased over time. This could be used to improve the model automatically and infer additional features in the environment (unaccounted for forces in this case).

References

Sutton, 2018: Sutton, Rich, Tile Coding Software -- Reference Manual, Version 3.0, 2018

Ecoffet, 2021: Ecoffet, Adrien and Huizinga, Joost and Lehman, Joel and Stanley, Kenneth O. and Clune, Jeff, First return, then explore, 2021

Badia, 2020: Adrià Puigdomènech Badia and Bilal Piot and Steven Kapturowski and Pablo Sprechmann and Alex Vitvitskyi and Daniel Guo and Charles Blundell, Agent57: Outperforming the Atari Human Benchmark, 2020

Hawkins, 2021: Jeff Hawkins, A Thousand Brains: a New Theory of Intelligence, 2021

Sinclair, 2019: Sinclair, Sean R. and Banerjee, Siddhartha and Yu, Christina Lee, Adaptive Discretization for Episodic Reinforcement Learning in Metric Spaces, 2019

Ecoffet, 2020: Adrien Ecoffet and Joost Huizinga and Joel Lehman and Kenneth O. Stanley and Jeff Clune, First return, then explore, 2020

Ghiassian, 2020: Sina Ghiassian and Banafsheh Rafiee and Yat Long Lo and Adam White, Improving Performance in Reinforcement Learning by Breaking Generalization in Neural Networks, 2020

Weng, 2020: Weng, Lilian, Exploration Strategies in Deep Reinforcement Learning, 2020, https://lilianweng.github.io/lil-log/2020/06/07/exploration-strategies-in-deep-reinforcement-learning.html

Mario, E Di, 2013: Mario, E Di, Talebpour, Z, A Comparison of PSO and Reinforcement Learning for Multi-Robot Obstacle Avoidance, 2013

Sun W, 2020: Weiwei Sun and Andrea Tagliasacchi and Boyang Deng and Sara Sabour and Soroosh Yazdani and Geoffrey Hinton and Kwang Moo Yi, Canonical Capsules: Unsupervised Capsules in Canonical Pose, 2020

Video Presentation

https://github.com/Intelligent-Net/rep_search_in_RL/
raw/main/presentation.mp4