INTERPRETABILITY & REINFORCEMENT LEARNING

Abram Schönfeldt
SCHABR004

ADVANCED TOPICS IN REINFORCEMENT LEARNING
supervised by Dr Shock

The University of Cape Town

First Semester 2020

Abstract

Reinforcement Learning (RL) is a powerful and general field of artificial intelligence that provides a means to learn optimal strategies, or *policies*, to achieve specified goals. Although RL models have surpassed both human and other machine-learning models' performance in a number of domains, the policies that they learn can seldom be understood by humans, especially non-experts. Explainable artificial intelligence is a field that focuses on developing AI models that can be interpreted by humans. Given AI's, and specifically RL's, potential and current use in contexts that impact humans (autonomous vehicles, advising policy, etc.), there is a growing need for the transparency and interpretability of these models. This project explores some of the current literature on interpretability in Reinforcement learning. This exploration takes the form of explaining and discussing different case studies of interpretable RL frameworks. It then, not completely tangentially, explores a case-study on how RL has been used to come up with policies to deal with epidemics. Given the current COVID-19 pandemic, there is renewed necessity for developing sophisticated but also transparent strategies for dealing with epidemics. Finally, 4 fundamental RL methods (REINFORCE, Actor-Critic, Deep-Q learning and Proximal Policy Optimisation) are explained and implemented as linked Colaboratory notebooks.

List of Abbreviations

Abbreviations are explained as they are introduced, but for the sake of convenience, a collection of the abbreviations employed are presented below:

A2C - Advantage Actor-Critic

AI - Artificial Intelligence

ANN - Artificial Neural Network

COVID-19 - Coronavirus Disease 2019

DQN - Deep Q Network

DNN - Deep Neural Network

DRL - Deep Reinforcement Learning

GUI - Graphical User Interface

HNS - Human Normalised scores

LSTM - Long Short Term Memory

NDPS - Neurally Directed Program Search

NMT - Neural Machine Translation

PIRL - Programmatically Interpretable Reinforcement Learning

POMDP - Partially Observable Markov Decision Process

PPO - Proximal Policy Optimisation

RL - Reinforcement Learning

RNN - Recurrent Neural Network

STG - Stochastic Temporal Grammar

TORCS - The Open Racing Car Simulator

t-SNE - t-distributed Stochastic Neighbour Embedding

1. Introduction

Reinforcement Learning is a computational approach to understanding and automating goal-directed learning and decision making [37]. These goals are typically defined in terms of maximising a numerical *reward*, obtained when a RL agent performs appropriate *actions* in an environment. The strategy that governs how an agent takes actions depending on the *state* of the environment is called the agent's *policy*. As an agent gains experience from interacting with its environment, it is able to refine its policy so that it performs the actions that result in greater reward more often. However, the agent needs to balance this exploitation of actions that it has previously found to be profitable in certain states with an exploration of trying new actions that may yet yield greater rewards in those states. This simple framework consisting of a goal in terms of a numerical reward, an environment and an agent with a set of rules for updating its policy has proved to be a powerful means of discovering novel and often superior strategies in a variety of contexts.

To date, reinforcement learning has achieved expert levels of mastery in both 'classical' strategy games such as Go [34], and video games such as Dota 2 [8], Minecraft [23] and Starcraft [42]. RL has also managed to do more than achieve impressive performance in specialised domains, recently surpassing human levels of mastery on a suite of 57 Atari games [4]. These achievements, particularly in complex video games, hint at the promise of RL for advising decision-making in real world contexts, and indeed, RL has already seen applications in real world contexts. From trying to ease traffic congestion [3], to autonomous drivining [18], contexts which seem distant in the current COVID-19 lockdown, to exploring epidemic mitigation strategies [29, 19], a context more familiar, the applications of RL are widespread. As RL, along with other already prevalent machine learning techniques, begins to have a bigger impact on our reality, the necessity for the transparency and interpretability of these frameworks becomes apparent. As we become accustomed to the idea that AI will make decisions for us in our daily lives [2] we need to be able to understand how these decisions were made.

Not to say that gaming isn't a real world context.

RL is also an interesting relationship with neuroscience and psychology (see chapters 14 and 15 of Sutton & Barto [37])

2. Interpretability in Reinforcement Learning

The interpretability of a reinforcement learning model has to do with understanding both the model's behaviour in specific contexts, and the model's general behaviour. It has to do with the model's transparency, and the manner in which the model presents itself. It also has to do with the level of expertise of the person seeking to interpret the model. To capture some of these many nuances of interpretability, we adopt Puiutta & Veith's definition of interpretability, presented in *Explainable Reinforcement Learning: A Survey* [30] ,

Adopting a definition might be considered lazy, however, I agree with Puiutta & Veith's emphasis on coming to a consensus on the nomenclature.

> "The ability of a model to not only extract or generate explanations for its decisions, but to present this information in a way that it is understandable by human (non-expert) users to, ultimately, enable them to predict the model's behaviour."

The emphasis in this definition on interpretability being something relevant to not only the reinforcement learning expert is something we will often return to in the course of this work. One hopes that in promoting a definition of interpretability that considers it as not only an interest of the expert, that future work in RL will try be more accessible to 'anyone', and similarly anyone will be more inclined to use RL as a framework to approach to decision making.

'Anyone', as opposed to everyone, since not everyone cares to take an interest in such things (the AI that influences one's life, current affairs, policy-making, etc.)

One of the central structures to this exploration of interpretability in RL is a 'taxonomy' which we will use to broadly classify the approaches to interpretability that we encounter. This taxonomy is influenced by both the XRL survey, as well as by earlier work by Du et al. [11]. This taxonomy classifies the interpretability of RL based on the scope of the methods:

This taxonomy is also applicable to machine learning in general.

- *Global* methods attempt to explain the general behaviour of the agent
- *Local* methods attempt to explain individual predictions of the agent

and based on when it is applied:

- *Intrinsic* methods are mechanisms built into the agent that generate insights as the agent interacts with its environment
- *Post-hoc* methods are applied to the agent after it has interacted with the environment and generate insights retrospectively.

The papers discussed in this exploration are fitted into this taxonomy as best as possible (see table 1), given that the reality of many of their approaches to interpretability is more complex than these broad classifications. I am greatly indebted to Puiutta & Veith's survey of Interpretable RL. Many of the papers explored here were discovered thanks to their survey. Compared to their survey, this work goes into greater detail of the mechanics of a smaller selection of interpretable RL frameworks, as well as spends more time discussing the success of the interpretability frameworks relative to the given definition of interpretability.

		Scope	
		Global	Local
Time	Intrinsic	PIRL [41]	Hierarchical Policies [33] Attention Augmented agents [27]
	Post-hoc	t-SNE DQN [47] Outbreak Policies [29]	Expected Consequences [43]

Table 1: Categorization of selected XRL methods based on the taxonomy described by Puiutta & Veith [30]

2.1. Readings

Graying the black box: Understanding DQNs (Zahavy, Zrihem, & Mannor, 2016) [47]

The Deep-Q network (DQN), as proposed in the 2015 paper *Human-level control through deep reinforcement learning* [26], introduced a stable method of training an agent that uses a deep neural network (DNN) to approximate its Q-value function. As opposed to using linear function approximators, which often rely on problem-specific state representations, deep neural networks offer a more general and nuanced approximator. Although DQNs provide a more general and powerful approach to RL, optimizing a deep neural network, as mentioned in this paper and as seen in my implementation in section 4.3, is a complicated process of exploring hyper-parameters, and architectures. Other than challenges in optimizing the DNN, DNNs are hard to interpret and are often termed 'black boxes'. This paper proposes a set of tools, most notably a t-distributed Stochastic Neighbour Embedding (t-SNE) of the activations of the last hidden layer in the DNN coloured by state features, to understand the state representations learnt by the DQN agent.

For more information on Deep-Q networks, see my explanation of DQNs in section 4.3

T-SNEs were proposed in the 2008 paper *Visualizing Data using t-SNE* [22] and can create either two or three dimensional representations of high-dimensional data. After converting high-dimensional data into a matrix of pairwise similarities, t-SNE provides a way of embedding this similarity data that both preserves local structure and reveals global structures such as clusters of similar points. By embedding a DQN agent's learnt representation of the state space into two or three dimensional space (viewable on a two dimensional monitor) and by colouring and filtering the embedded datapoints (states representations) based on features such as Q-value estimates, reward received, action taken, they were able to identify clusters of state representations with similar features and understand certain dynamics between these clusters.

Three Atari 2600 games were used as environments for the DQN agents. These games were:

Feel free to skip these Atari game synopses...

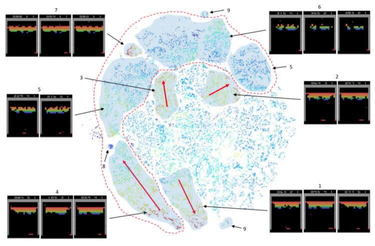
Breakout - your aim is to destroy rows of multicoloured bricks at the top of your screen. You control a platform at the bottom of the screen which you can move left and right in order to keep a bouncing ball in play that destroys the bricks when it comes into contact with them.

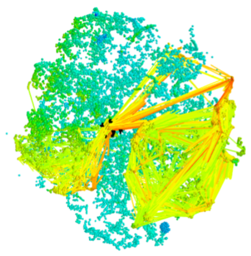
Sequest - you control a submarine which you use to dive underwater, collecting divers and shooting sea creatures which take away one of your finite lives each time they come into contact with you. You can only dive underwater for so long before you have to come up to replenish your oxygen. Your aim is to maximise the points you get from collecting divers and shooting sea creatures, while avoiding losing all your life.

Pacman - you control a yellow, circular, constantly munching creature that navigates a maze filled with ghosts. Your goal is to maximise your points by munching your way through every part of the maze and to avoid losing lives through contact with the ghosts (I've never questioned the premise until now).

... End of synopses.

Once they had trained a DQN agent in each of these environments, they had each agent interact with its environment again, this time recording at each state both game specific features, such as whether the agent had picked up a diver (Seaquest), or whether the agent had created a 'tunnel' through the bricks (Breakout), as well as general features such as the activations of the last hidden layer of DNN, information on the gradients of the DNN, and the raw pixel frame of that state. This information was used to augment the t-SNE embedding of the activations. By selecting individual points in the embedding, one can preview an image of the state, a saliency map of the state and the other extracted features of that state. Additionally, the whole embedding can be coloured based on the extracted features. In the paper, they consider colouring the embeddings based on Q-values, the amount of oxygen remaining (Seaquest), and the amount of the maze that had not been visited (Pacman). Finally, they produced some 3D t-SNEs that included the transitions between states drawn as arrows between the datapoints in the embeddings.


t-SNE with examples of pixel states in Breakout [47]


3D t-SNE [47]

Following this methodology, they were able to gain insights into both game-specific strategies learnt by the agents, as well as the general kinds of structures learnt in the DQN's state representations. Particularly, they were able to identify 'hierarchies' of clusters of state representations. These hierarchies in the embedding reflect a hierarchy to the policy. For instance, in Breakout, they identified that the agent learnt a hierarchical strategy that prioritised building a tunnel through the bricks, and then keeping the ball above the bricks for as long as possible. This hierarchical strategy was reflected in the t-SNE that showed distinct clusters of states associated with digging a tunnel, with transitions from these clusters to other distinct clusters associated with the ball bouncing around above the layers of bricks, having made its above through the tunnel.

I found the most meaningful parts of this paper to be the annotated figures of the t-SNEs, which were frequently referred back to in order to justify insights into the agent's policy. In this approach to analysing agent behaviour, the success of the provided explanations lies in the clarity of the patterns of the t-SNEs. However, in many cases I felt that the patterns in the t-SNEs were not as clear as made out by the authors, particularly in the case of the 3D t-SNE. Being able to interact with their collected data using the proposed graphical user interface (GUI) would make me more improve both my understanding and trust of their provided insights. This also touches on a broader theme of reproducibility of the results in this paper. Overall, this post-hoc technique to understanding the behaviour of DQN agents relies on the fidelity of the structure of the t-SNE to the actual structure of the agent's state representation. Implicit in this is also a reliance on understanding how t-SNEs work. Returning to Puiutta & Veith's definition of interpretability and its emphasis on presenting this information in a way understandable by non-experts, there are other shortcomings to this method. Unless intuition on the extent to which t-SNEs capture the structure of high dimensional data becomes wide-spread, this remains a technique to aid experts.

See the paper on attention augmented agents (Mott et al.)[27] for an example of a paper that provides satisfactory supplementary resources

In figure 1 of the paper[47], a GUI is proposed, but never mentioned again.

A nice explanation and implementation of t-SNE is open source in this Google experiment

Hierachical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning (Shu et al. 2017) [33]

We now briefly dive into the world of multi-task RL. In this setting, the agent solves complex tasks consisting of a series of sub-tasks each requiring different skills to solve. Guided by the observation that humans often take advantage of existing skills and by combining and composing them are able to come up with new skills, Shu et al. propose a hierarchical policy network that can reuse previously learnt skills alongside and as components of new skills [33]. This decomposition of a task into several simpler tasks is a central idea to this paper. In order to improve the agent's understanding of the appropriate sequence of actions and of prioritising certain tasks, a stochastic temporal grammar (STG) model was incorporated into their model. Similar to the policy sketches of Verma et al. which we discuss below [41] the STG improved the policy by explicitly modelling certain 'commonsense' approaches to tasks. The tasks carried out by this policy network are inherently human-readable because they make use of human instructions to encode them, e.g. "Find blue". This approach was implemented in a Minecraft game created using the Malmo platform

However, human-readable does not necessarily mean interpretable.

[16] and the tasks centered around finding, getting, placing and stacking blocks of 6 different colours. This environment was sparse in rewards, with the agent getting a $+1$ reward only when tasks were completed, and a -0.5 reward each time the agent generated an instruction that was not executable in the current environment. The resulting 'global policy' was successfully able to construct instructions that built on a hierarchy of learnt skills in order to accomplish tasks, albeit with weak supervision from humans. The performance of their full model was compared to that of an agent that used a flat-policy and their model was found to have a higher learning efficiency, as well as generalize better to unseen environments.

Unseen environment = A bigger minecraft room.

In order to explain the components of their policy network, we briefly have to explain some multi-task RL terminology. $\mathcal{G}$ is defined as a set of tasks $\{g_0, g_1, ..., g_k\}$ described by human instructions. Each of these instructions takes the form $\langle u_{skill}, u_{item} \rangle$ describing an item manipulation task e.g. "Get blue". Associated with each task is a Markov Decision Process consisting of states s and 'primitive' actions a . Rewards are dependent on both the state and the task $R(s, g)$.
Initially, a 'terminal policy' π_0 is in place, trained to do a set of basic tasks which define the 'terminal task set' $\mathcal{G}_0$. As the agent is trained to do more tasks, these task sets progressively grow in the manner $\mathcal{G}_0 \subset \mathcal{G}_1 \subset ... \subset \mathcal{G}_k$. At each stage k, the previous task set $\mathcal{G}_{k-1}$ is referred to as the *base task set* of the current task set $\mathcal{G}_k$. Similarly, the previous policy π_{k-1} is referred to as the *base policy* of the current policy π_k.

Primitive actions include move left, right, pick up, put down, etc.

I found 'terminal' misleading since I typically associate it with 'the end'

Their hierarchical policy network is constructed so that it can make use of the base policy to perform certain base tasks, but also elect to perform novel tasks. Explicitly, this hierarchy comprises four sub-policies:

1. A base policy π_{k-1} which executes learnt tasks
2. An instruction policy that communicates between the base policy and global policy
3. An augmented flat policy allowing the global policy to directly execute actions
4. A switch policy allowing the global policy to switch between the flat policy and base policy

A flat policy directly maps states and instruction to actions

At each time step, a binary variable $e_t \in \{0, 1\}$ is sampled from the switch policy to determine whether the global policy makes use of the base policy, or the augmented flat policy. In order to sample from the base policy, an instruction is first sampled from the instruction policy which is then fed as input to the base policy. Finally a STG interacts with the instruction and switch policy, using experience from past positive episodes to guide whether the the global policy should make use of the base policy, or augmented flat policy to take a primitive action.

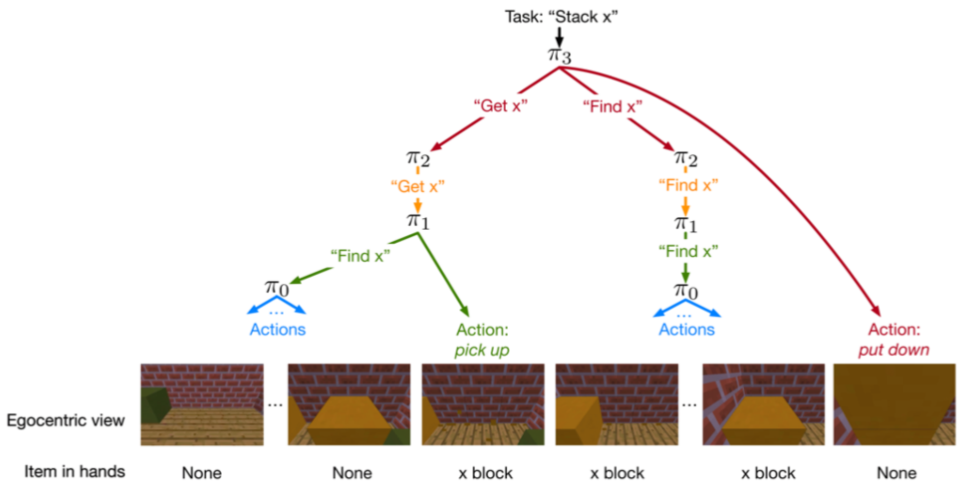


Figure 1: An example of the hierarchy of instructions generated by the global policy [33].

The hierarchical model was trained in stages of skill acquisition, each of which comprised a 2-phase learning curriculum. The first phase, termed the 'base skill acquisition phase', consists of the model purely sampling tasks from its base policy. The second phase, termed the 'novel skill acquisition phase', consists of the model sampling tasks from both the base policy and augmented flat policy. All policies were trained using advantage actor-critic (A2C) [35]. The model was then trained making use of RMSProp [38] with a learning rate of $\alpha = 1 \cdot 10^{-4}$, a discount rate $\gamma = 0.95$, a batch size of 36, and with gradient clipping to a unit norm. An ϵ-greedy policy was adopted at the top-level of the global policy. The model performance was compared to a flat policy, as well as variants of the global policy. They found that all variants of their hierarchical policy managed to converge to the (converted) maximum reward of 1 within 20 000 episodes, whereas the flat policy had not yet converged within this window, reaching a maximum reward of 0.8. Their hierarchical model was also able to generalize to a different context (a larger Minecraft room) better than the flat policy. In this larger environment, it was still able to accomplish the task of finding and stacking all pairs of coloured blocks 94% of the time, whereas the flat policy was only able to achieve this feat 29% of the time.

The rewards were converted to be in the range [0,1] for fair comparison

Overall, this paper presents an intrinsic, local approach to interpretability. The model's use of human-readable instructions make *what* the model does at each time step understandable, hence we gain insight into the local behaviour of the model. The fact that this interpretability mechanism is built into the model makes it intrinsic. An example of the hierarchy of instructions generated by the global policy, taken from the paper, is shown in figure 1. Here we can see how a complex task is decomposed into a series of learnt tasks, sampled from previous policies. Branches are ordered from left to right in time indicating the order of the instructions carried out by the global policy. This decomposition of a complex task is intuitive from a human perspective. However, although what the model does is apparent, why it does it is still hidden in the 'black box' of the neural networks driving several A2C policies. In this sense, it would become quickly apparent when the model performs a task in an undesired manner, but understanding why it displays undesirable behaviour remains a task for the 'expert'. Although this environment is significantly more complex than the grid-world example considered by van der Waa et al. (2.1.4), there is the same question of how to generalise the generation of the descriptions of policy tasks and actions. One of the big questions this case-study on interpretability raises is whether being able to name the actions that an agent carries out inherently makes it more understandable.

Programmatically Interpretable Reinforcement Learning (Verma et al. 2018) [41]

This paper proposes an intrinsic, global framework to interpretable RL. The framework, *Programmatically Interpretable Reinforcement Learning* (PIRL), uses a high-level, domain specific programming language to represent the agent policy. Other than making these policies readable by humans, the other benefit of representing them as a high-level programming language is that it helps make them verifiable by traditional symbolic methods [17] These policies attempt to mimic the behaviour of an 'oracular' DRL policy, and although the agents that use the PIRL framework do not achieve as high rewards as the DRL agents in the environments considered in this paper, the ability to interrogate whether the PIRL policies posses certain desired properties, and whether they avoid undesired properties, makes them worth consideration especially in 'safety-critical' applications. They additionally present evidence that PIRL agents might generalize better to other environments, in this case evaluating the performance of both the PIRL agent and the DRL agent on an unseen TORCS race course. In the video of this showdown, linked at the end of the paper, the PIRL agent demonstrates a series of smooth, rapid turns and easily completes the course, whereas the DRL makes a series of jerky movements that cause it to lose control, bump into a wall and fail the course.

The main environment considered was *The Open Racing Car Simulator* (TORCS), but three OpenAI classic control problems appear in the appendix, including CartPole.

In order to learn a high-level programmatic policy, the agent has to be supplied with a *policy sketch*, defining the syntax of the set of learnable policies. This syntactic restriction on the set of policies has the benefit of 'pruning' undesired policies, encoding how the agent generalizes its behaviour, and allows the use of symbolic program verification techniques on the policy. In order to understand how these sketches can be phrased, we have to explain some terminology, certainly foreign to this Sutton &

Barto bred RL amateur. Observations, actions, as well as integers and reals, generated during computation are referred to as *atoms*. The programmatic policy language consists of two kinds of *data*: atoms, and sequences of atoms. E is used to denote expressions that evaluate to atoms, and x to expressions that evaluate to histories, where a history is a sequence of alternating observations and actions (similar to a trajectory). In addition, the paper defines the following language constructs:

This paper frames RL as a Partially Observable Markov Decision Process (POMDP), hence the terms 'observations' and 'histories'

x, x_1, x_2 are variables

c ranges over a universe of numerical constants

What a phrase.

$\oplus$ is a basic operator

Think +, -, ...

peek(x, i) returns the observation at time i of history x

peek$(x, -1)$ is shorthand for the previous observation

fold is a *combinator* over a sequence with the semantics:

$$\textbf{fold}(f, [e_1, ..., e_k]) = f(e_k, f(e_{k-1}, ...f(e_1, e)))$$

The syntax of a PIRL policy can thus be defined as:

$$E \doteq c \,|\, x \,|\, \oplus (E_1, ..., E_k) \,|\, \textbf{peek}(x, i) \,|\, \textbf{fold}((\lambda x_1, x_2.E_1), \alpha)$$

and a policy sketch is formally obtained by restricting the grammar of this syntax.

The 'PIRL problem' can then be formally stated as: "Suppose we are given a POMDP M and a sketch S. Our goal is to find a program e^* out of the set of programs permitted by our sketch $[\![S]\!]$ with optimal reward:

$$e^* = \operatorname*{argmax}_{e \in [\![S]\!]} R(e)$$

Having restricted the set of programmatic policies by introducing a policy sketch, Verma et al. propose a framework inspired by imitation learning called *Neurally Directed Program Search* (NDPS) which uses a DRL policy as an oracle to find an approximation of the optimal PIRL policy e^*. The policy e' is the estimate for e^* that minimizes the distance between the policy and the deep neural network policy e_N, where the distance is defined as,

$$d(e_n, e) = \sum_{h \in \mathcal{H}} \|e(h) - e_N(h)\|$$

$\|\cdot\|$ is a norm and $\mathcal{H}$ is a set of histories

The performance of this framework was then evaluated by generating controllers for cars in TORCS, a "highly portable multi-platform car racing simulation" [46]. The paper trained multiple PIRL agents and a DRL agent on one of the courses in the 'practice mode' setting of TORCS. Practice mode restricts the input to 29 sensory inputs, and the actions to just controlling the acceleration and steering of the car. The PIRL agents differed in that one of them did not have access to a policy oracle, another did not have a sketch, etc. Ultimately, the DRL agent performed the best on the course, followed by the agent that used a policy generated by the proposed NDPS algorithm, but as we noted in the introduction to this paper, the PIRL agent outperforms the DRL in other ways. To put the interpretability of the PIRL policies to the test, the 'human-readable' program concerning choosing accelaration is presented below:

Apparently the full mode of TORCS with input from 89 sensors, clutch and gear control, and pit stops was a bit too challenging.

$$\begin{aligned}
&\mathbf{if}(0.001 - \mathbf{peek}(h_{\text{TrackPos}} - 1) > 0)\mathbf{and}(0.001 - \mathbf{peek}(h_{\text{TrackPos}} - 1) > 0)\\
&\quad\mathbf{then}\, 3.97 * \mathbf{peek}((0.44 - h_{\text{RPM}}), -1) + 0.001 * \mathbf{fold}(+, (0.44 - h_{\text{RPM}})) + (\mathbf{peek}((h_{\text{RPM}}), -2) - \mathbf{peek}((h_{\text{RPM}}), -1))\\
&\quad\mathbf{else}\, 3.97 * \mathbf{peek}((0.40 - h_{\text{RPM}}), -1) + 0.001 * \mathbf{fold}(+, (0.40 - h_{\text{RPM}})) + (\mathbf{peek}((h_{\text{RPM}}), -2) - \mathbf{peek}((h_{\text{RPM}}), -1))
\end{aligned} \tag{1}$$

Now, while this certainly constitutes readable, especially in comparison to staring at a matrix of weights associated with a deep neural network, whether it is interpretable is another matter. The non-trivial amount of effort required to understand the meaning of the language constructs used to formulate this policy, let alone make sense of the implications of the difference between the 0.44 and the 0.40 in $\mathbf{peek}((0.44 - h_{\text{RPM}}), -1)$ and $\mathbf{peek}((0.40 - h_{\text{RPM}}), -1)$ to acceleration, makes me question the interpretability of these programs in hands other than the 'expert'. Other shortcomings noted in the explainable RL survey are that only symbolic inputs and deterministic policies were considered. [30]

I found an introductory course in the theory of computation also necessary to grasp the broader concepts of grammars and programmatic languages

The next paper we consider makes progress in terms of considering the non-expert opinion in assessing the interpretability of their framework.

Contrastive explanations for reinforcement learning in terms of expected consequences (van der Waa et al. 2018) [43]

Contrastive explanations for reinforcement learning in terms of expected consequences differs from the papers by Zahavy et al. and Verma et al. in that it begins to take into consideration an external opinion on the interpretability of their proposed framework. Once they had developed a framework for explaining the actions of an RL agent in a model environment, they conducted a formal, but small user study with 82 participants over the internet to gain insights into which sorts of explanations were more satisfactory. They also take into consideration ideas proposed in Miller's paper, *Explanation in Artificial Intelligence: Insights from the Social Sciences*, which advocates, "drawing on the vast bodies of research in philosophy, psychology and cognitive science on how people define, select, evaluate, and present explanations" in developing explanations in artificial intelligence [24]. Specifically, van der Waa et al.

frame explanations for the behaviour of an RL agent by using *contrastive explanations*. Contrastive explanations involve explaining the cause of an event relative to another event that did not occur, or concisely, offers explanations of the form, "*Why P rather than Q?*", where P is the target event and Q is the counterfactual, contrast case that did not occur [24]. Van der Waa et al. adopt the terms *fact* to refer to the learned policy π_t, guiding the target actions, and *foil* to refer to a contrasting policy π_f of interest to someone (a 'user') seeking an explanation for why an agent performed in a certain way, perhaps contrary to an expected policy. Broadly, their methodology is to generate a fact policy, apparently (but not explicitly) using a form of Q-learning, and then create a foil policy by modifying the action-value function Q_t of the fact policy so that the resulting foil policy π_f uses a new action-value function Q_f that values actions in line with the user's foil(suggested contrastive behaviour) in the states the user is curious about, but otherwise behaves in line with the fact policy.

This slightly alters Lipton's definitions of fact as referring to the event that did occur versus foil as referring to the event that did not [20]

Their method consists of first translating the set of actions and states to a set of descriptive state classes $\mathbf{C}$ and action outcomes $\mathbf{O}$. This translation involves defining a function $\mathbf{k}$, consisting of a number of classifiers that maps the states S to the descriptive set $\mathbf{C}$, $\mathbf{k} : S \to \mathbf{C}$. The action outcomes are created by defining a function $\mathbf{t}$ that maps the Cartesian product between the set of state classes $\mathbf{C}$ and the set of actions A to the probabilities of the action outcomes $\mathbf{O}$, i.e. $\mathbf{t} : \mathbf{C} \times A \to Pr(\mathbf{O})$. In the paper, they provide no explicit methodology for finding the functions $\mathbf{k}$ and $\mathbf{t}$, instead referring to the approaches taken by Hayes et al. [14] and Sherstov & Stone [32]. Although, they do offer examples of state classes $c \in \mathbf{C}$ and action outcomes $o \in \mathbf{O}$ in their chosen context - a gridworld environment boasting a small forest, spikey traps and a monster - the lack of explicit methodology for finding $\mathbf{k}$ and $\mathbf{t}$ begs the the question whether this framework of interpretability is feasible for more complex RL benchmarks. They do note in their conclusions that their future work will be focused on implementing this framework on more complicated environments to explore the scalability of it, for both the reason of exploring more methods of constructing the translation functions and assessing the computational feasibility of simulating the additional foil policy.

If I am being vague in defining functions $\boldsymbol{k}$ and $\boldsymbol{t}$, it is because they were as well, which we pick up on later.

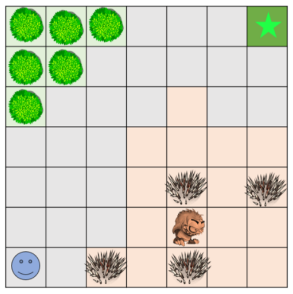


Illustration of their gridworld environment, taken from the paper [43]

Next, their method explains how to convert a contrastive question, consisting of a fact and foil, into a foil policy. They use the following example of a contrastive question,

> Why do you move up and then right (*fact*) instead of moving to the right until you hit a wall and then move up (*foil*)?

The foil policy is obtained by creating a modified action-value function Q_f combining an action-value function Q_I that alters the value of actions relating to the foil of the user's question, with the learned action-value function Q_t:

$$Q_f(s,a) = Q_t(s,a) + Q_I(s,a)$$

Following the same method of action selection as in the original policy π_t (e.g. $\epsilon - greedy$), but now selecting actions based new action-value Q_f, we obtain the foil policy π_f. The action-value function Q_I only modifies the value of Q_f for the states queried in the user's question. Continuing with the above exemplar question, one way we can define Q_I is based on the reward scheme:

The action a_f^1 = 'Right' receives reward s.t. $Q_f(s, Right) > Q_t(s, \pi_t(t))$

if 'RightWall' $\in \mathbf{k}(s)$

then the action a_f^2 = 'Up' receives reward s.t. $Q_f(\cdot, Right) > Q_t(\cdot, \pi_t(t))$

In order to update the action-values of Q_I, a modified reward needs to be defined that distinguishes between the influences of Q_I and Q_t. The rewards associated with Q_I are given as:

$$R_I(s_i, a_f) = \frac{\gamma_f}{\gamma} w(s_i, s_t)[R(s_i, a_f) - R(s_i, a_t)](1 + \epsilon)$$

where the ratio γ_f / γ compensates for the potential difference between the original discount factor γ and the discount factor associated with the foil value function γ_f, a_f are the actions associated with the foil policy and a_t are the originally learned actions, $w(s_i, s_t)$ reduces the rewards R_I as the distance between the original state s_t and s_i increases, and ϵ controls how much we reward the contribution of the foil policy. $w(s_i, s_t)$ is obtained from a radial basis function with a Gaussian kernel and a distance function d:

$$w(s_i, s_t) = \exp\left[-\left(\frac{d(s_i, s_t)}{\sigma}\right)^2\right]$$

By setting σ, we can define how far away from the state s_t we reward behaviour based on the foil action-value function. As the distance between the states chosen using the foil policy and the states that would have been chosen following the original policy becomes very large, the rewards R_I tend to zero and $Q_f \to Q_t$.

Paths were then generated following both the original policy π_t and the foil policy π_f where a path is defined as a translation of a trajectory of states and actions following a policy, to a trajectory of state classes and action outcomes:

$$Path(s_t, \pi) = \{(c_0, o_0), ..., (c_n, o_n)\},$$
$$c_i = \mathbf{k}(s_i), o_i = \mathbf{t}(a_i)$$

By using the difference between the paths, $Path(s_t, \pi_t)$ and $Path(s_f, \pi_f)$, contrastive explanations for the model behaviour can be explained in relation to a foil posed by a user. Van der Waa et al. present two ways of looking at the differences between the paths of the fact and foil policy - one which considers the relative compliment $Path(s_t, \pi_t) \setminus Path(s_f, \pi_f)$ between the two paths, and another which considers the symmetric difference $Path(s_t, \pi_t) \triangle Path(s_f, \pi_f)$ between the paths.

The relative compliment is set of all the 'consequences' (c, o) that are unique to the foil path, whereas the symmetric difference is the set of consequences unique to the fact and to the foil path.

Overall, this approach to interpretability involves explaining what the RL agent does in terms of expected states and outcomes, and why an agent chooses this behaviour by contrasting the expected outcomes of its policy with a foil policy, posed by the user. In order to explore which sorts of explanations are preferred by humans, a user study of 82 participants was conducted. The users were presented with explanations of the actions of the agent in the gridworld example, both in terms of its immediate actions (local explanations), and the actions of the agent as a whole (global explanations). These different explanations were obtained by varying σ in the weighting function $w(s_i, s_t)$, and the foil discount factor γ_f. The results of this study indicate that the participants preferred explanations that address policy behaviour and explanations that provide ample information. This paper does acknowledge the limitations of their simple gridworld environment where immediate actions are in most cases obvious to the participant. Unfortunately, the paper doesn't offer examples of the contrastive explanations that were constructed following its methodology and presented in this user study. There remain many questions surrounding the scalability, feasibility and the quality of the explanations of this framework. However, including the voices of non-experts, and considering existing research of what constitutes satisfactory explanations are important steps to coming up with truly interpretable artificial intelligence.

Attention

My interest in attention stems from wanting to understand how actions taken by an agent at a moment in time relates to how the agent perceives its context. The aim of considering attention mechanisms in an interpretability context, is the promise that they could allow one to do exactly this. There are also numerous other benefits to learning about attention mechanisms not just restricted to interpretability applications. So, to begin to explore these, we start by trying to understand attention out of the context it evolved - Natural Language Processing.

For an accessible introduction to attention, see Weng's blog post [44]

The seminal paper on attention, entitled *Neural Machine Translation by Jointly Learning to Align and Translate* ('the NMT paper') frames it in the context of *neural machine translation*, which aims to create a single, large neural network that translates input sequences (e.g. a phrase in English) into target phrases (e.g. a phrase in French)

[5]. The approach in the paper builds on previous encoder-decoder neural machine translation models [36, 10] but addresses the issue that these models face of representing input sequences of variable sizes as fixed sized 'context vectors.' The inclusion of an attention mechanism that can 'attend' to encoded representations of the full input sequence allows for translations that can take full input contexts into consideration, including dependencies between words that might fall outside the window of a fixed context vector.

The 2 main components of an encoder-decoder, the *encoder* and *decoder* are typically Recurrent Neural Networks, which generalise feed-forward neural networks to sequential inputs so that they can model long term dependencies between the current input, and previous inputs in the sequence. If we think of an RNN as a recurrent cell that takes in a vector input $\boldsymbol{x}_i$ at time i and transforms it based on a function $f_{\boldsymbol{w}}$ parameterised by a set of weights $\boldsymbol{w}$ into a 'cell state' or 'hidden state' $\boldsymbol{h}_i$, then modelling dependencies along a sequential input is achieved by passing the previous hidden state as an input to the next hidden state $\boldsymbol{h}_i = f_{\boldsymbol{w}(\boldsymbol{h}_{i-1},\boldsymbol{x}_i)}$. The output of the RNN is then some function of the hidden states. In order to make training these RNNs feasible, mechanisms to prevent the gradients from vanishing due to long term dependencies are put in place. For instance, using Long Short Term Memory (LSTM) units for the recurrent cells introduces gates that selectively allow the flow of information between hidden states. RNNs that make use of LSTM units are often just referred to as LSTMs.

Bidirectional RNNs additionally model dependencies with later inputs

See Colah's blogpost for an explanation of LSTM units and its variants.

Returning to the encoder-decoder model, the encoder takes the input sequence and transforms it into a sequence of hidden states. The context vector $\boldsymbol{c}$, is then some function of the set of hidden states $\boldsymbol{c} = q(\boldsymbol{h}_1, \boldsymbol{h}_2, ...\boldsymbol{h}_{T_x})$. The sequence to sequence (seq2seq) encoder-decoder model [36] mapped the hidden states to a fixed context vector, whereas the seminal paper on attention proposes creating a context vector of variable length where each element $\boldsymbol{c}_j$ of the context vector is a weighted sum of the hidden states created by the encoder, with 'alignment weights' α_{ij}, which intuitively measure how much the sequential input $\boldsymbol{x}_i$ aligns with the output $\boldsymbol{y}_j$.

$$\boldsymbol{c}_j = \sum_{i=1}^{T_x} \alpha_{ij} \boldsymbol{h}_i$$

T is the last time-step of the input sequence, and T_x denotes the last hidden state. To be more precise about defining the alignment weights, alignment weights are the soft-max outputs of the alignment scores $a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i)$ between the encoder hidden states $\boldsymbol{h}_i$ and the previous decoder hidden state $\boldsymbol{s}_j$:

$$\alpha_{ij} = \frac{\exp\left[a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i)\right]}{\sum_{k=1}^{T_x} \exp\left[a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_k)\right]}$$

The type of alignment score used in the NMT paper $a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i)$, is defined by a single layer, a feed-forward neural network that takes in a concatenation of the encoder and decoder hidden states as an input. A $\tanh$ activation function was used.

$$a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i) = \boldsymbol{V}_a^\top \tanh(\boldsymbol{W}_a[\boldsymbol{s}_{j-1}; \boldsymbol{h}_i])$$

where $\boldsymbol{V}_a$ is a vector and $\boldsymbol{W}_a$ is a matrix of weights associated with the alignment score. Based on different definitions of the alignment weights, and the alignment score function, we can introduce different attention mechanisms. This particular implementation is broadly 'soft' or 'global' attention since alignment weights are placed all over the hidden representation of the input space, and because of its definition of the alignment score function, 'additive' attention.

The distinction between global and local attention comes after the NMT paper [21].

Having defined context vectors which attend to the entire set of hidden states, we now explain how these contexts get used by the decoder to predict the outputs $\boldsymbol{y}_j$. As mentioned above, the decoder is also a RNN. Its hidden states $\boldsymbol{s}_j$ depend on the context vector $\boldsymbol{c}_j$, the previous hidden state $\boldsymbol{s}_{j-1}$ and the previous output $\boldsymbol{y}_{j-1}$, $\boldsymbol{s}_j = f(\boldsymbol{s}_{j-1}, \boldsymbol{y}_{j-1}, \boldsymbol{c}_j)$. The probability of predicting output $\boldsymbol{y}_j$ given the previously predicted outputs $\boldsymbol{y}_i$, $i = 1, j-1$ and the input $\boldsymbol{x}$ is given by a function $g(\boldsymbol{y}_{j-1}, \boldsymbol{s}_j, \boldsymbol{c}_j)$ which depends on the previous output, hidden state $\boldsymbol{s}_j$ and context vector. The hidden state captures the previously predicted outputs, and the context vector captures the pertinent parts of the input sequence. The function g predicting the output could be a full connected layer with nodes for each of the words in the output language's vocabulary.

In summary, attention evolved to allow decoders selective access to the full latent representation of a sequence, facilitating the understanding of long term dependencies between the important parts of a sequence. We can get a sense of how the hidden states in the decoder relate to the hidden states in the encoder by looking at the attention weights α_{ij}. These will give us a rough idea of how important each part of the input sequence is to each predicted output in the output sequence.

Attention Implementation for Neural Machine Translation

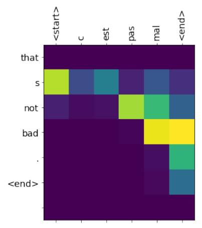
To deepen my understanding of how attention works, and specifically how additive attention works, I followed the Tensorflow tutorial on implementing a sequence to sequence model equipped with the same attention head as the NMT paper. The tutorial translates from Spanish to English. I adapted it to French to English translation - a setting for which I have access to expert quality control. My colab notebook is available here. Some of the phrases, and their (often absurd) translations, that I tested this 'toy' model on are shown on the next page, along with their attention matrices to the right.

Note that minimal punctuation was included in the training data, hence the lack of apostrophes.

1. Input: *C'est pas mal*

   Reference: It's not bad

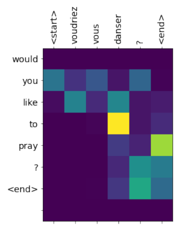
   Predicted: `that s not bad .`



2. Input: *Voudriez-vous danser?*

   Reference: Would you like to dance?

   Predicted: `would you like to pray ?`

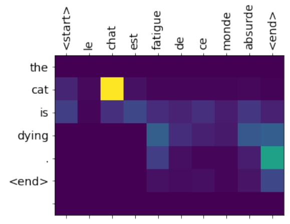


Two of my two favourite translations are listed below, touching on existentialism and focusing on the simple pleasures in life.

3. Input: *Le chat est fatiqué de ce monde absurde*

   Reference: The cat is tired of this absurd world

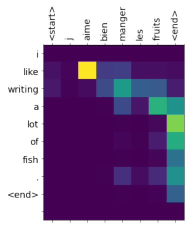
   Predicted: `the cat is dead .`



4. Input: *J'aime bien manger les fruits*

   Reference: I like eating fruit

   Predicted: `i like writing a lot of fish .`



Towards Interpretable Reinforcement Learning Using Attention Augmented Agents (Mott et al. 2019) [27]

This paper introduces an intrinsic, local interpretability mechanism by augmenting a RL agent with a soft, *top-down* attention mechanism that strongly influences the hidden state representation of the agent (in the authors' words, the attention mechanism creates a *bottleneck*). These attention augmented agents were trained and tested in 57 Atari 2600 games, provided by the Arcade Learning Environment [6], and their attention maps were analysed to explain how the agents behaved. The performance

This benchmark is dubbed *Atari57*

of the attention augmented agent in each environment was compared against two models that did not have attention bottlenecks - both based on deeper residual networks described by Espeholt et al. [12] - one which used the outputs of the ResNet directly to compute the policy π and value function V^{π} and one which inserted a LSTM on top of the RestNet. The attention augmented agent achieved higher median and mean *human normalised scores* (HNS) than both of these alternative models. The attention augmented agents were also also exposed to novel states in one of the games (Seaquest), created by adding a new enemy fish at the pixel level. This tested whether the agent had purely learnt predictable patterns in the Atari game, or whether it had indeed learnt to attend to and act upon the current state of the game. The agent performed favourably, attending to and reacting to the unfamiliar enemy fish. An interesting analysis into whether the queries directing the attention were driven by identifying *what* visual elements were, or *where* visual elements was conducted by creating visualisations of the attended regions of the game coloured by whether the *what* or *where* part of the query dominated. Finally, another approach to creating saliency maps [13] by introducing small, local Gaussian blurs to pixels in the image and measuring the magnitude of the change in the policy and in the value function was considered and applied to both the benchmark and the attention augmented agent. This *saliency analysis* facilitated two types of comparisons - the saliency maps of the benchmark and attention augmented agent were compared, and the saliency map and the attention map of the attention augmented agent were compared. Videos of these visualisations are linked in the paper.

For an interesting discussion on the credibility of HNS, see [39]

In order to understand the attention mechanism of this model, a rough sketch of the overall architecture is required. An observation $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$, consisting of an frame with height H and width W and channels C (RGB in this case) is processed by the *vision core* of the model which outputs a tensor $\mathbf{O}_{\text{vis}} \in \mathbb{R}^{h \times w \times c}$. This vision core comprises a convolutional neural network followed by a recurrent layer that is additionally fed the previous state of the vision core. The output of the vision core is split into a *key* tensor $\mathbf{K} \in \mathbb{R}^{h \times w \times c_K}$ and a *value* tensor $\mathbf{V} \in \mathbb{R}^{h \times w \times c_V}$, where $c = c_K + c_V$. The key and value tensors are concatenated with a fixed spatial basis $\mathbf{S} \in \mathbb{R}^{h \times w \times c_S}$ to encode locations (hence the channel dimension of the key and value tensor are $c_K + c_S$ and $c_V + c_S$ respectively). A Fourier basis type of representation was used for the spatial basis. The key tensor is then 'queried' by a *query network*. These 'queries' take the form of an inner product between query vectors and the key tensor. The query vectors $\boldsymbol{q}^n$ have sizes that match the channel-size $c_K + c_S$ of the key tensors, and the inner product of the n^{th} query with the key tensor forms the n^{th} attention logits map $\bar{\boldsymbol{A}}^n \in \mathbb{R}^{h \times w}$.

The function of the spatial basis is to preserve some of the spatial information in the channel c_S since the height and width dimensions are eventually collapsed in the attention 'bottleneck'

$$\bar{A}^n_{i,j} = \sum_l q^n_l K_{i,j,l}$$

The query network, which outputs N query vectors of size $c_K + c_S$, is a feed-forward neural network that receives the state of an LSTM from the previous time step as input. This LSTM represents the top of this top-down attention process and these attention queries are independent of the current state, instead guided by passed on information from the LSTM. We pick up from the logits $\bar{\boldsymbol{A}}^n$. A spatial soft-max (summing over i, j in the denominator) is then applied to obtain the final normalised attention maps $\boldsymbol{A}^n$. To then obtain the *answer* vectors $\boldsymbol{a}^n \in \mathbb{R}^{1\times 1\times (c_V + c_S)}$, we point-wise multiply the normalised attention maps with with the value tensor:

$$a_c^n = \sum_{i,j} A_{i,j}^n V_{i,j,c}$$

This weighted summation over the height and width of the key removes all the structural information - hence the motivation for the spatial basis which encodes some of the spatial information in the channels c. It is also referred to as the *bottleneck* since all the information about the current state captured by the value tensor, is weighted by the attention map. In this sense, the bottleneck forces the agent to focus on 'task-relevant' information. The bottleneck also forces the system to construct meaningful attention maps such that all state information is not 'blurred out' in the summation.

Finally, these N answer vectors, the N query vectors and the previous LSTM state are fed into the LSTM to produce the output $\boldsymbol{o}(t)$ of this time step, and the current LSTM state $\boldsymbol{s}_{\text{LSTM}}(t)$. Each of the query-answer processes is referred to as an *attention head*, thus we can talk about the N attention heads, and their respective attention maps at each time step. Since we use a soft-max function to create the attention map, the model is fully-differentiable, and trainable using back-propagation.

We now turn to discussing the insights obtained by the Mott et al. in their analysis of the attention maps. The recurring foci of the attention maps were the player controlled by the agent, the enemies, the power-ups and the score. This was seen as evidence that that the agents attended to 'task-relevant' elements in their environments. Some of the game-specific insights included observing paths of attention stemming from the player in maze-based environments such as Pacman and Alien, which were interpreted as the agent scanning potential paths for enemies. In projectile based games, such as Space Invaders and Breakout, the placement of *trip wires* was observed. These trip wires consisted of areas of attention that, when objects appeared in these areas (typically projectiles), triggered appropriate actions (such as quickly moving the platform to rebound the ball in breakout). These observations, facilitated by visualising the attention maps along with the game play as context, make it seem that the agent develops short-term strategies and reactions similar to human behaviour.

The workings of Sequest, Breakout and Pacman have been succinctly described in the write-up on t-SNEs of DQNs.

In my discussions of interpretability mechanisms thus far, I have questioned whether understanding local behaviour provides a satisfactory level of interpretability. In this study, the general behaviour of the agent was described in terms of typical patterns in the attention maps. What is interesting about these attention maps is that they are reflections of queries created by the agent based on its previous state, as opposed to saliency maps representing the importance of the current state in its decisions. These attention maps also reveal patterns in the agent's learnt policy. In this sense one cannot entirely dismiss this analysis as only explaining immediate behaviour. The emerging ways in which the agent queries its environment give us an insight into the agent's general behaviour. The user study by van der Waa et al. (2.1.4) suggested that humans are more interested in policy-level explanations. The example of a RL framework interpretable at the policy level that we have seen so far involved high-level programmatic policies (2.1.3). In order to interpret these policies, one had to first learn the meaning of the language constructs of the programmatic language, and then try to understand the logic of the resulting policy, which as seen in the example of the acceleration policy (eq. 1), can be intimidating. This programmatic policy does describe the dynamics of the agent in entirety and is human-readable, but it is still confusing, at least to a non-expert. This paper on attention augmented agents raises the question whether there can be a case made for understanding based on patterns of behaviour.

This user study included 82 participants, so is hardly the conclusive source on satisfactory explanations.

I am not too sure why I have settled on the term *non-expert*. Layman, amateur, and dabbler are all perfectly good words.

Context Matters: using reinforcement learning to develop human-readable, state-dependent outbreak response policies (Probert et al. 2019) [29]

This paper, published by the Royal Society, appears in an issue with the theme *Modelling infectious disease outbreaks in humans, animals and plants: epidemic forecasting and control*. This particular paper represents an intersection between RL, the interpretability of machine learning and epidemic modelling. It emphasizes the importance of taking context into account in policies that aim to prevent the spread of diseases. A typical approach to determining the effectiveness of different disease control policies is to run many stochastic simulations and asses the policies by how well they perform on average. However, the policy that performs best on average may not be the optimal policy for a real outbreak, which has only one realization that may well deviate from expected behaviour.

This paper goes on to highlight three limiting factors with delivering practical state-dependent policies. Computational challenges, which increase as larger state spaces are considered, challenges in interpreting and communicating policies, and challenges in implementation. Through two case studies, two illustrations of human-readable policies are provided. Both case studies focus on control methods for preventing the spread of disease in livestock. This scenario is based on the *foot and mouth disease (FMD)* outbreak in the UK in 2001. A stochastic, individual based (each individual represents a farm) model of FMD spread was used to simulate outbreaks.

The state of the outbreak at each time step was captured in an image of all the farms from above, with the shade of the farm (unshaded = susceptible, shaded-in = infected) indicating the state of the individual farm.

The first case study demonstrates the use of a deep-Q network (DQN) with a convolutional neural network to approximate its action-value function. At each time step (which seems to be each day), the model could choose to cull one farm. The reward function is shown to the right. There are penalties for excessive culling and for the outbreak resurging as a result of neglecting to cull infected farms, and there is a reward for the number of remaining cattle at the end of management. The first case study considers 3 scenarios:

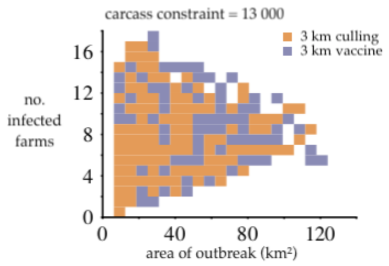
$$\begin{aligned} R = &-100(\text{no. farms culled}) \\ &-500(\text{no. resurgences}) \\ &+(\text{no. remaining cattle}) \end{aligned}$$

1. 30 farms of 25-500 animals on a 10×10 km grid with 6 initially infected farms
2. 120 farms on a 15×15 km grid with 10 initially infected farms
3. 30 farms in a clustered spatial distribution with an initial infection in a single cluster

In the first two scenarios, the reward trajectories did not converge, indicating that the DQN failed to converge on an optimal policy. In the third scenario, the rewards plateaued. This RL approach to outbreak control performed at least on par with the other control strategies in all scenarios, performing better on average than the other strategies in the first and third scenario. Using post-hoc (or '*a posteriori*') statistical analysis of the resultant policies for each scenario, they found that the farms that tended to be culled were the larger and closer ones to the initially infected farms. The last scenario, which had the initial infection in a single cluster, had the property that the infection had to move through 'bridging farms' in order to move to another cluster. In this scenario, the resultant policy learned to cull the bridging farms to prevent further outbreak.

The other, static control strategies considered were *random culling*, *culling known infectious farms* and *ring culling*

The second case study used epsilon-soft Monte Carlo control to construct policies to minimize the duration of an outbreak. 4001 farms of 500 livestock each were randomly placed in a landscape, and an initial outbreak was allowed to spread for 12 days. After 7 days of culling infected farms, a single action to either implement ring culling, or ring vaccinations of farms within 3 km of confirmed infected farms was made. The state was summarised in two dimensions - the number of farms infected, and the spatial extent of the outbreak. This means that the resulting policies could be visualised in two dimensions (see right). The implications of constraining the total number of carcasses that could be culled in a day, and the chosen control strategy were considered. For 'stringent'($\leq$ 12000) carcass constraints, ring vaccination was preferred, for 'ample resources' (carcass constraint $\geq$ 18000) , ring culling



was preferred, and for intermediate carcass constraints, a combination of vaccination and culling was preferred. On average, over all carcass constraints the RL control strategies resulted in shorter outbreaks than static strategies of purely ring culling or ring vaccinating. However, the performance of the RL policy was dependent on the starting conditions. When tested on different starting conditions, it only performed on par with the static strategies.

I would imagine that particularly for intermediate carcass constraints, the more nuanced RL policies outperform 'static' policies

These two case studies illustrate two ways of delivering human readable policies. The first case study generated a full policy, and then used statistical methods to reduce the policy to an understandable dimension. The second simplified the state space to an understandable dimension prior to searching for a policy. Both ways have their shortcomings. Summarising a full policy using statistical methods means ignoring some of the nuances of the full policy. Generating a full policy is also computationally intensive. The second scenario of the first case study allegedly took a couple of weeks to simulate. Simplifying the state space could mean a simplistic description of reality. The selection of features for the summary states requires careful consideration. Also, the policies generated based on these summarised states might not be optimal because they themselves are over-simplified. The paper notes that further research is required into "the limiting complexity of a policy; for example, what is the best low-dimensional representation of the state space, or what is the upper limit of complexity of the state space."

3. *Conclusions*

In this work a number of approaches to making RL more interpretable were explored. Although no single approach to interpretability emerged as perfect, each approach offered unique insights into the workings of RL agents, and into what interpretability is. There is potential to draw on aspects of each of these approaches and use them to design ever more transparent frameworks. Particularly, the post-hoc methods discussed can be used in conjunction with intrinsic interpretability mechanisms to provide an additional layer of understanding. In general, there is promise to building a collection of interpretability mechanisms, each with particular advantages and drawing on the appropriate ones as context requires.

As noted by Puitta & Veith, one of the emerging patterns of these frameworks is a trade-off between performance and transparency. The models with intrinsic interpretability mechanisms (PIRL [41], hierachical policies [33] and attention augmented agents [27]) which offered some of the more satisfactory explanations of policy behaviour all had bottlenecks in various forms* that restricted the performance of the

*a policy sketch, reliance on human instructions, an attention bottleneck agents. However, in all these cases, despite this trade-off, the agents implemented using these frameworks managed to demonstrate competitive performance, in some cases even outperforming their competition in other ways.

The PIRL agent was found to generalize better, and the attention augmented agent allegedly outperformed its competition.

Moving forward, I hope to see more transparency within interpretability papers. The papers I enjoyed reading and discussing were the ones that made their methods and results accessible - often providing additional materials in the form of explicit calculations, or raw footage of their agents in action. Along with seeing increasing transparency, I hope to see more inclusivity, or rather 'hear' more non-expert voices considered in the evaluations of these interpretability frameworks. Interpretability is not only an examination of how RL methods work, but also an examination of how we come up with explanations. The call to considering intersections between RL and other fields such as philosophy, psychology and neuroscience is worthwhile picking up [24, 37].

'Meta-transparency'

A nice paradox.

As a last word, if you, as the reader of this work, have gained very little, know at least that I as the writer have gained very much. This work reflects my expanding grasp on the field of RL, and my own attention mechanism, weighting recent developments in interpretability with great interest. I have gained confidence in implementing machine learning techniques in Python and TensorFlow [1, 28], in typesetting in LaTeX and channeling the teachings of Edward Tufte [40] , and in my own writing abilities. It has been a challenge to structure an eclectic work on this scale. However, it has been a challenge full of reward.

(in adding snarky margin notes)

Some future work - itemized:

Or *a final note to self.*

- Review the many papers discovered during of the course of this project that did not make an appearance.
- Implement fundamental algorithms on more complex environments.
- Explore more efficient means of searching through 'hyper-parameter space'. *Genetic algorithms* perhaps?
- Keep trying to interpret the endlessly evolving field of Interpretability.

4. *Implementing Fundamental Algorithms*

We implement four 'fundamental' algorithms in the context of the classic pole balancing problem called *cartpole*. In this problem, a cart that can move back and forwards on a track attempts to keep a pole balanced without coming to the ends of the track. At each time step, our agent chooses to apply a force to the cart to the left, or to the right, based on information it gets regarding the position and velocity of the cart, and the angle and velocity of the pole. For more information concerning the environment, see the OpenAI documentation [9].

4.1. *REINFORCE*

Explanation

REINFORCE is a policy gradient method that learns a parameterised policy $\pi(a|s,\boldsymbol{\theta})$ without the use of a value function. It is also called *Monte Carlo policy gradient*, alluding to its use of just experience to learn a policy. In order to explain how this direct learning approach links to our broad goal of maximising a reward, we have to briefly discuss the Policy Gradient theorem. Let us define our goal to be maximising a scalar performance measure, which adapting the notation in Sutton and Barto [37], we will denote $J(\boldsymbol{\theta})$. Maximising this performance measure will typical involve a form of gradient ascent, making incremental updates to the policy parameters $\boldsymbol{\theta}$ in the direction of the steepest ascent until we converge upon a local maximum. However, the connection between updating the policy parameters and maximising the performance metric is not trivial.

In the case where our agent interacts with an environment that is episodic, we define performance as the value of following our policy given the state that we start in. The difficulty with maximising this performance metric is that it depends on the unknown effects of policy changes to the state distribution. Fortunately, we can make use of the Policy Gradient Theorem to express the gradient of our performance metric with respect to our policy parameters without involving changes in state distribution[37]. The Policy Gradient theorem states that the gradient of our performance metric is proportional to the average sum over the actions of the gradient of our policy, weighted by the state-action value of each action:

$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}(s_0)}$

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[\sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \right] \tag{2}$$

The Policy Gradient theorem connects how updates to our policy, parameterized by

$\boldsymbol{\theta}$, based on the gradient of the policy improves our performance metric J. In the *all-actions method*, which follows directly from the Policy Gradient theorem, updates to the policy parameters take the form,

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t, a, \boldsymbol{w}) \nabla \pi(a|S_t, \boldsymbol{\theta}_t)$$

However, the policy gradient method we focus our attention on is the REINFORCE method which, as opposed to making updates based on the values of all actions at each state, only considers the actual action A_t taken at each time step [45]. For this method, our updates are of the form

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \tag{3}$$

We explain a simple implementation of the REINFORCE algorithm using action preferences $h(s, a, \boldsymbol{\theta})$ that are a linear combination of the features of each state. Although Sutton and Barto explain several ways of constructing features, a simple linear combinations sufficed to solve the cartpole problem.

Solving cartpole (v0) involves getting an average reward of 195 over 100 consecutive trials.

If we use h_t to denote the action preference (or 'logit') of taking the t^{th} action (out of a total of T actions), and if we consider an environment with N features for each state, then our action preferences are given by

$$h_t = \theta_{t,1} \cdot x_1 + \theta_{t,2} \cdot x_2 + ... + \theta_{t,N} \cdot x_N \tag{4}$$

We then apply the soft-max function

$$S(h_t) = \frac{e^{h_t}}{\sum_{k=1}^{T} e^{h_k}}$$

to the action preferences, so that we obtain the relative probabilities of taking each action in each state. This defines our policy.

In order to update our weights $\boldsymbol{\theta}$, following the rule defined in equation 3, we need to compute the so-called *eligibility vector*

$$\frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} = \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}_t)$$

This involves taking the partial derivatives of the policy with respect to $\boldsymbol{\theta}$. Since our policy can be thought of as a composition of functions, applying the soft-max function to linear action preferences, we need to use the chain rule. We first need to find the derivative of the soft-max function with respect to the action preferences, and then the derivative of the action preferences with respect to policy parameters $\boldsymbol{\theta}$.

We follow along with Eli Bendersky's blog post [7] on taking derivatives of the soft-max function. The partial derivative of the i^{th} component the soft-max function (the soft-max function has an output for each of the N actions) with respect to the action preference h_j is given by

$$\frac{\partial S_i}{\partial h_j} = \frac{\partial}{\partial h_j} \frac{e^{h_i}}{\sum_{k=1}^{T} e^{h_k}}$$

Since the soft-max function is of the form $f(x) = g(x)/h(x)$, we need to use the quotient rule $f' = \frac{g'h - h'g}{h^2}$. In the case of the soft-max function, this translates to

$$\frac{\partial S_i}{\partial h_j} = \frac{\frac{\partial e^{h_i}}{\partial h_j} \cdot \sum_k e^{h_k} - \frac{\partial \sum_k e^{h_k}}{\partial h_j} \cdot e^{h_i}}{[\sum_k e^{h_k}]^2}$$

When $i \neq j$, $\frac{\partial}{\partial h_j} e^{h_i} = 0$. Also, $\frac{\partial}{\partial h_j} \sum_k e^{h_k} = e^{h_j}$. Thus,

$$\frac{\partial S_i}{\partial h_j} = \frac{-e^{h_j} e^{h_i}}{[\sum_k e^{h_k}]^2} = -S_j S_i$$

When $i = j$,

$$\frac{\partial S_i}{\partial h_j} = \frac{e^{h_i} \sum_k e^{h_k} - e^{h_j} e^{h_i}}{[\sum_k e^{h_k}]^2} = \frac{e^{h_i}}{\sum_k e^{h_k}} \cdot \frac{\sum_k e^{h_k} - e^{h_j}}{\sum_k e^{h_k}} = S_i(1 - S_j)$$

We can combine these cases with the help of the Kronecker delta function into

$$\frac{\partial S_i}{\partial h_j} = S_i[\delta_{ij} - S_j]$$

Next, we need to take derivatives of the action preferences $\boldsymbol{h}$ with respect to the parameters $\boldsymbol{\theta}$. Each action preference is a linear combination of the features of each state (see equation 4), and each parameter has 2 indices $\theta_{t,n}$ - 1 relating to the action t it is associated with, and 1 relating to the feature n it is associated with.

If we take the derivative of some action preference h_j with respect to parameter $\theta_{t,n}$, if $t = j$

$$\frac{\partial h_j}{\partial \theta_{t,n}} = \frac{\partial \theta_{t,1} \cdot x_1 + ... + \theta_{t,N} \cdot x_N}{\partial \theta_t, n} = x_n$$

and if $t \neq j$, then $\partial h_j / \partial \theta_{t,n} = 0$.

Overall, taking the partial derivative of the one of the T outputs of the soft-max function with respect to one of the $T \times N$ policy parameters can represented as

$$\frac{\partial S_i}{\partial \theta_{t,n}} = S_i(\delta_{it} - S_t)x_n$$

Thus, our updates take the explicit form:

$$\theta'_{t,n} = \theta_{t,n} + \alpha G_t \frac{S_i(\delta_{it} - S_t)x_n}{S_i} = \theta_{t,n} + \alpha G_t(\delta_{it} - S_t)x_n$$

where S_i is the component of the output of the soft-max function corresponding to the action chosen at time t', and G_t is the complete return from the current time step t' defined by $G_t = \sum_{k=t'+1}^{T} \gamma^{k-t'-1} R_k$.

In future, I will reserve t for time, not for the index of action t, which was retrospectively a bad decision

Implementation

Here is a link to the Colaboratory notebook of my implementation of REINFORCE applied to the cart-pole problem.

Results

A range of values for α, the step-size parameter, and γ, the discount rate parameter were explored. For each value of α and γ, the policy parameters $\boldsymbol{\theta}$ were initialized at random, but using the same seed. The agent was then put through 30 repetitions of 500 episodes, to obtain average performances for each of the 500 episodes. The average performance over the 500 episodes of the agent with varying values of α, with $\gamma = 1$ can be seen in figure 4.1. Values for α range from 1 to 0.0001, being divided by 10 each time. The value of α effects how much of an impact experience from the last episode has on the updates of the policy parameters. Small values of α mean very gradual adjustments to the policy parameters, and very large values of α mean very large adjustments. Large values of α cause the agent to learn faster, but they also cause larger fluctuations in the performance of the agent. Small values of α have lower fluctuations in performance. From figure 4.1, the best trade-off between learning fast enough, and fluctuations in performance seems to be at around $\alpha = 0.001$.

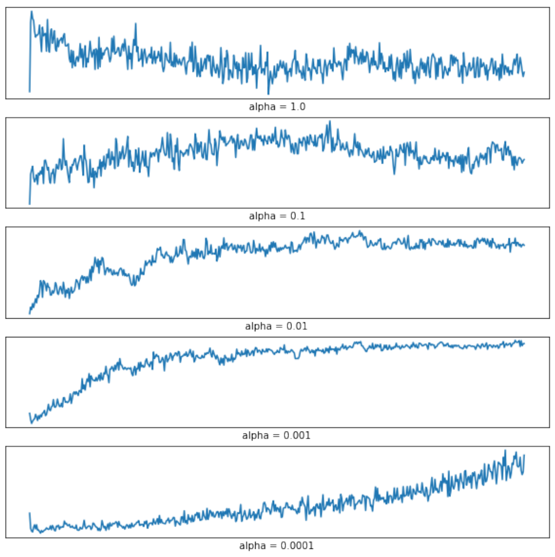


Figure 2: Varying α, $\gamma = 1$

Reward trajectories over the 500 episodes. A learning rate of $\alpha = 1 \times 10^{-3}$ was found to be the most stable.

The average performance of the agent over 30 repetitions of 500 episodes was repeated with different discount rates (γ). γ was varied from 1 to 0.6, decrementing by 0.1 each time. The results of this can be seen in figure 3. The discount rate influences how much the agent takes future rewards into account in its present actions. Choosing a discount rate much less than 1 (in general, $0 \leq \gamma \leq 1$) would mean that the agent focuses more on doing actions that maximise present rewards, and choosing a discount rate close to 1 would mean that the agent takes perceived rewards far in the future into account in its present actions.

In the cartpole context, the goal is to balance a pole for as long as possible, which provides some intuition for why we see values of γ closer to one achieving better performance

Intuitively, a discount rate much less than 1 is not suited to this context since the agent would be content with states that, although immediately are 'balanced', will soon become unbalanced. A discount rate close to, if not equal to 1 is a lot more desirable for this problem, where remaining balanced for as long as possible is the goal.

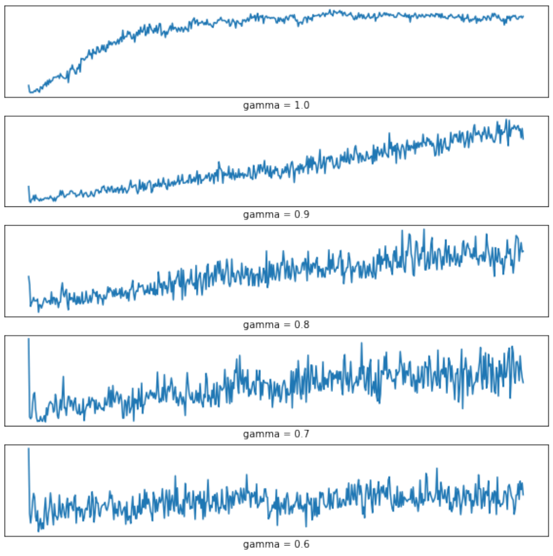


Figure 3: Varying γ, $\alpha = 0.001$

Reward trajectories over the 500 episodes. The undiscounted case $\gamma = 1$ was found to be the most stable.

This intuition is mirrored in the average performance we observe for different values of the discount rate γ. As γ is decremented, the average reward for each episode becomes more and more erratic. The overall performances also worsens.

There is still room to perform more fine grain experiments into the step-size and discount rates, as well as into the relationship between the two parameters, however, these exploratory experiments will suffice for now.

REINFORCE resources

1. The blog post which inspired this simple approach to implementing REINFORCE

4.2. *Actor-Critic*

Explanation

Actor-critic methods also fall under the category of policy gradient methods. However, in addition to learning a policy, actor-critic methods learn a state-value function and make use of 'bootstrapping' in the updates to the policy and value function parameters. In order to understand the progression from REINFORCE to actor-critic methods, we briefly discuss a generalization to the Policy Gradient theorem. In order to encode the idea of what a good action to take is relative to a state dependent 'baseline', we can include a term $b(s)$ in the equation for the Policy Gradient theorem:

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi}\left[\sum_{a}(q_{\pi}(s,a) - b(s))\nabla\pi(a|s,\boldsymbol{\theta})\right]$$

This baseline can be defined as the state-value function $v(s)$ for instance. We now see that the gradient of our performance metric is proportional to the difference between the values of actions in a state, and an overall value of that state. There is an intuitive appeal to this generalization. Provided that the baseline is not dependent on the action taken at each state, its inclusion in the Policy Gradient theorem leaves the expectation unchanged, but has potentially beneficial impacts on the variance of the expectation. Based on this, we can define the update rule for REINFORCE with baseline:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha(G_t - b(S_t))\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

Now, although REINFORCE with baseline is a method that learns both a policy and, potentially, a state-value function, we do not consider it an actor-critic method. Updates to its policy and state-value function typically happen at the end of episodes and draw on a collection of experiences accumulated during the episode. Making updates based on raw experience, as opposed to estimates, is part of the reason why REINFORCE with baseline lacks the ingredient of 'boostrapping' that defines actor-critic methods. Thus REINFORCE with baseline both also falls into the category of a Monte Carlo method.

Bootstrapping is a concept that is introduced along with Dynamic Programming and Temporal Difference methods and involves making updates to estimates partly based on other learnt estimates (See Chapter 6 of Sutton and Barto [37]). One of the simplest illustrations of the difference between an update rule that relies primarily on experience and one that bootstraps, is the difference between the state-value updates of purely Monte Carlo Methods and those of Temporal difference methods. The state-value update of Monte Carlo Methods take the form,

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Compare this to the 1-step Temporal Difference update,

Also denoted *TD(0)*

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t-1} + \gamma V(S_{t+1}) - V(S_t)]$$

which replaces the return G_t, which can only be determined following the completion of the episode, with an estimated return $R_{t+1} + \gamma V(S_{t+1})$, which is available immediately after taking action A_t and finding out the following state S_{t+1}. Through bootstrapping, we introduce bias, as well as reliance on on how we represent states. As with the introduction of a baseline, this bias can help reduce the variance in expectation relating to our performance metric.

Having had success with a very simple implementation of the REINFORCE algorithm, a simple implementation of an actor-critic method was opted for. Again, the policy was parameterised by a linear combination of a representation of the features of each state, forming action preferences that were then put into a soft max function to obtain probabilities of taking each action.

However, the way we represented each state did change. This time interactions between the features of each state were included. For instance, for the cartpole problem, where 4 numbers represent the position (x_1) and velocity (x_2) of the cart, and the angle (x_3) and velocity (x_4) of the pole at each state, our representation consisted of all combinations of size $n = 1, 2, ..4$ of the features of each state:

$$\boldsymbol{x}(s) = (1, x_1, x_2, ..., x_4,\ x_1x_2, x_1x_3, ..., x_3x_4,\ x_1x_2x_3, x_1x_2x_4, ..., x_2x_3x_4,\ x_1x_2x_4x_4)$$

Where each of the features x_i changes based on the state, but this dependence is left implicit for the sake of brevity. Other state representations are worth looking into and comparing.

The state-value function was defined as the dot product between our state representation vector and a vector of weights $\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{x}(s)$. Overall, a 1-step Actor-Critic algorithm was implemented. Defining the the Temporal-Difference error:

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$$

Using the above definition, we define the updates to the policy parameters as:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta}}\, \gamma^t\, \delta_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

For details on how to compute the eligibility vector (including what the eligibility vector is) refer to REINFORCE above.

for discrete time steps $t = 0, 1, ..., n$, and where $\alpha_{\boldsymbol{\theta}}$ denotes the learning rate associated with the policy.

The updates to the state-value weights are defined as:

$$\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha_{\boldsymbol{w}}\, \delta_t \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

where $\alpha_{\boldsymbol{\theta}}$ denotes the learning rate associated with state-value function. The gradient of the state-value function with respect to the weights is straightforward given that the state-value function is a dot product between the state representation vector and the weight vector $\nabla \hat{v}(S_t, \boldsymbol{w}_t) = \boldsymbol{x}(S_t)$.

Implementation

Here is a link to the Colaboratory notebook of my 1-step Actor-Critic implementation applied to the cart-pole problem.

Results

Similarly to the REINFORCE implementation, a number of combinations of hyper-parameters were explored for this implementation of actor-critic. Previously, values of α were explored with γ being held constant and vice versa. The undiscounted case $\gamma = 1$, had the most stable reward trajectory, and a value of $\alpha = 0.001$ seemed to be the best compromise between learning speed and stability. This time, we assume the undiscounted case $\gamma = 1$, and consider different combinations of the learning rates associated with the policy $\alpha_{\boldsymbol{\theta}}$, and the state-value function $\alpha_{\boldsymbol{w}}$.

The learning rates each took on a value in the set $\{0.1, 0.01, 0.001, 0.0001, 0.00001\}$. Every combination of the two learning rates were simulated for 500 episodes, with the rewards being recorded for each of the episodes. This was repeated 30 times to obtain average rewards at each stage of the simulations. Each simulation always started with the same set of initialised parameters and weights. The average reward 'trajectories' over the 30 realizations of 500 episodes for each pair of learning rates is shown in figure 4. In comparison to the average reward trajectories we saw for the different learning rates in the REINFORCE implementation, these trajectories, even when averaged, have a lot of variation, and the overall shape of the trajectories associated with small learning rates (e.g. $\alpha = 1e-5$) cannot be determined in the window of 500 episodes. On the other hand, we find for that many combinations of large learning rates (e.g.$\alpha_{\boldsymbol{\theta}} = 0.01, \alpha_{\boldsymbol{w}} = 0.01$) approach plateaus within the 500 episode window, but well below the maximum reward of 199 (in this case, the last reward was not included). Other combinations of large learning rates ((e.g.$\alpha_{\boldsymbol{\theta}} = 0.01, \alpha_{\boldsymbol{w}} = 0.01$)) rapidly decay to almost intentional attempts at destabilizing the pole.

In many of these case, one would wish that these trajectories would evolve backwards

Whereas previously, this format of exploring learning rates helped identify that a learning rate of around $\alpha = 0.001$ resulted in the most steady reward trajectory for the REINFORCE implementation, this time the exploration does not show any

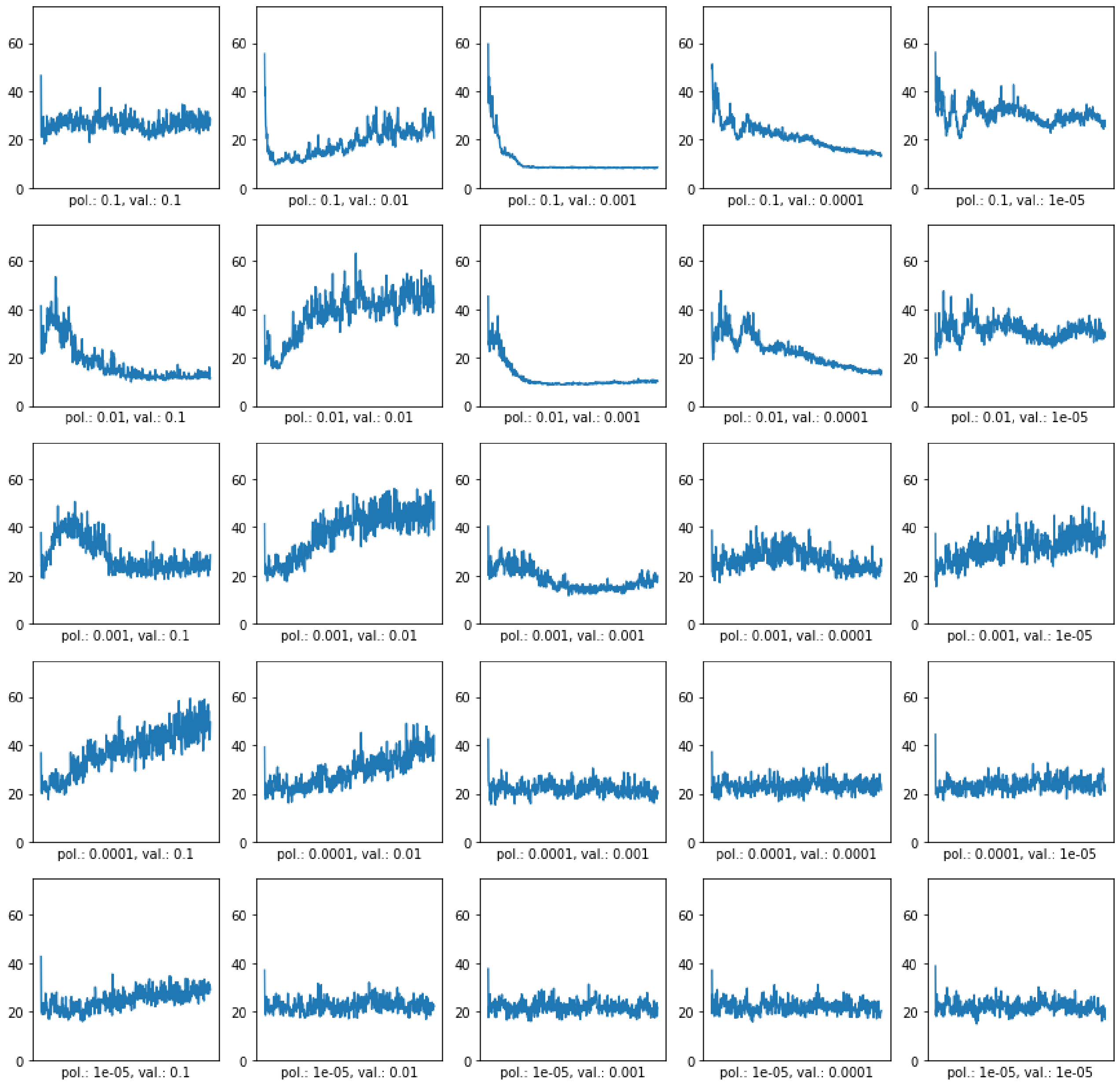


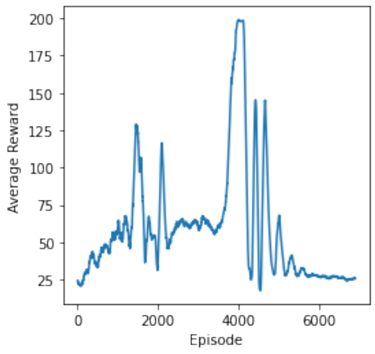
Figure 4: Exploring the average reward trajectories of different combinations of the policy learning rate (pol.), and the state-value function learning rate (val.). The x-axis is the number of episodes the agent has interacted with the environment for, and the y-axis represents the average reward obtained by the agent at that episode, where each average is obtained from the rewards of 30 episodes taken from different simulations.

obvious candidates for the optimal combination of learning rates. This exploration does, however, make it clear which combinations of learning rates are unstable. We also noted that the overall shape of the trajectories associated with small learning rates could not be determined in the window of 500 episodes. Does this perhaps arise from the general principle that the smaller the learning rate, the longer it takes for our value function to accurately approximate the value of each state ?

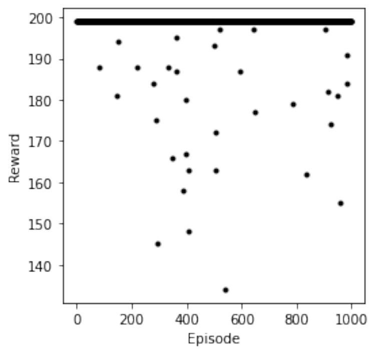
See Chapter 9.6 on Selecting Step-Size Parameters Manually in Sutton and Barto [37]

With only indications of which combinations of hyper-parameters to avoid, a 'Monte Carlo' approach was adopted to exploring hyper-parameters (i.e. combinations of random numbers were explored, typically with $\alpha \geq 0.001$). One particularly successful combination of hyper-parameters that was discovered, whose performance you can watch at the end of the colab notebbook, was $\alpha_{\boldsymbol{\theta}} = 0.002, \alpha_{\boldsymbol{w}} = 0.003$.

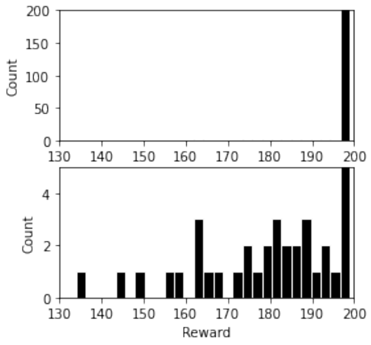
In order to deal with the variability seen in the rewards trajectories, and hence, in the performance of the agent, a simple method of 'saving weights' was implemented. Each time the average performance (measured over the last 100 episodes) of an agent achieved a new best, the current policy and value function weights were saved. A fateful training session consisting of exactly 7000 episodes, with learning rates $\alpha_{\boldsymbol{\theta}} = 0.002, \alpha_{\boldsymbol{w}} = 0.003$ culminated in a brief window of near perfect performance at approximately episode number 4000. The weight saving mechanism kicked in, and this window of near perfect performance was saved. The plot to the right shows the reward trajectory for this particular training session. Since the average performance over the past 100 episodes is plotted, the spike at the 4000 episode mark would mean that from around episode 3900 to episode 4000, the agent consistently achieved a score of around 199 (the maximum score in this implementation). Although I paint this result as miraculous, captured by chance, the initial motivation to save weights was precisely because this Actor-Critic implementation has a tendency to produce brief windows of near perfect performance.



An agent was loaded with this saved set of weights and a further 1000 episodes were simulated in order to quantify its performance on a larger scale. Perhaps its miraculous performance was a result of a fortunate window of starting states generated by the environment. For these 1000 episodes, the weights were not updated. The agent achieved an average score 198.265 over the 1000 episodes, with a minimum score of 134. To get a sense of how consistent this performance was, the scatter plot to the right shows the rewards for each of the 1000 episodes. One can quickly identify the 30 or so episodes where the agent scored scores less than 200.



Tallies of the number of the number of episodes where the agent's rewards fell within various bins of 2.5 units are also displayed as histograms to the right below the scatter plot. The top histogram makes it look as if the agent only every scored in the bin $(197.5, 200)$, but in the lower histogram, we zoom into the lower section of the histogram to reveal the tallies of the lower scores.



Actor-critic resources

1. Applying asynchronous gradient descent to an Actor-Critic method
2. Soft Actor Critic explanation in Spinning Up
3. Actor Critic implementation in Tensorflow 2

4.3. Deep Q-Learning

Explanation

In order to understand Deep Q-learning, it is worth revising Q-learning. Q-learning is an example of off-policy, temporal-difference learning. Temporal-difference learning combines the versatility of Monte-Carlo methods, which do not require full models of the environment and learn from experience, with bootstrapping, which, in this case, involves making 'guesses' on the basis of other guesses. This is an idea from dynamic programming . Off-policy methods make use of two policies - a policy that is being learnt about, called the *target policy*, and a policy that is used to generate behaviour, called the *behaviour policy*. A benefit of interacting with the environment with a separate policy, is that we can maintain exploration with the behaviour policy while making updates to the target policy, (ideally) moving towards the optimal policy. With on-policy methods, the policy that interacts with the environments is the same as the one we update, and in order to maintain exploration, one has to settle with moving towards a near-optimal policy.

See the sections on temporal-difference learning and off-policy methods in Sutton and Barto [37]

Q-learning, as hinted in its name, makes updates to the action-value function, $Q(S_t, A_t)$. These updates take the form:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{5}$$

The behavioural policy interacts with the environment, maintaining a certain amount of exploration, by following an ϵ-greedy policy for instance, and then updates are made to the target policy following equation 5, which assumes that actions are chosen greedily.

Now, moving towards Deep Q-learning, we start by approximating the Q function (which previously was a table of values for each state-action pair) with a deep neural network (a neural network with hidden layers), that predicts the Q-values of different state-action pairs. By approximating the Q function with a deep neural network, we can now consider environments that have continuous states, or unfeasible-large (potentially infinite) state spaces.

Ray Heberer [15] articulates many of the difficulties of implementing Deep Q-learning

However, Deep Q-learning is not as simple as replacing a table of Q values with a deep neural network. Firstly, there are stability issues with using a neural network to learn the Q-values. Secondly, we cannot update Q-values using equation 5 directly. Q-values are the outputs of our neural network. The trainable parameters, the weights of our neural network, are not Q-values. We have to define a loss function - how far off the Q-values output by the neural network are from the true Q-values (or rather, our best guess of the true Q-values) - and use a gradient descent method to update our neural network parameters to minimise this loss.

There are many techniques that make Deep Q-learning more feasible. The two that we will focus on are using *experience replay*, and using separate neural networks for the target and behaviour policies. What is experience replay, and what is its role? Experience replay collects tuples of the state, action taken, reward and next state (S_t, A_t, R_{t+1}, S_{t+1}) at each 'experienced' time step which are then randomly *replayed* to train the model. In Q-learning, the Q-values are updated at each time step with only the latest experience. Using experience replay, we make updates to the behaviour network, drawing on a random selection of collected experiences, not just the latest experience. The updated weights in the behaviour network are then periodically transferred over to the target network.

For information on how experience replay improves the stability of neural networks - see the seminal paper on Deep Q-Networks [26]

During these experience replays, we go through a random selection of the stored experiences, compare how far away our behavioural network's predicted Q-value for the state-action pair is from our best guess of the true Q-value for that pair, and then update the weights in our behavioural network so that the predicted Q-value is closer to this best guess. This 'best guess' is made up of the actual reward, plus the discounted, more steady estimate of the value at the next state (obtained from the target network) $= R_{t+1} + \gamma \max_a Q_{\text{Target}}(S_{t+1}, a)$. The squared difference between this and the behaviour network's predicted Q-value for that state-action pair defines our loss function.

$$L = \Big[R_{t+1} + \gamma \max_a Q_{\text{Target}}(S_{t+1}, a) - Q_{Behaviour}(S_t, A_t)\Big]^2$$

Notice the resemblance of this to the part of update equation 5 in square brackets.

To summarise, we generate experiences using our behaviour network, choosing actions in an ϵ-greedy manner based on the network's predicted Q-value for action at each state. The weights in the behaviour network are updated to minimise the loss. These updated weights are periodically transferred to the target network.

Implementation

Here is a link to the Colaboratory notebook of my Deep-Q network implementation.

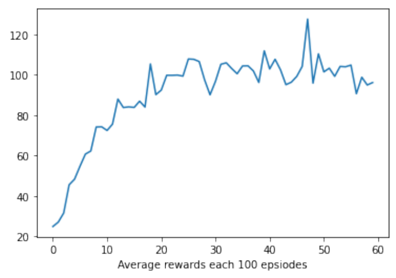
Results

Tuning the Deep-Q network hyper-parameters is a lot more challenging than tuning the REINFORCE hyper-parameters, largely because there are so many more of them. For the current implementation, I used neural networks with 2 hidden layers, and defined the discount rate based on what worked well in the REINFORCE implementation (i.e. undiscounted rewards $\gamma = 1$). I used rectifiers as activation functions. A minimum of 100 experiences are needed before the behaviour network starts updating its weights, and each update involves going through 100 randomly selected experiences from a collection of stored experiences (this collection will contain at most 1000 recent experiences).

Various combinations of hidden units, *copy steps* and learning rates were explored. For each combination, up to 1000 episodes were simulated. Certain combinations weren't simulated for the full 1000 episodes if the average rewards went below a certain threshold. Combinations which obtained average rewards above a certain threshold within the 1000 episodes were 'flagged' with the line "FURTHER TRAINING MERITED", and their subsequent performance was printed out. The results of this hyper-parameter exploration is saved in this document.

The copy step defines how often weights are copied from the behaviour network to the target network

The networks with 4 hidden units seemed to show the most promise. However, even then, the average rewards seem to plateau at around 100. The graph to the right shows the average rewards for every 100 episodes of a DQN. In total, 6000 epsiodes were simulated. This DQN used hyper-parameters that showed promise in the exploration described above. It was made up of 2 layers of 4 hidden units, had learning rate $\alpha = 0.001$, discount rate $\gamma = 1$, copy step $= 50$, and epsilon decayed by $\epsilon_{t+1} = 0.999 \cdot \epsilon_t$. The other parameters are as described above.



DQN resources

1. The DeepMind paper which proposed DQNs
2. Blogpost tutorial of how to implement a DQN in Tensorflow 2
3. A really helpful blog post that clarifies the difference between Q-learning and Deep Q-learning

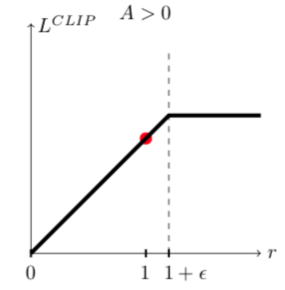
4.4. Proximal Policy Optimization

Explanation

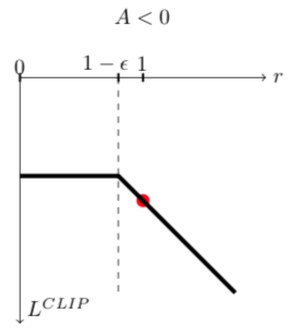
Proximal Policy Optimization (PPO) methods were introduced in 2017 by a team at OpenAI and provided a more data efficient and reliable family of policy gradient methods that alternate between sampling from the environment and optimising a novel objective function that enables multiple epochs of mini-batch updates [31]. This novel objective function uses clipped probability ratios that help increase the stability of parameter updates. This main proposed objective function takes the form:

$$L^{CLIP} = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta)$ is defined as the ratio between the policy and the policy parameterised by the weights θ_{old} before the update, i.e. $r_t(\theta) = \pi_\theta / \pi_{\theta \text{old}}$, and $\hat{A}_t$ is an estimate of the advantage function. The advantage function, which has not been explicitly mentioned until now, is the difference between the action-value and value of a state $A(s,a) = Q(s,a) - V(s)$. One estimate of the advantage function is the the temporal difference error δ_t introduced in the actor critic implementation. The clipping function clip$(r_t(\theta), 1-\epsilon, 1+\epsilon)$ bounds the range of $r_t(\theta)$ between $[1-\epsilon, 1+\epsilon]$. For $1-\epsilon < r_t(\theta) 1+\epsilon$, $(r_t(\theta)$ takes on its usual values. The overall effect of the objective function is that forms a pessimistic lower bound (defined by the clip function) of the unclipped objective. The objective is defined in terms of an estimated expectation, involving a summation of min $\left(r_t(\theta)\hat{A}_t, \, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right)$ over time. For each of these minimums, an illustration of their behaviour is shown to the right, for cases when the advantage is positive and negative. For positive advantages, as the ratio $r(\theta)$ increases, eventually the upper bound of the clipped term becomes the minimum. For negative advantages, the minimum is at most the lower bound of the clipped term. This scheme aims to ignore changes in the probability ratio that would improve the objective function and include changes in the probability ratio that make the objective worse.



Behaviour of L^{CLIP} for $\hat{A}_t > 0$ [31]



Behaviour of L^{CLIP} for $\hat{A}_t < 0$ [31]

Having defined the objective function L^{CLIP}, we briefly discuss the implementation of a PPO algorithm (although this is done concretely as a Colab notebook below). N actors are initialised and interact with the environment for T time-steps each, collecting NT experiences each consisting of a state, action, reward and approximated value of the state. This is ideally computed in parallel. For each of these trajectories, the advantage and total return is computed at each time step. Finally, a 'surrogate loss', which can simply be defined as the objective function L^{CLIP}, or can be modified to account for parameter sharing between the parameters between the policy and the value function and to include an entropy bonus [25], is computed from batches of the collected experiences, and is maximised with respect to the policy gradients using a form of stochastic gradient ascent. Similarly, a loss function associated with the value function is computed from these batches. This loss function can be defined as the mean squared error between the values and the complete returns, in which case a form of stochastic gradient ascent is applied to the value loss function.

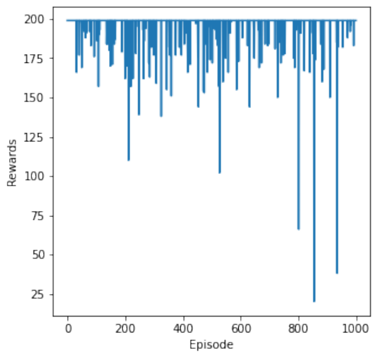
Implementation

Here is a link to the Colaboratory notebook of my implementation of PPO.

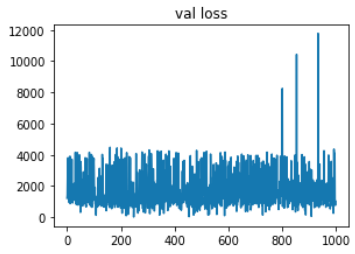
Results

For each phase of accumulating experience, 10 agents interacted with the environment 100 times each, generating 1000 trajectories. In cartpole, it is not feasible to specify a length T for each episode, since episodes can terminate at varying times, especially early on in the training process. Because of this, batches were taken to be individual trajectories, and an epoch consisted of going through all the trajectories that had been accumulated, performing updates to the surrogate policy and value function at the end of each trajectory.

This PPO implementation was one of the last additions to this report. As a result, I have not had the same amount of time to explore hyper-parameters and document the results of varying these. The current hyper-parameters are taken from the REINFORCE hyperparameter exploration. I admit there is no particular reason why these hyperparameters should be the same - the REINFORCE and this PPO implementation are greatly different. The learning rates of the policy α_π and value function α_V were both 0.001. The learning rate determines how aggressive the parameter updates are, and in this setting, is supplied to the *optimizer* which is responsible for back-propagating the gradient of the loss. Un-discounted episodes were used $\gamma = 1$. The feed-forward neural networks for the policy function, mapping state features to probabilities of performing each action, and the value function, mapping state features to a numerical value, both had two fully-connected, hidden layers of 16 units. The policy function output 2 logits, corresponding to action-preferences for going left and right, which were then normalised using a softmax layer.



After 10 epochs of training, the performance of the 10 agents, fitted with the surrogate function weights, is shown to the right. Each agent interacts with the environment for 100 episodes. These 10 sets of 100 episodes are shown one after another. The average performance over these 10×100 episodes was 195.59. The final model weights were saved in the Colaboratory notebook and can be reloaded for the sake of reproducibility. What is quite interesting about this admittedly naive implementation of PPO is the extremely high loss of the value function. The loss of the value function for each of the 100 trajectories of the 10 agents is shown to the right. We see that the loss is in the thousands. My best explanation of this extremely high loss is that in implementing an undiscounted version of cartpole where the value function is not given input as to how long the simulation has been running for, it is unable to arrive at an accurate value function. Identical situations could have very different total returns, distinguishable only by how many time steps have passed. For instance, a steady pole with a stationary pole in the middle of the track at time 0 would have a potential total return of 199, whereas the same scenario at time-step 199 would have a total return of 0. Since the loss of the value function was defined as the mean squared error between it and the total reward, it is incapable of arriving at a truly accurate value approximator without input as to how long the simulation has been running.



PPO resources

1. Initial paper
2. Spinning up resources
3. Tensorflow documentation

References

[1] Martın Abadi et al. 'Tensorflow: A system for large-scale machine learning'. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Amina Adadi and Mohammed Berrada. 'Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI)'. In: *IEEE Access* 6 (2018), pp. 52138–52160.

[3] Itamar Arel et al. 'Reinforcement learning-based multi-agent system for network traffic signal control'. In: *IET Intelligent Transport Systems* 4.2 (2010), pp. 128–135.

[4] Adrià Puigdomènech Badia et al. 'Agent57: Outperforming the atari human benchmark'. In: *arXiv preprint arXiv:2003.13350* (2020).

[5] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. 'Neural machine translation by jointly learning to align and translate'. In: *arXiv preprint arXiv:1409.0473* (2014).

[6] Marc G Bellemare et al. 'The arcade learning environment: An evaluation platform for general agents'. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[7] Eli Bendersky. *The Softmax function and its derivative*. Oct. 2016. URL: `https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative`.

[8] Christopher Berner et al. 'Dota 2 with Large Scale Deep Reinforcement Learning'. In: *arXiv preprint arXiv:1912.06680* (2019).

[9] *CartPole Environment*. 2020. URL: `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`.

[10] Kyunghyun Cho et al. 'Learning phrase representations using RNN encoder-decoder for statistical machine translation'. In: *arXiv preprint arXiv:1406.1078* (2014).

[11] Mengnan Du, Ninghao Liu and Xia Hu. 'Techniques for interpretable machine learning'. In: *Communications of the ACM* 63.1 (2019), pp. 68–77.

[12] Lasse Espeholt et al. 'Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures'. In: *arXiv preprint arXiv:1802.01561* (2018).

[13] Samuel Greydanus et al. 'Visualizing and understanding atari agents'. In: *International Conference on Machine Learning*. 2018, pp. 1792–1801.

[14] Bradley Hayes and Julie A Shah. 'Improving robot controller transparency through autonomous policy explanation'. In: *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*. IEEE. 2017, pp. 303–312.

[15] Ray Heberer. *Why Going from Implementing Q-learning to Deep Q-learning Can Be Difficult*. Feb. 2020. URL: `https://towardsdatascience.com/why-going-from-implementing-q-learning-to-deep-q-learning-can-be-difficult-36e7ea1648af`.

[16] Matthew Johnson et al. 'The Malmo Platform for Artificial Intelligence Experimentation.' In: *IJCAI*. 2016, pp. 4246–4247.

[17] James C King. 'Symbolic execution and program testing'. In: *Communications of the ACM* 19.7 (1976), pp. 385–394.

[18] B Ravi Kiran et al. 'Deep reinforcement learning for autonomous driving: A survey'. In: *arXiv preprint arXiv:2002.00444* (2020).

[19] Pieter Libin et al. 'Deep reinforcement learning for large-scale epidemic control'. In: *arXiv preprint arXiv:2003.13676* (2020).

[20] Peter Lipton. 'Contrastive explanation'. In: *Royal Institute of Philosophy Supplement* 27 (1990), pp. 247–266.

[21] Minh-Thang Luong, Hieu Pham and Christopher D Manning. 'Effective approaches to attention-based neural machine translation'. In: *arXiv preprint arXiv:1508.04025* (2015).

[22] Laurens van der Maaten and Geoffrey Hinton. 'Visualizing data using t-SNE'. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[23] Stephanie Milani et al. 'The MineRL Competition on Sample-Efficient Reinforcement Learning Using Human Priors: A Retrospective'. In: *arXiv preprint arXiv:2003.05012* (2020).

[24] Tim Miller. 'Explanation in artificial intelligence: Insights from the social sciences'. In: *Artificial Intelligence* 267 (2019), pp. 1–38.

[25] Volodymyr Mnih et al. 'Asynchronous methods for deep reinforcement learning'. In: *International conference on machine learning*. 2016, pp. 1928–1937.

[26] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. DOI: `10.1038/nature14236`.

[27] Alexander Mott et al. 'Towards interpretable reinforcement learning using attention augmented agents'. In: *Advances in Neural Information Processing Systems*. 2019, pp. 12329–12338.

[28] Travis E Oliphant. 'Python for scientific computing'. In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20.

[29] William JM Probert et al. 'Context matters: using reinforcement learning to develop human-readable, state-dependent outbreak response policies'. In: *Philosophical Transactions of the Royal Society B* 374.1776 (2019), p. 20180277.

[30] Erika Puiutta and Eric Veith. 'Explainable Reinforcement Learning: A Survey'. In: *arXiv preprint arXiv:2005.06247* (2020).

[31] John Schulman et al. 'Proximal policy optimization algorithms'. In: *arXiv preprint arXiv:1707.06347* (2017).

[32] Alexander A Sherstov and Peter Stone. 'Improving action selection in MDP's via knowledge transfer'. In: *AAAI*. Vol. 5. 2005, pp. 1024–1029.

[33] Tianmin Shu, Caiming Xiong and Richard Socher. 'Hierarchical and interpretable skill acquisition in multi-task reinforcement learning'. In: *arXiv preprint arXiv:1712.07294* (2017).

[34] David Silver et al. 'Mastering the game of go without human knowledge'. In: *Nature* 550.7676 (2017), pp. 354–359.

[35] Pei-Hao Su et al. 'Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management'. In: *arXiv preprint arXiv:1707.00130* (2017).

[36] Ilya Sutskever, Oriol Vinyals and Quoc V Le. 'Sequence to sequence learning with neural networks'. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[37] Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction*. 2nd ed. The MIT Press, 2018.

[38] Tijmen Tieleman and Geoffrey Hinton. 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude'. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[39] Marin Toromanoff, Emilie Wirbel and Fabien Moutarde. 'Is Deep Reinforcement Learning Really Superhuman on Atari? Leveling the playing field'. In: *arXiv preprint arXiv:1908.04683* (2019).

[40] Edward R Tufte. *The visual display of quantitative information*. Vol. 2. Graphics press Cheshire, CT, 2001.

[41] Abhinav Verma et al. 'Programmatically interpretable reinforcement learning'. In: *arXiv preprint arXiv:1804.02477* (2018).

[42] Oriol Vinyals et al. 'Grandmaster level in StarCraft II using multi-agent reinforcement learning'. In: *Nature* 575.7782 (2019), pp. 350–354.

[43] Jasper van der Waa et al. 'Contrastive explanations for reinforcement learning in terms of expected consequences'. In: *arXiv preprint arXiv:1807.08706* (2018).

[44] Lilian Weng. 'Attention? Attention!' In: *lilianweng.github.io/lil-log* (2018). URL: `http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`.

[45] Ronald J Williams. 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[46] Bernhard Wymann et al. 'Torcs, the open racing car simulator'. In: *Software available at http://torcs. sourceforge. net* 4.6 (2000), p. 2.

[47] Tom Zahavy, Nir Ben-Zrihem and Shie Mannor. 'Graying the black box: Understanding DQNs'. In: *International Conference on Machine Learning*. 2016, pp. 1899–1908.