scientific reports

OPEN

Self reward design with fine-grained interpretability

Erico Tjoa[1,2]✉ & Cuntai Guan[1]

The black-box nature of deep neural networks (DNN) has brought to attention the issues of transparency and fairness. Deep Reinforcement Learning (Deep RL or DRL), which uses DNN to learn its policy, value functions etc, is thus also subject to similar concerns. This paper proposes a way to circumvent the issues through the bottom-up design of neural networks with detailed interpretability, where each neuron or layer has its own meaning and utility that corresponds to humanly understandable concept. The framework introduced in this paper is called the Self Reward Design (SRD), inspired by the Inverse Reward Design, and this interpretable design can (1) solve the problem by pure design (although imperfectly) and (2) be optimized like a standard DNN. With deliberate human designs, we show that some RL problems such as lavaland and MuJoCo can be solved using a model constructed with standard NN components with few parameters. Furthermore, with our fish sale auction example, we demonstrate how SRD is used to address situations that will not make sense if black-box models are used, where humanly-understandable semantic-based decision is required.

Reinforcement Learning (RL) and Deep Neural Network (DNN) have recently been integrated into what is known as the Deep Reinforcement Learning (DRL) to solve various problems with remarkable performance. DRL greatly improves the state-of-the-art of control and, in the words of Sutton and Barto[1], *learning from interaction*. Among the well-known successes are (1) the Deep Q-Network[2] which enabled machine to play Atari Games with incredible performance, and (2) AlphaGo that is capable of playing notoriously complex game of Go[3] at and beyond pro human level. Although DNN has proven to possess great potentials, it is a black-box that is difficult to interpret. To address this difficulty, various works have emerged, thus we have a host of different approaches to eXplainable Artificial Intelligence (XAI); see surveys[4–6]. They have shed some lights into the inner working of a DNN, but there may still be large gaps to fill. Note that there is no guarantee that interpretability is even attainable, especially when context-dependent interpretability can be subjective.

In this paper, we propose the Self Reward Design (SRD), a non-traditional RL solution that combines highly interpretable human-centric design and the power of DNN. Our robot (or agent, interchangeably used) rewards itself through purposeful design of DNN architecture, enabling it to partially solve the problem without training. While the initial hand-designed solution might be sub-optimal, the use of trainable DNN modules allows it to be optimized. We show that deep-learning style training might improve the performance of an SRD model or alter the system's dynamic in general.

This paper is arranged as the following. We start with clarifications. Then we briefly go through related works that inspire this paper. Then our interpretable design and SRD optimization are demonstrated with a 1D toy example, RobotFish. We then extend our application to the Fish Sale Auction scenario which we go through more extensively in the main text. It is then followed by brief descriptions of how SRD can be used in 2D robot in the lavaland and MuJoCo simulation (their details in the appendix). We largely focus on the interpretability of design and SRD training, although we also include concepts like unknown avoidance, imagination and continuous control. All codes are available in https://github.com/ericotjo001/srd where the link to our full data can be found.

This paper focuses heavily on interpretable human design

What exactly is this paper about? This paper comprises demonstrations of how some reinforcement learning (RL) problems can be solved in an interpretable manner through self-reward mechanism. Our fish sale auction example also demonstrates an RL-like system that requires decision-making in RL style, but requires high human interpretability and is thus not fully compatible with standard RL optimization. Readers will be introduced to the design of different components tailored to different parts of the problems in a humanly understandable way.

Important note: significance and caveat. The paper has been reorganized to direct readers' focus to *interpretable design* since reviewers tend to focus on standard RL practice instead of focusing on our main proposal, which is the interpretable design. This paper demonstrates the integration of a system that *heavily uses human design*

[1]Nanyang Technological University, Singapore, Singapore. [2]Alibaba Group, Hangzhou, China. ✉email: ericotjo001@e.ntu.edu.sg

augmented by NN. Through this paper, we hope to encourage deep learning practitioners to develop transparent, highly interpretable NN-based reinforcement learning solutions in contrast to standard DRL models with large black-box components. Our designs can be presented in a way that are meaningful down to the granular level. *What we do not claim*: we do NOT claim to achieve any extraordinary performance, although our systems are capable of solving the given problems.

But what is interpretability? While there may be many ways to talk about interpretability, interpretability in the context of this paper is fine-grained, i.e. we go all the way down to *directly manipulating weights and biases* of DNN modules. DNN modules are usually optimized using gradient descent from random initialization, thus the resulting weights are hard to interpret. In our SRD model, the meaning and purpose of each neural network component can be explicitly stated with respect to the environmental and model settings.

How do we compare our interpretability with existing explainable deep RL methods? Since we directly manipulate the weights and biases, our interpretability is at a very low level of abstraction, unlike post-hoc analysis e.g. saliency[7] or semantically meaningful high level specification such as reward decomposition[8]. In other words, we aim to be the most transparent and interpretable system, allowing users to understand the model all the way down to its most basic unit. Unfortunately, this means numerical comparison of interpretability does not quite make sense.

Baseline. We believe comparing performance with other RL methods is difficult since powerful DRL is likely to solve some problems very well w.r.t some measure of accuracy. Furthermore, not only are they often black-boxes that do not work in humanly-comprehensible way, sometimes their reproducibility is not very straightforward[9]. Most importantly, in the context of this paper, focusing on performance distracts readers from our focus on interpretability. If possible, *we want to compare the level of interpretability*. However, quantitative comparison is tricky, and we are not aware of any meaningful way to quantify interpretability that can compare with our proposed fine-grained model. So, what baseline should be used? The short answer is: there is no baseline to measure our type of interpretability. As previously mentioned, this is because our interpretability is fine-grained as we directly manipulate weights and biases. In this sense, our proposed methods are already the most interpretable system, since each basic unit has a specific, humanly understandable meaning. Furthermore, the set up of our auction experiments is not compatible with the concept of reward maximization used in standard RL, rendering comparison of "quality" less viable. In any case, our 2D robot lavaland example achieves a high performance of approximately 90% accuracy, given 10% randomness to allow for exploration, which we believe is reasonable.

Related works: from RL to deep RL to SRD.

RL with imperfect human design. RL system can be set up by human manually specifying the rewards. Unfortunately, human design can easily be imperfect since the designers might not necessarily grasp the full extent of complex problem. For RL agents, designers' manual specification of rewards are fallible, subject to errors and problems such as *negative side effect of a misspecified reward*[10] and *reward hacking*[11]. Dylan's *inverse reward design* (IRD) paper[12] addresses this problem directly: it allows a model to learn beyond what imperfect designers specify - also see the appendix regarding Reward Design Problem (RDP). In this paper, the initialization of our models is generally also imperfect although we use SRD to meaningfully optimize the parameters. Another example of our solution to the imperfect designer problem is through *unknown avoidance*, particularly $w_{unknown}$ in lavaland problem.

From RL to Deep RL to interpretable DRL. Not only is human design fallible, a good design may be difficult to create especially for complex problems. In the introduction, we mention that DRL solves this by combining RL and the power of DNN. However, DRL is a black-box that is difficult to understand. Thus the study of explainable DRL emerges; RL papers that address explainability/interptretability problems have been compiled in some survey papers[13,14]. Saliency, a common XAI method, has been applied to visualize deep RL's mechanism[7]. Relational deep RL uses a relational module that not only improves the agent's performance on StarCraft II and Box-World, but also provides visualization on the attention heads useful for interpretability[15]. Other methods to improve interpretability include reward decomposition, in which each part of the decomposable reward is semantically meaningful[8]; do refer to the survey papers for several other ingenious designs and investigations into the interpretability of deep RL.

In particular, explainable DRL models with manually specified humanly understandable tasks have emerge. Programmatically Interpretable RL[16] (PIRL) is designed to find programmatic policies for semantically meaningful tasks, such as car acceleration or steering. Symbolic techniques can further be used for the verification of humanly interpretable logical statements. Automated search is performed with the help of an oracle i.e. the interpretable model "imitates" the oracle, eventually resulting in an interpretable model with comparable performance. Their interpretable policies consist of clearly delineated logical statements with some linear combinations of operators and terms. The components compounded through automatic searches might yield convoluted policies, possibly resulting in some loss of interpretability. By contrast, our model achieves interpretability by the strengths of activation of semantically meaningful neurons which should maintain their semantic assuming no extreme modification is performed.

Multi-task RL[17] uses *hierarchical policy architecture* so that its agent is able to select a sequence of humanly interpretable tasks, such as "get x" or "stack x", each with its own policy π_k. Each sub-policy can be separately trained on a sub-scenario and later integrated into the hierarchy. Each sub-policy itself might loss some interpretability if the sub-problem is difficult enough to require a deep learning. By contrast, each of our neurons will maintain its meaning regardless of the complexity, although our neural network could become too complex as well for difficult problems.

From interpreable DRL to SRD. Our model is called *self reward* design because our robot computes its own reward, similar to DRL computation of Q-values. However, human design is necessary to put constraints on how self-rewarding is performed so that interpretability is maintained. In our SRD framework, human designer has the responsibility of understanding the problems, dividing the problems into smaller chunks and then finding the relevant modules to plug into the design in a fully interpretable way (see our first example in which we use convolution layer to create the *food location detector*). We intend to take interpretable DRL to the extreme by advocating the use of very fine-grained, semantically meaningful components.

Other relevant concepts include for example self-supervised Learning. DRL like Value Prediction Network (VPN[18]) is self-supervised. Exploration-based RL algorithm is applied i.e. the model gathers data real-time for training and optimization on the go. Our model is similar in this aspect. Unlike VPN, however, our design avoids all the abstraction of DNN i.e. ours is interpretable. Our SRD training is also self-supervised in the sense that we do not require datasets with ground-truth labels. Instead, we induce semantic bias via interpretable components to achieve the correct solutions. The components possess trainable components, just like DRL, and we demonstrate that our models are thus also able to achieve high performance. Our pipeline includes several rollouts of possible future trajectories similar to existing RL papers that use imagination components, with differences as the following. Compared to uncertainty-driven optimization[19] towards a target value $\hat{y}_i = r_i + \gamma Q'_{i+1}$ (heavily abbreviated), SRD (1) is similar because agent updates on every imaginary sample available, but (2) has different, context-dependent loss computation.

Novelty and contributions

We introduce a hybrid solution to reinforcement learning problem: SRD is a deliberately interpretable arrangement of trainable NN components. Each component leverages the tunability of parameters that has helped DNN achieve remarkable performance. The major novelty of our framework is its full interpretability through component-wise arrangement of our models. Our aim is to encourage the development of RL system with both high interpretability and optimizability via the use of tunable parameters of NN components.

General framework with interpretable components for specialized contexts. In general, SRD uses a neural network for the robot/agent to choose its actions, plus the pre-frontal cortex as the mechanism for self reward; this is somewhat comparable to the actor-critic setup. There is no further strict cookie-cutter requirement for SRD since fine-grained interpretability (as we defined previously) may need different components. However, generally, the components only differ in terms of arrangement i.e. existing, well-known components such as convolution layers are arranged in meaningful ways. In this paper, we present the following interpretable components:

1. stimulus-specific neuron. With a combination of manually selected weights for convolutional kernel or fully connected layers plus the activation functions (e.g. selective or threshold), each neuron is designed to respond to very specific situation, thus greatly improving interpretability e.g. Food Location Detector in robot fish, eq. (1). Named neurons such as *fh*, *ft* in robot fish example and *PG*, *SZ* in fish sale auction example ensure that the role and meaning of these neurons are fully understandable.
2. The self-reward design (SRD). In a mammalian brain, prefrontal cortex (PFC) manages internal decision[20]. Our models are equipped with interpretable PFC modules designed to provide interpretability e.g. we explicitly see why robot fish decides to move rather than eat. We demonstrate how different loss functions can be tailored to the different scenarios.
3. Other components for such as ABA and DeconvSeq for lavaland problem can be found in the appendix.

So, what's the benefit of our model? Interpretability and efficiency. We build our interpretable designs based on existing DNN modules i.e. we leverage the tunable parameters that make Deep RL works. The aim is to achieve both interpretability and good performance without arbitrary specification of reward (like pre-DRL models). *Furthermore, targeted design is efficient*. Our SRD models only use modules from standard DNN modules such as convolution (conv), deconvolution (deconv) and fully-connected (FC) layers with few trainable parameters (e.g. only 180 parameters in Robot2NN). With proper choice of initial parameters, we can skip the long, arduous training and optimization processes that are usually required by DRL models to learn unseen concepts. We trade off the time spent on training algorithm with the time spent on human design, thus addressing what is known as the *sample inefficiency*[21] (the need for large dataset hence long training time) in a human-centric way.

Robot fish: 1D toy example

Problem setting. To broadly illustrate the idea, we start with a one-dimensional model Fish1D with Fish Neural Network (FishNN) deliberately designed to survive the simple environment. Robot Fish1D has *energy* which is represented by a neuron labelled F. Energy diminishes over time. If the energy reaches 0, the fish dies. The environment is $env = [e_1, e_2, e_3]$ where $e_i = 0.5$ indicates there is a food at position i and no food if $e_i = 0$. The fish is always located in the first block of env, Fig. 1A. In this problem, 'food here' scenario is $env = [0.5, 0, 0]$ which means the food is near the fish. Similarly, 'food there' scenario is $env = [0, 0.5, 0]$ or $env = [0, 0, 0.5]$, which means the food is somewhere ahead and visible. 'No food' scenario is $env = [0, 0, 0]$.

Fish1D's Actions. (1) 'eat': recover energy F when there is food in its current position. (2) 'move': movement to the right. In our implementation, 'move' causes env to be rolled left. If we treat the environment as an infinite roll tape and env as fish vision's on the 3 immediately visible blocks, then the food is available every 5 block.

How to design an interpretable component of neural network? First, we want the fish to be able to distinguish 3 scenarios previously defined: food here, food there and no food. Suppose we want a neuron that strongly activates when there is food nearby (name it *food here* neuron, *fh*), another neuron that strongly activates when there is a food nearby (name it *food there* neuron, *ft*), and we want to represent the no-food scenario as 'neither *fh* and *ft* respond'. How do we design a layer with two neurons with the above properties? We use 1D convolution layer and *selective activation* function as $\sigma_{sa}(x) = \epsilon/(||x||^2 + \epsilon)$, as the following.

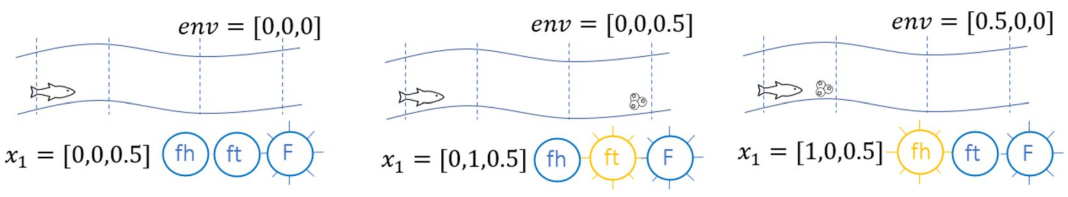

Figure 1. Robot Fish setting. The fish is half full/hungry, $F = 0.5$. Left: no food. Middle: food on env_3, thus "food there" neuron lights up. Right: food on env_1, thus "food here" neuron lights up.

The *fh* and *ft* neurons. Define the activation of *fh* neuron as $a_{fh} = \sigma_{sa}[conv_{fh}(env)]$ where $conv_{fh}$, a Conv1D with weight array $w_{fh} = [1, 0, 0]$ and bias $b_{fh} = 0.5$. When there is food near the fish, we get $y_{fh} = conv_{fh}(env) = [1, 0, 0] * [0.5, 0, 0] - 0.5 = 0$ where $*$ denotes the convolution operator, so $a_{fh} = \sigma_{sa}(y_{fh}) = 1$. This is a strong activation of neuron, because, by design, the maximum value of selective activation function is 1. We are not done yet. Similar to a_{fh} above, define a_{ft}. The important task is to make sure that when 'there is food there but NOT HERE', a_{ft} activates strongly but a_{fh} does not. They are $w_{ft} = [0, 1, 1]$, $b_{ft} = -0.5$. Together, they form the first layer called the Food Location Detector (FLD). Generally, we have used

$$a_\eta = \sigma_{sa}[conv_\eta(env)] \tag{1}$$

Interpretable FishNN. To construct the neural network responsible for the fish's actions (eat or move), we need one last step: connecting the neurons plus fish's internal state (energy) (altogether $[a_{fh}, a_{ft}, F]$) to the action output vector $[eat, move] \equiv [e, m]$ through FC layer, as shown in Fig. 2 blue dotted box. The FC weights are chosen meaningfully e.g. 'eat when hungry and there is food' and to avoid scenarios like 'eat when there is no food'. This is interpretable through manual weight and bias setting.

Is FishNN's decision correct? The prefrontal cortex (PFC) decides whether FishNN's decision is correct or not. PFC is seen in Fig. 2 green dotted box. The name 'PFC' is only borrowed from the neuroscience to reflect our idea that this part of FishNN is associated with internal goals and decisions, similar to real brain[20]. How do we construct PFC? First, define *threshold activation* as $\tau(x) = Tanh(LeakyReLU(x))$. Then PFC is constructed deliberately in the same way FishNN is constructed in an interpretable way as the following.

First, aggregate states from FishNN into a vector $v_0 = [a_{fh}, a_{ft}, F, e, m]$. This will be the input to PFC. Then, v_0 is processed using $conv_{PFC}$ followed by softmax and threshold activation. The choice of weights and biases can be seen in Table 1. With this design, we achieve meaningful activations $v_1 = [e_1, m_1, ex]$ as before. For example, e_1 is activated when "there is food and the fish eats it when it is hungry", i.e. $fh = 1, F < 1$ and e is activated relative to m_1. The output of PFC is binary vector $[True, False] = [T, F]$ obtained from passing v_1 through a FC layer FC_{PFC}. In this implementation, we have designed the model such that the activation of any v_1 neuron is considered a *True* response; otherwise it is considered false. This is how PFC judges whether FishNN's action decision is correct.

Self reward optimization. As seen above, fish robot has FishNN that decides on an action to take and the PFC that determines whether the action is correct. Is this system already optimal? Yes, if we are only concerned with the fish's survival, since the fish will not die from hunger. However, it is not optimal with respect to average energy. We optimize the system through standard DNN backpropagation with the following self reward loss

$$loss = CEL(z, argmax(z)) \tag{2}$$

where CEL is the Cross Entropy Loss and $z = \Sigma_{i=1}^{mem}[T, F]_i$ is the accumulated decision over $mem = 8$ iterations to consider past actions (pytorch notation is used). The ground-truth used in the loss is $argmax(z)$, computed by the fish itself: hence, self-reward design.

Results. Human design ensures survival i.e. problem is solved correctly. Initially, fish robot decides to move rather than eat food when $F \approx 0.5$, but after SRD training, it will prefer to eat whenever food is available, as shown in Fig. 3. New equilibrium is attained: it does not affect the robot's survivability, but fish robot will now survive with higher average energy.

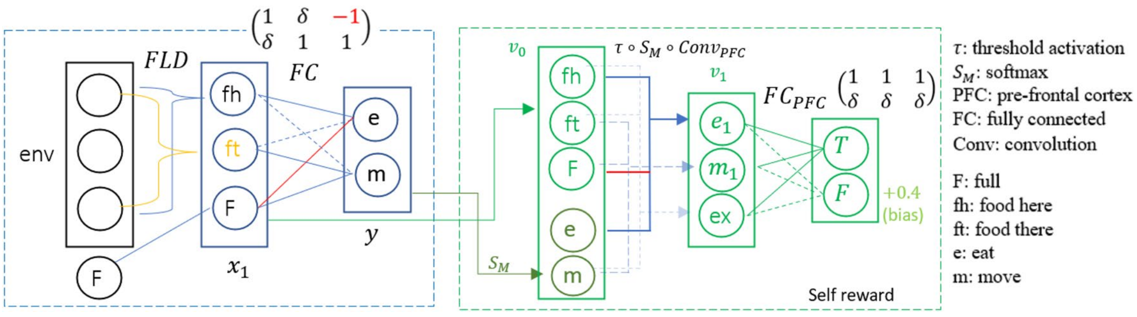

Figure 2. FishNN architecture, $\delta > 0$ a small value.

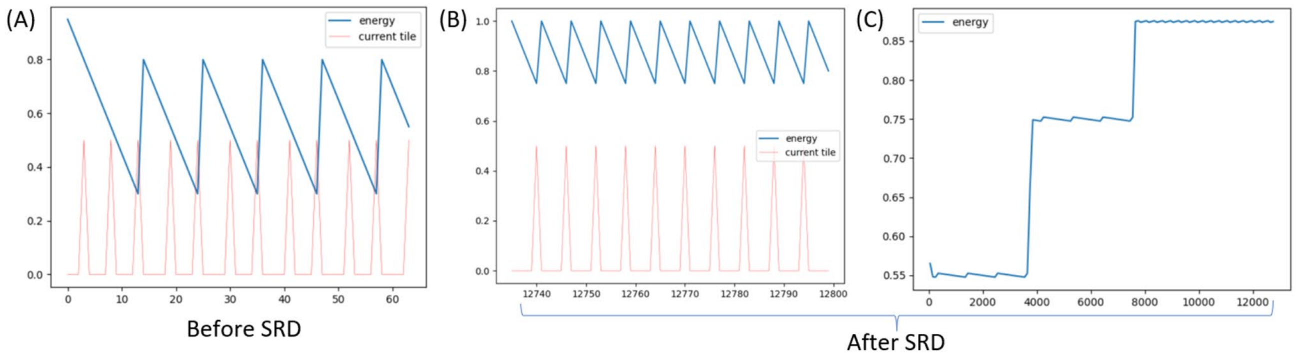

Figure 3. (**A**) plot of energy (F, blue) and food availability (red) for untrained model. SRD-trained model at its early stage looks almost identical. (**B**) same as (**A**) but after 12000 SRD training iterations. (**C**) Average energy of SRD-trained fish model.

Output	Conv. weights	Bias
a_{fh}	[1, 0, 0]	−0.5
a_{ft}	[0, 1, 1]	−0.5
e_1	[1, 0, −1, 1, 0]	None
m_1	[0, 1, −1, 0, 1]	None
ex	[− 1,− 1,0,0,1]	None

Table 1. Weights and biases of Robot Fish's convolution layer.

Fish sale auction

Here, we introduce a more complex scenario that justifies the use of SRD: the fish sale auction. In this scenario, multiple agents compete with each other in their bid to purchase a limited number of fish. A central server manages the dynamic bid by iteratively collecting agents' decision to purchase, to hold (and demand lower price) or to quit the auction. Generally, if the demand is high, the price is automatically marked up and vice versa until all items are sold out, every agent has successfully made a purchase or decided to quit the auction, or the termination condition is reached. In addition, the agents are allowed to fine-tune themselves during part of the bidding process, for example to bias themselves towards purchase when the price is low or when the demand is high.

Interpretability requirement. Here is probably the most important part. The participating agents are required to submit automated models that make the decision to purchase a fish or not based on its price and other parameters (e.g. its length and weight). The automated models have to be interpretable, ideally to prevent agents from submitting malicious models that sabotage the system e.g. by artificially lowering the demand so that the price goes down. Interpretability is required because we want a human agent to act as a mediator, inspecting all the models submitted by the participants, rejecting potentially harmful models from entering the auction.

Deep RL models and other black box models are not desirable in our auction model since they are less amenable to meaningful evaluation. As we already know, deep RL models have become so powerful they might be fine-tuned to exploit the dynamic of a system, and this will be difficult to detect due to their black-box nature. Furthermore, unlike Mujoco and Atari games, there is no reward to maximize in this scenario, especially because the dynamic depends on many other agents' decision. In other words, standard deep RL training may not be compatible with this auction model since there is no true "correct" answer or maximum reward to speak of. Our SRD framework, on the other hand, advocates the design of meaningful self-reward; in our previous 1D robot fish example, PFC is designed to be the interpretable cognitive faculty for the agent to decide the correctness of its own action.

Remark. This scenario is completely arbitrary and is designed for concept illustration. The item being auctioned can be anything else, and, more importantly, the scenario does not necessarily have to be an auction. It could be a resource allocation problem in which each agent attempts to justify their need for more or less resources, or it could be an advertising problem in which each agent competes with each other to win greater exposure or air time.

The server and fish sale negotiator.

Now we describe the auction system and a specific implementation. First, we assume that all agents follow the SRD framework and no malicious models are present, i.e. any undesirable models have been rejected after some screening process. In practice, *screening* is a process whereby human inspector(s) is/are tasked to read the agents' model specifications and manually admit only semantically sensible models. This inspection task is not modeled here; instead, a dummy screener is used to initiated interpretable SRD models by default -we will simulate a system with failed screening process later for comparison.

The server's state diagram is shown in Fig. 4A. Data initialization (not shown in the diagram) is as the following. The main item on sale, the fish, is encoded as a vector $(p, l, w, g, st_1, st_2, st_3, f) \in X \subseteq \mathbb{R}^8$ where $p = 5$ denotes

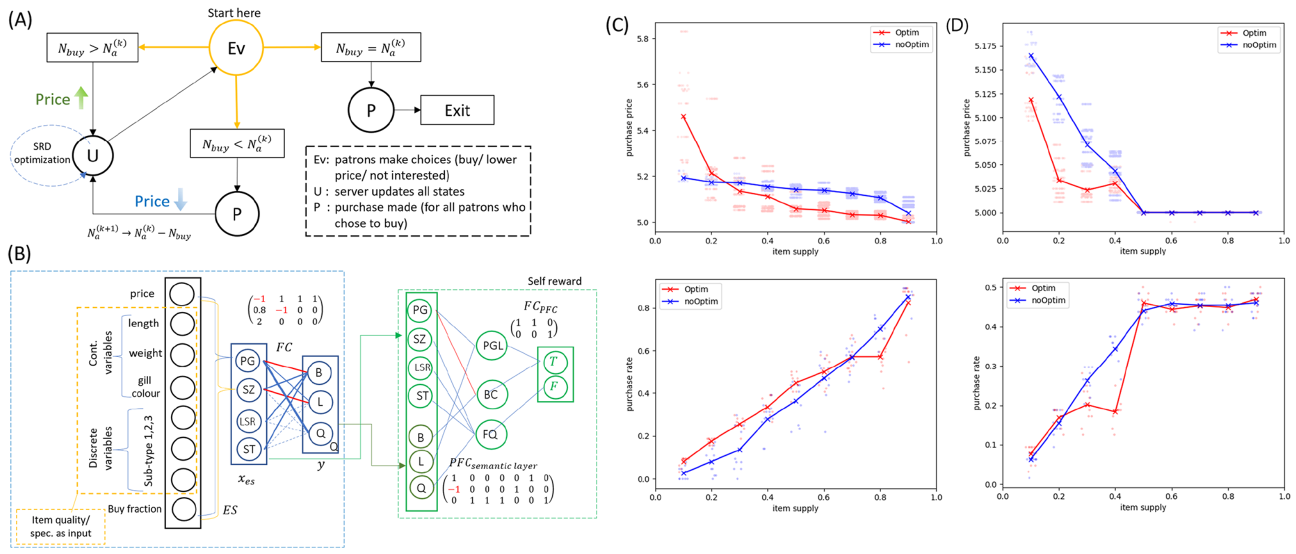

Figure 4. (**A**) Fish sale server. Ev: evaluation process. U: the states update process. P: updating records of successful purchases. (**B**) The Fish Sale Negotiator. (**C**) Optim (noOptim) denotes fish sale auction proceeding in which (no) SRD optimization is performed. Each dot in the background corresponds to an actual purchase price at a given *item supply*, whose value in the plot is slightly perturbed to show multiple purchases at similar prices. *Top*: purchase price vs item supply. With SRD optimization, the inverse trend (red) is more pronounced i.e. when there are much fewer fish available relative to the no. of patrons, the fish tends to be sold at a higher price. *Bottom*: purchase rate vs item supply, where purchase rate is the fraction of available fish sold. (**D**) Same as (**C**) but half the participants submit malicious models to the auction.

the price, l, w, g are continuous variables, respectively the length, weight, and gill colour normalized to 1 (higher values better), st_i discrete variables corresponding to some unspecified sub-types that we simply encode as either -0.5 or 0.5 (technically binary variables) and f the fraction of participants who voted to purchase in the previous iteration (assumed to be 0.5 at the start). The base parameter is encoded as $(5, 1, 1, 1, 0.5, 0.5, 0.5, 0.5)$, and for each run, 16 variations are generated as the perturbed version of this vector (data augmentation).

The server then collects each agent's decision to purchase, hold or quit (corresponding to Ev in the diagram or *_patrons_evaluate* function in the code) and redirects the process to one of the three following branches. If the demand at the specific price and parameters is high, i.e. $N_{buy} > N_a^{(k)}$ where $N_a^{(k)}$ denotes the remaining number of fish available at the k-th iteration, then the states are updated, including price increment. If the demand is low, then purchase transaction is made for those who voted to buy, availability reduced $N_a^{(k+1)} = N_a^{(k)} - N_{buy}$, price lowered, and states are correspondingly updated. If the number of purchase is equal to the number of available fish, the purchase is made and the process is terminated. The above constitutes one iteration in the bidding process and this process is repeated up to a maximum of 64 times. In our experiments we observe that the process terminates before this limit is reached, often with a few fish unsold.

The Fish Sale Negotiator (FSN) is shown in Fig. 4B and this is the fully interpretable SRD model that makes the purchase decision. Each agent will initialize one SRD model with fully interpretable weights, i.e. each agent adjusts their model's parameters based on their willingness to purchase. In our implementation, we add random perturbation to the parameters to simulate the variation of participants. The full details are available in the github, but here we describe how this particular model achieves fine-grained interpretability. FSN is a neural network with (1) the external sensor (ES) module that takes in the fish price and parameters as the input and compute x_{es} and (2) a fully-connected (FC) layer that computes the decision y.

External sensor (ES) module is a 1D convolution layer followed by threshold or selective activations. This module takes the input $x \in X$ and outputs $x_{es} \in \mathbb{R}^4$. The weights w_{ES} and biases b_{ES} of convolution layer are chosen meaningfully as the following (note that its shape is $(4, 1, 8)$ based on pytorch notation). For example, $w_{ES}[0, :, 8]$ corresponds to the PG neuron, which is a neuron that lights up more strongly when the price is higher than baseline and, to a lesser extent, when the continuous variables length, weight and gill colour are lower than 1. Thus, $w_{ES}[0, :, 8] = (1/b, -2\delta, -2\delta, -2\delta, 0, 0, 0, 0) + \delta$ where $b = 5$ is the baseline purchase price and $\delta = 0.001$ is a relatively small number. The bias corresponding to PG is set to 0.

Likewise, other neurons SZ, LSR and ST are initialized similarly: (1) SZ scales with fish sizes i.e. length and weight, thus $w_{ES}[1, :, 8] = (0, 1/2, 1/2, 0, 0, 0, 0, 0) + \delta$ with zero bias (2) LSR with the fraction of purchase in the previous iteration (LSR stands for limited supply rush); semantically, this means that the model is more likely to make the decision to purchase if the demand is high (so that it is not outbidden by competitors) thus $w_{ES}[2, :, 8] = (0, 0, 0, 0, 0, 0, 0, 1) + \delta$ with zero bias (3) ST focuses on the specific sub-type $(0.5, 0.5, 0.5)$, i.e. the neuron is less activated when the sub-type does not match, e.g. $(0.5, -0.5, 0.5)$ thus $w_{ES}[3, :, 8] = (0, 0, 0, 0, 1/3, 1/3, 1/3, 0) + \delta$ with -0.5 bias. To see how these numbers are adjusted, refer to how we run our codes with *--stage observation* argument.

Implicit augmentation is performed by cloning D_{ic} copies of the input $x \in X$ and stacking them to $(p', l', ...f') \in \mathbb{R}^{8 \times D_{ic}}$ where $p' = (p, p_1, ..., p_{D_{ic}})$ (likewise the other variables) and $D_{ic} = 5$ is called the *implicit contrastive dimension* (the number is arbitrarily chosen in this experiment). This is performed as a form of data augmentation in similar spirit to contrastive learning[22–24], in which the cloned copies are randomly perturbed. The *ES* module will have *dilation* $= D_{ic}$ so that it outputs D_{ic} instances of each of the aforementioned neurons that later can be averaged. The benefit of implicit augmentation is clearer in practical application. Suppose the server resides in a remote location and each agent sends and receives its model repeatedly during fine-tuning. If data augmentation is performed in the server and sent to the agent, the network traffic load will be greater. Instead, with implicit augmentation, the process is more efficient since there will be less data transferred back and forth.

Fully-connected layer, FC, is designed similarly. Taking x_{es} at the input, this layer computes the activation of neurons B, L, Q, respectively buy, hold (and lower price) and quit. The decision is made greedily using torch. argmax. We can see that the layer is semantically clear: neuron PG contributes negatively to buy decision, since buying at higher price is less desirable. Neuron SZ contributes positively to neuron B since longer or heavier fish is more desirable; others can be explained similarly. In our experiments, the bias values of FC layer are initiated with some uniform random perturbation to simulate a variation in the strength of intent to purchase the fish.

Prefrontal Cortex (PFC) and SRD optimization. PFC performs the self-reward mechanism, just like 1D robot fish PFC. The neurons are semantically meaningful in the same fashion. PGL neuron strongly activates the true T neuron when PG and L are strongly activated; this means that deciding to hold (L activated) when the price is high (PG activated) is considered a correct decision by this FSN model. BC neuron corresponds to buying at low price, a desirable decision (hence this activates T as well) while FQ neuron corresponds to "false quitting", i.e. the decision to quit the auction when SZ, LSR and ST are activated (i.e. when the fish parameters are desirable) is considered a wrong decision, hence activating F neuron. With this, SRD optimization can be performed by minimizing loss like equation (2). In this particular experiment, optimization setting is randomized from one agent to another and hyperparameters are arbitrarily chosen. The no. of epochs are chosen uniformly between 0 to 2 (note that this means about one third of the participants do not wish to perform SRD optimization), batch size is randomly chosen between 4 to 15 and learning rate on a standard Stochastic Gradient Descent is initiated uniform randomly $lr \in [10^{-7}, 10^{-4}]$. Also, SRD optimization is allowed only during the first 4 iterations (chosen arbitrarily).

Auction results. Each auction in this paper admits $n = 64$ participants given an *item supply* or rarity r. In our code, r is defined such that the no. of available fish on auction corresponds to $r \times n$. Each such auction is repeated 10 times and each trial independent of any other. The results of each trial is then collected, and the mean values of purchase price and purchase rate are shown in Fig. 4C. In this specific implementation, the auction that allows SRD optimization shows a more pronounced price vs supply curve, as shown in Fig. 4C top, red curve. We can see that lower supply results in higher purchase price on average. Figure 4C bottom shows that the fraction of successful buys varies in a nearly linear fashion as a function of supply.

We should remember that the well-behaved trends that we just observed are obtained from *sensible* automated models, in the sense that they make decisions with humanly understandable reasoning such as "buy when the price is low". This is possible thanks to the full interpretability afforded by our SRD framework. For comparison, we are interested in the case where interpretability is lacking and malicious actors sneak into auction. Thus we repeat the experiments, except half the agents always make *hold* decisions, intending to sabotage the system by forcefully lowering the price. The results are shown in Fig. 4D: the graph of purchase price (top) is greatly compromised which translates into a loss for the auction host. The graph of purchase rate (bottom) appears to be slightly irregular at low item supply. However, the purchase rate plateaus off at higher supply (malicious agents refuse to make any purchase) i.e. the auction fails to sell as many fish as it should have, resulting in a loss. By admitting only sensible interpretable models, the auction host can avoid such losses.

More scenarios

Generally, SRD framework encourages developers to solve control problems with neural networks in the most transparent manner. Different problems may require different solutions and there are possibly infinitely many solutions. This paper is intended to be a non-exhaustive demonstration of how common components of black-box neural network can be repurposed into a system that is not black-box. Here, we will briefly describe two more scenarios and leave the details in the appendix: (1) 2D robot lavaland and (2) the Multi-Joint dynamics with Contact (MuJoCo) simulator.

2D robot lavaland. Like Dylan's IRD paper, we use lavaland as a test bed. An agent/robot (marked with blue x) traverses the *lavaland* by moving from one tile to the next towards the objective, which is marked as a yellow tile. With components such as ABA and selective activation function (see appendix), semantically meaningful neurons are activated as we have done in our previous two examples. In this scenario, each neuron will respond to a specific tile in the map. More specifically, brown tiles (dirt patches) are considered easy to traverse while green tiles (grassy patches) are considered harder to traverse. The robot is designed to prefer the *easier* path, thus each neuron responds more favourably towards brown tiles as shown by red patches of v_1 in Fig. 5. The problem is considered solved if the target is reached within 36 steps

Furthermore, we demonstrate how *unknown avoidance* can be achieved. There will be red tiles (lava) that are not only dangerous, but also have never been seen by the robot during the training process (thus unknown). The agent is trained only on green and brown tiles, but when it encounters the unknown red tiles, our interpretable design ensures that the agent avoids the tile, thus *unknown avoidance*. This is useful when we need the agent to "err on the safer side" basis. Once human designer understands more about the unknown, using SRD design

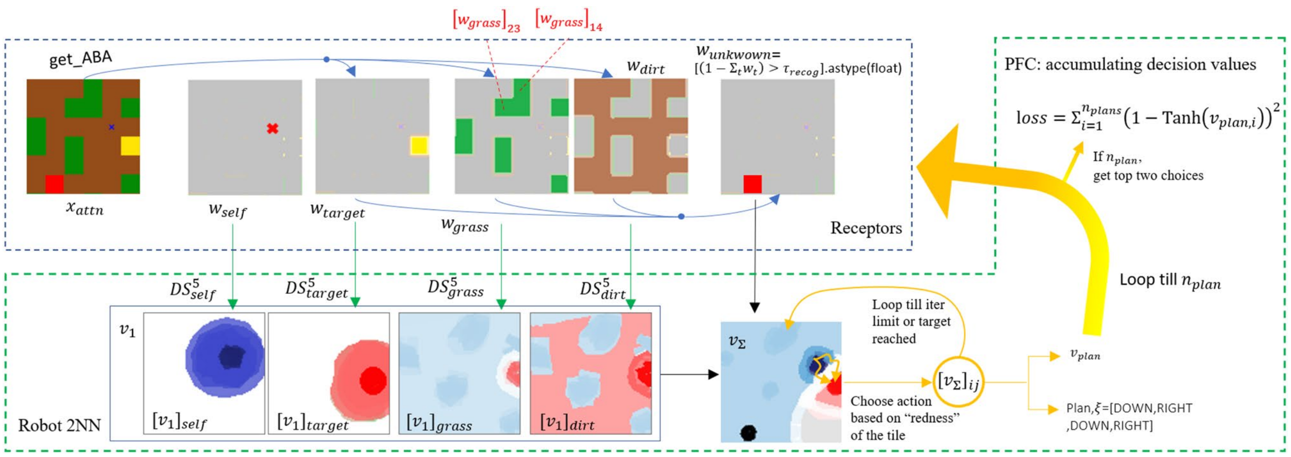
$[w_{grass}]_{23}$ $[w_{grass}]_{14}$
get_ABA
x_{attn}
w_{self}
w_{target}
w_{grass}
w_{dirt}
$w_{unkwown}$= $[(1-\Sigma_t w_t) > \tau_{recog}]$.astype(float)
Receptors
PFC: accumulating decision values
$loss = \Sigma_{i=1}^{n_{plans}}(1-\mathrm{Tanh}(v_{plan,i}))^2$
If n_{plan}, get top two choices
Loop till n_{plan}
DS^5_{self} DS^5_{target} DS^5_{grass} DS^5_{dirt}
v_1
$[v_1]_{self}$ $[v_1]_{target}$ $[v_1]_{grass}$ $[v_1]_{dirt}$
Robot 2NN
v_Σ
Loop till iter limit or target reached
$[v_\Sigma]_{ij}$
Choose action based on "redness" of the tile
v_{plan}
Plan,ξ=[DOWN,RIGHT ,DOWN,RIGHT]

Figure 5. Robot2NN schematic. Receptors help split incoming signals for further processing. x_{attn} (the whole visible map) and w_{self} (the agent's position) are the direct input to the model. In v_1, v_Σ, red values are positive (desirable), white zero and blue negative (not desirable). PFC (green dotted box) contains trainable series of parameters to make and adjust decisions. $[w_{grass}]_{14}$, $[w_{grass}]_{23}$ are examples of strong activations that are interpretable through tile-based module.

principle, a new model can be created by taking into account this unknown. The full experiments and results are available in the appendix.

MuJoCo with SRD. MuJoCo[25] is a well-known open source physics engine for accurate simulation. It has been widely used to demonstrate the ability of RL models in solving control problems. Here, we briefly describe the use of SRD framework to control the motion of a half-cheetah. All technical details are available in the appendix and our github. More importantly, in the appendix, we will describe the design process step by step from the start until we arrive at the full design presented here.

To solve this problem in its simplest setting, an RL model is trained with the goal of making the agent (half-cheetah) learn how to run forward. Multiple degrees of freedom and coordinates of the agent's body parts are available as input and feedback in this control scenario, although only a subset of all possible states will enable the agent to perform the task correctly. More specifically, at each time step, the agent controls its actuators (6 joints of the half cheetah) based on its current pose (we use x and z position coordinates relative to the agent's torso), resulting in a small change in the agent's own pose. Over an interval of time, accumulated changes result in the overall motion of the agent. The objective is for the agent to move its body parts in a way that produces regular motions i.e it runs forward without stalling, tripping, collapsing etc. Typically, this is achieved by a training (maximizing some rewards); without proper training, the agent might end up stuck in an unnatural pose or even start moving backwards.

With SRD, deliberate design of a neural network as shown in Fig. 6A enables the cheetah to start running forward without training (or any optimization of reward), just like our previous examples. Self reward mechanism is also similar: PFC part of the neural network will decide the correctness of the chosen action, and optimization can be performed by minimizing cross-entropy loss as well. Snapshots of the half cheetah are shown in Fig. 6B. The plots of mean x displacement from the original position over time are shown in Fig. 6C (top: without optimization, bottom with SRD optimization). The average is computed over different trials initialized with small noises and the shaded region indicates the variance from the mean values. We test different *backswings*, i.e. different magnitudes with which the rear thighs are swung. Some backswings result in slightly faster motion. More importantly, the structure of the neural network yield stable motion over the span of backswing magnitudes that we tested. Furthermore, we also tested an additional feature: half cheetah is designed to respond to the instruction to stop moving (which we denote with $inhibitor = 2$). The result is shown in Fig. 6D, in which the instruction to stop moving is given at regular intervals. Finally, the effect of SRD optimization is not immediately clear. At lower backswing magnitudes, the optimization might have yielded a more stable configuration, which is more clearly visible in Fig. 6D top (compared to bottom).

Limitation, future directions and conclusion

Generalizability and scalability. An important limitation to this design is the possible difficulty in creating specific design for very complex problems, for example, computer vision problems with high-dimensional state space. Problems with multitude of unknown variables might be difficult to factor into the system, or, if they are factored in, the designers' imperfect understanding of the variables may create a poor model. Future works can be aimed at tackling these problems. Regardless, we should mention that, ideally, a more complex problem can be solved if a definite list of tasks can be enumerated, the robot's state can be mapped to a task and no state requires contradictory actions (or perhaps stochasticity can be introduced). Following the step-by-step approach of our SRD framework, each task can thus be solved with a finite addition of neurons or layers. Further research is necessary to understand how noisy states can be properly mapped to an action.

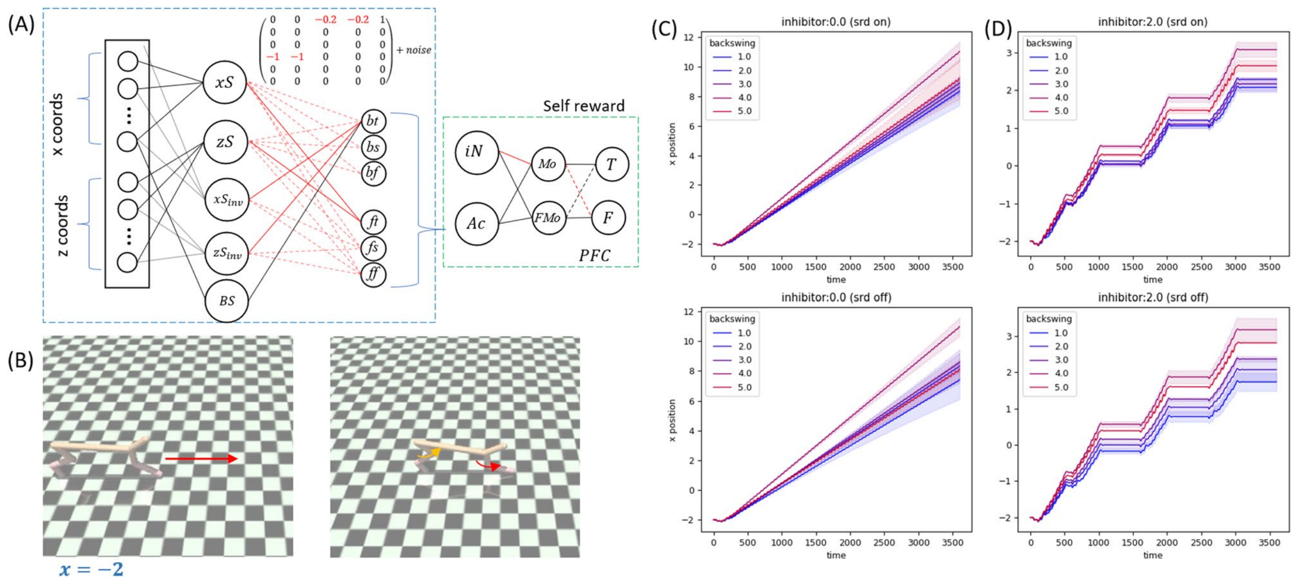

Figure 6. (**A**) The HalfCheetahSRD, the neural network model used to move MuJoCo half-cheetah. (**B**) Half cheetah starts at position $x = -2$. Red/orange curved arrows denote the swings of front/rear thighs. (**C**) mean x position over time for different magnitudes of *backswing*. (**D**) Same as (**C**), but with *inhibitor* set to 2, allowing the agent to response the instruction to stop moving.

Also, an existing technical limitation includes the lack of APIs that perform the exact operations we may need to parallelize the imaginative portions of SRD optimization (see lavaland example in the appendix). We have explored implicit contrastive learning for data augmentation but more research is needed to understand the effect of different techniques. Also, a general formula seems to be preferred in the RL field, and this is not currently available in SRD. For now, a standard SRD framework consists of (1) a NN that decides the agent's actions based on its current state and (2) a PFC that facilitates the process of self-reward optimization with true or false output. Further research is necessary.

Controlling Weights and Biases. A future study on regularizing weights may be interesting. While we have shown that our design provides a good consistency between the interpretable weights before and after training, it is not surprising that the combination of small differences in weights can yield different final weights that still perform well. So far, it is not clear how different parameters affect the performance of a model, or if there are any ways to regularize the training of weight patterns towards something more interpretable and consistent.

To summarize, we have demonstrated interpretable neural network design with fine-grained interpretability. Human experts purposefully design models that solve specific problems based on their knowledge of the problems. SRD is also introduced as a way to improve performance on top of the designers' imperfection. With manual adjustment of weights and biases, designers are compelled to assign meaningful labels or names to specific parts, giving the system a great readability. It is also efficient since very few weights are needed compared to traditional DNN.

Data availability

Our data and results can all be found in the following: https://drive.google.com/drive/u/3/folders/1FoeGgfcO4hdWZynxVFrzPYWvYwIWVZ0p.

Received: 25 May 2022; Accepted: 24 January 2023
Published online: 30 January 2023

References

1. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (MIT press, 2018).
2. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533. https://doi.org/10.1038/nature14236 (2015).
3. Silver, D. *et al.* Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489. https://doi.org/10.1038/nature16961 (2016).
4. Arrieta, A. B. *et al.* Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Inf. Fusion* **58**, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012 (2020).
5. Gilpin, L. H. *et al.* Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, 80–89 (2018).
6. Tjoa, E. & Guan, C. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE Transactions on Neural Networks and Learning Systems* 1–21, https://doi.org/10.1109/TNNLS.2020.3027314 (2020).
7. Greydanus, S., Koul, A., Dodge, J. & Fern, A. Visualizing and understanding Atari agents. In Dy, J. & Krause, A. (eds.) *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, 1792–1801 (PMLR, 2018).

8. Juozapaitis, Z., Koul, A., Fern, A., Erwig, M. & Doshi-Velez, F. Explainable reinforcement learning via reward decomposition. In *Proceedings at the International Joint Conference on Artificial Intelligence. A Workshop on Explainable Artificial Intelligence.* (2019).
9. Henderson, P. *et al.* Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32 (2018).
10. Clark, J. & Amodei, D. Faulty reward functions in the wild. Internet: https://blog.openai.com/faulty-reward-functions (2016).
11. Russell, S. J. *Artificial Intelligence a Modern Approach* (Pearson Education, Inc., 2010).
12. Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. & Dragan, A. D. Inverse reward design. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 6768–6777 (Curran Associates Inc., Red Hook, NY, USA, 2017).
13. Heuillet, A., Couthouis, F. & Díaz-Rodríguez, N. Explainability in deep reinforcement learning. *Knowledge-Based Syst.* **214**, 106685. https://doi.org/10.1016/j.knosys.2020.106685 (2021).
14. Puiutta, E. & Veith, E. M. S. P. Explainable reinforcement learning: A survey. In Holzinger, A., Kieseberg, P., Tjoa, A. M. & Weippl, E. (eds.) *Machine Learning and Knowledge Extraction*, 77–95 (Springer International Publishing, Cham, 2020).
15. Zambaldi, V. *et al.* Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations* (2019).
16. Verma, A., Murali, V., Singh, R., Kohli, P. & Chaudhuri, S. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, 5045–5054 (PMLR, 2018).
17. Shu, T., Xiong, C. & Socher, R. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint*arXiv:1712.07294 *(2017).*
18. Oh, J., Singh, S. & Lee, H. Value prediction network. In Guyon, I. *et al.* (eds.) *Advances in Neural Information Processing Systems*, vol. 30 (Curran Associates, Inc., 2017).
19. Kalweit, G. & Boedecker, J. Uncertainty-driven imagination for continuous deep reinforcement learning. In Levine, S., Vanhoucke, V. & Goldberg, K. (eds.) *Proceedings of the 1st Annual Conference on Robot Learning*, vol. 78 of *Proceedings of Machine Learning Research*, 195–206 (PMLR, 2017).
20. Miller, E. K., Freedman, D. J. & Wallis, J. D. The prefrontal cortex: Categories, concepts and cognition. *Philos. Trans. R. Soc. Lond. Ser. B Biol. Sci.* **357**, 1123–1136 (2002).
21. Kahn, G., Villaflor, A., Ding, B., Abbeel, P. & Levine, S. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 5129–5136, https://doi.org/10.1109/ICRA.2018.8460655 (2018).
22. Chen, X. & He, K. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 15750–15758 (2021).
23. Chen, T., Kornblith, S., Norouzi, M. & Hinton, G. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 1597–1607 (PMLR, 2020).
24. Chen, X., Fan, H., Girshick, R. & He, K. Improved baselines with momentum contrastive learning. *arXiv preprint*arXiv:2003.04297 *(2020).*
25. Todorov, E., Erez, T. & Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033, https://doi.org/10.1109/IROS.2012.6386109 (2012).
26. Singh, S., Lewis, R. L. & Barto, A. G. Where do rewards come from. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 2601–2606 (Cognitive Science Society, 2009).
27. Racanière, S. *et al.* Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 5694–5705 (Curran Associates Inc., Red Hook, NY, USA, 2017).
28. Hafner, D., Lillicrap, T., Ba, J. & Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations* (2020).

Acknowledgements
This research was supported by Alibaba Group Holding Limited, DAMO Academy, Health-AI division under Alibaba-NTU Talent Program. The program is the collaboration between Alibaba and Nanyang Technological University, Singapore. This work was also supported by the RIE2020 AME Programmatic Fund, Singapore (No. A20G8b0102).

Author contributions
E.T. performed the experiments. All authors reviewed the manuscript.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-023-28804-9.

Correspondence and requests for materials should be addressed to E.T.

Reprints and permissions information is available at www.nature.com/reprints.