SELF REWARD DESIGN WITH FINE-GRAINED INTERPRETABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Transparency and fairness issues in Deep Reinforcement Learning may stem from the black-box nature of deep neural networks used to learn its policy, value functions etc. This paper proposes a way to circumvent the issues through the bottom-up design of neural networks (NN) with detailed interpretability, where each neuron or layer has its own meaning and utility that corresponds to humanly understandable concept. With deliberate design, we show that lavaland problems can be solved using NN model with few parameters. Furthermore, we introduce the Self Reward Design (SRD), inspired by the Inverse Reward Design, so that our interpretable design can (1) solve the problem by pure design (although imperfectly) (2) be optimized via SRD (3) perform avoidance of unknown states by recognizing the inactivations of neurons aggregated as the activation in $w_{unknown}$.

1 INTRODUCTION AND RELATED WORKS

Reinforcement Learning (RL) and Deep Neural Network (DNN) have recently been integrated to solve problems with remarkable performance. The deep reinforcement learning greatly improves the state-of-the-art of control and, in the words of Sutton & Barto (2018), *learning from interaction*. Among the well-known successes are (1) the Deep Q-Network (Mnih et al. (2015)) which enabled machine to play Atari Games with incredible performance, and (2) AlphaGo which is capable of playing notoriously complex game of Go (Silver et al. (2016)), which has also been further developed and popularized as being capable of defeating human at pro level. Although DNN has proven to possess great potentials, it is a blackbox 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 Arrieta et al. (2020); Gilpin et al. (2018); Tjoa & Guan (2020). 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.

RL papers that address explainability/interptretability problems have been compiled in survey papers (eg. Heuillet et al. (2021); Puiutta & Veith (2020)). Saliency, a common XAI method, has been applied to visualize deep RL's mechanism (Greydanus et al. (2018)). 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 (Zambaldi et al. (2019)). Other methods to improve interpretability include reward decomposition, in which each part of the decomposable reward is semantically meaningful (Juozapaitis et al. (2019)); do refer to the survey papers for several other ingenious designs and investigations into the interpretability of deep RL. However, gradient-descent optimization of commonly used DNN modules (e.g. fully connected layers) may still end up transforming the data into complex representation that are not interpretable. In our model, we instead assign meaningful values to the parameters. The fact that solutions can be achieved by our NN-based models **even without training** means that meaningful and interpretable weights can be purposefully designed to solve problems with more transparency.

This paper is arranged as the following. We will briefly go through related works that inspire this paper. Then our interpretable design and SRD training idea are demonstrated with a 1D toy example, RobotFish. Following that, we introduce SRD on the main robot navigation problem on the 2D *lavaland* with a large focus on interpretable design, followed by SRD training and unknown avoid-

ance. Its experimental results will be discussed in the section after, and finally, we conclude with limitations and future works.

Interpretable Neural Network Design. In this paper, we demonstrate highly interpretable neural network designs on RL problems. Our interpretability is fine-grained, i.e. the meaning and purpose of each neural network component can be explicitly stated with respect to the environmental and model settings. We aim to be as transparent as possible: in our Robot Fish example, each neuron is even given meaningful semantically-accurate name; in Robot2NN, favourable areas are internally assigned positive values (coloured red). The models only use modules from standard DNN API such as convolution, deconvolution and fully-connected layers with few trainable parameters (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 for deep learning to learn unseen concepts. We trade off the time spent on training algorithm with the time spent on human design, thus addressing what Kahn et al. (2018) called *sample inefficiency* (the need for large dataset hence long training time) in a human-centric way.

Self Reward Design training scheme supplements designers' imperfection. Human design can easily be imperfect since the designers might not necessarily grasp the full extent of the 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* (Clark & Amodei (2016)) and *reward hacking* (Russell & Norvig (2010)). Dylan's *inverse reward design* (IRD) paper (Hadfield-Menell et al. (2017)) addresses this problem directly: it allows a model to learn beyond what imperfect designers specify. In the IRD paper, robots (representing any artificial agent) solve the *lavaland problem*. Robots are tasked to move towards the target while avoiding "lava tiles", which represent parts of the environment that designers fail to properly design into the model. Inspired by IRD's motivation in our pursuit for interpretable AI, we propose the *self reward design* (SRD) and test them on lavaland problems as well. While we try to incorporate interpretability, we have diverted away from the standard RL. Rather than training parameters towards any ground-truth RL policies and rewards (e.g. from experience replay), our robot will "self-reward" or decide the truth of its own decision real-time. Designers induce the context through the initial parameters and the hard-coded instructions for the robots to adjust their own internal parameters that are eventually processed into the worth of actions. Optimization makes up for the imperfection of human design and **we demonstrate it using a possibly novel training loss computations** (1) in Robot Fish, training loss is computed from pre-frontal cortex's output values pitted against itself (more precisely, against its own greedy decision), hence the *self reward* (2) in 2D Robot, normalized self-assigned reward of each plan is optimized towards a single value, which is 1. We empirically show that a short duration of SRD optimization helps improve their performances.

SRD training is self-supervised with explicitly self-assigned truth-values. Kahn et al. (2018) demonstrates self-supervised DRL on a simulated car that learns from real-time experience; the paper also applies the model to a real-world RC car. After hybrid graph-based RL model is incorporated, it travels around avoiding collisions without human intervention. Our robots self-supervise similarly. After the initial interpretable design, 2D Robot moves towards the target, favoring or avoiding different types of tiles without human supervision. In particular, grass tiles are recognizable, but less desirable, while lava tiles are "not recognized", hence must be completely avoided. Informally, the robots have their own preferences on how to assign the eventual values of the actions with known local consequence, while they are careful about their own ignorance induced by imperfect design (unknown avoidance). Arguably, DRL like Value Prediction Network (VPN) by Oh et al. (2017) is self-supervised to the extent the usual value-based exploration-based RL algorithm is applied, where the model computes its own rewards for choosing a series of actions. Our model is similar in this aspect; with exploration, data are gathered real-time during training. Unlike VPN, however, our interpretable design avoids all the abstraction of DNN. Our SRD training is also self-supervised in the sense that we do not require datasets with ground-truth labels as we induce semantic bias via interpretable components.

SRD will incorporate imagination component, i.e. our pipeline includes several rollouts of possible future trajectories similar to existing RL papers that use imagination components, with differences as the following. Unlike the rollout in Racanière et al. (2017), each of our SRD rollout consists of a series of asymmetric binary choices a_1, a_2 chosen from $\{a \in \mathcal{A}\}$ so that $v(a_1) \geq v(a_2)$, where $\mathcal{A}$ is the set of actions and v is any generic function that gives each action a local value. The values will be aggregated into a self-reward. Unlike Kalweit & Boedecker (2017) whose target value is

$\hat{y}_i = r_i + \gamma Q'_{i+1}$ (heavily abbreviated), as we previously mentioned, our target value for Robot2NN update is simply the value 1 with a possibly innovative definition of loss (see eq. 4 later). But, similar to the paper, we also update on every imaginary sample available. Dreamer (Hafner et al. (2020)) solves RL problem using only latent imagination where many models (such as reward, action and value models) are specified as probability distributions. By contrast, SRD creates no specific submodel. All values are just NN activations, and they will be aggregated into impromptu, just-in-time scores, based on which the plans can be greedily chosen.

2 ROBOT FISH: 1D TOY EXAMPLE

To broadly illustrate the idea, we use a one-dimensional model Fish1D with Fish Neural Network (FishNN) deliberately designed to survive the simple environment $env = [e_1, e_2, e_3]$ 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. 1(A). The only allowed movement is to the right, and the env will be rolled left during movement. Furthermore, 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. Also, define *selective activation* as $\sigma_{sa}(x) = \epsilon/(||x||^2 + \epsilon)$ and *threshold activation* as $\tau(x) = Tanh(LeakyReLU(x))$.

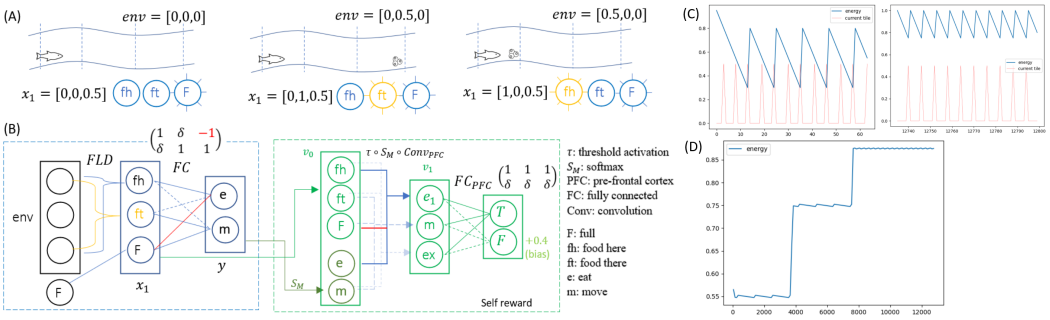

Figure 1: (A) Fish, environment and activations of named neurons. The fish is half full/hungry, $F = 0.5$. Left: no food. Middle: food on e_3, thus "food there" neuron lights up. Right: food on e_1, thus "food here" neuron lights up. (B) FishNN architecture, $\delta > 0$ a small value. (C) Left: plot of energy (F, blue) and food availability (red) for both untrained interpretable model and SRD fish model respectively. Right: same plot, but after 12000 SRD training iterations. (D) Average energy of SRD-trained fish model.

Action choice. The choice of action (eat or move) is controlled by a small neural network (NN) in fig. 1(B) blue dashed-box. The first layer is the *food location detector* (*FLD*), which consists of Conv1D or 1D convolution layer, a regular module in the standard API collection for DNN (we specifically use Pytorch). *FLD* has 1 input channel since it takes env as the input, and 2 output channels that correspond to *food here* (fh) and *food there* (ft) neurons. The fh neuron is activated by $conv_{fh}$, a Conv1D with weight array $[1, 0, 0]$, and, as importantly, its bias $b_{fh} = 0.5$. The weight and bias values are chosen by design, such that each neuron η has a meaningful activation

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

For example, when there is food near the fish, i.e. $env = [0.5, 0, 0]$, we get $y_{fh} = conv_{fh}(env) = [1, 0, 0] * [0.5, 0, 0] - 0.5 = 0$ where $*$ denotes the convolution operator, so $a_{fh} = \sigma_{sa}(y_{fh}) = 1$ shows strong activation. The ft neuron is similar, but for food not near the fish. F neuron is the fish's internal state with value $F \in [0, 1]$ where 0 denotes hungry fish and 1 full, also shown as *energy* in fig. 1(C,D). The choice of action is then determined by the fully-connected layer FC, another regular layer used in DNN. The weights are also designed such that neuron e will activate strongly when "the fish is hungry and food is here". Neuron m works similarly. We want to point out the non-activation case: for example when there is no food, fh and ft will be inactive, and the decision is completely determined by F. With any $F > 0$, neuron m will be activated more strongly than e, so it will choose to move. In our main 2D robot later, we demonstrate how inactivation is used for *lava avoidance*, effectively achieving an interpretable, modular variant of models with a

deliberate semantic-driven design aimed to achieve the motivation of Dylan's IRD of dealing with designers' infallibility.

Self Reward Design. Using Conv1D and FC of popular API allows us to optimize the weights easily. This section demonstrates a self-reward design, where the prefrontal cortex (PFC) modules judge the worth of the fish's own action against its environment; see fig. 1(B) 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 (Miller et al. (2002)). Similar to how action choice is decided, the first self reward layer is a computation using $conv_{PFC}$ followed by softmax and threshold activation. The resulting neurons are similarly interpretable: 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. By reading the initial parameters we design into the system, we can interpret what each neuron is for. In this particular design, the True or T neuron is activated if the total sum of e_1, m, e_x is greater than the False or F neuron that is activated with 0.4 initial bias.

After the "ground-truth" is computed by the PFC, which is the robot's own acting self reward module, backpropagation is then used to perform self reward training. After $mem = 8$ iterations, the cumulated decision $z = \Sigma_{i=1}^{mem}[T, F]_i$ is backpropagated with Cross Entropy Loss, where

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

where Pytorch CEL notation is used, mem is the memory size, corresponding to the fish's ability to account for past decisions. The aim of including cumulative loss over mem iterations is to make sure that apparently poor local decision can be offset by future decisions if the agent does eventually reach the specified target. The loss is computed by pitting fish's own PFC cumulated decision z against its own greedy decision $argmax(z)$, i.e. it asserts its own correctness or falseness, and we have thus attained the final form of self-reward design.

Results. No specific objective has been given and the system is naturally allowed to adapt to its environment. Without training, due to interpretable design, robot fish survives with oscillating energy, F, fig. 1(C) left. Fig. 1(C) and (D) also show that F automatically shifts towards higher value over time. Initially, the fish may not eat the food when it is full. However, after SRD iterations, the fish has evolved to prefer eating food whenever available. This clearly indicates that the initial parameter setting specified by designers' intuition is not in an equilibrium.

3 Self-Rewarding Model with Interpretable Design

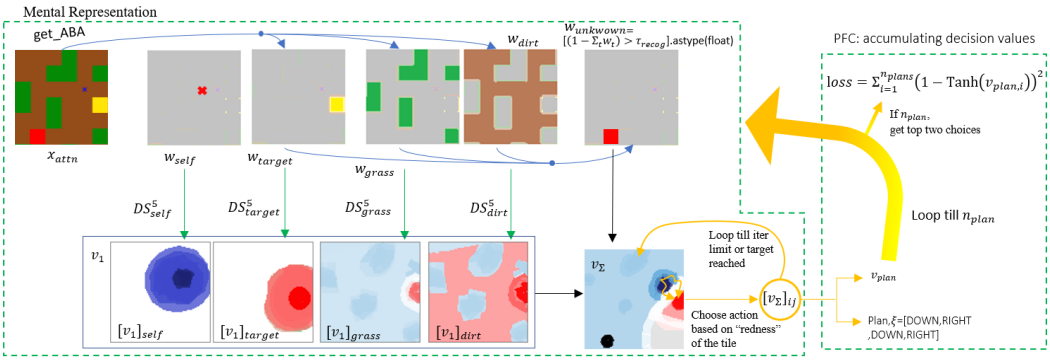

Figure 2: Robot2NN schematic, the brain of 2D robot. x_{attn} (the whole visible map) and w_{self} (the agent's position) are the direct input to the model. In v_1, v_Σ, red values are positive, white zero and blue negative, and the examples are only for illustrations. The lower part of the schematic can be thought of as Robot2NN's pre-frontal cortex, similar to RobotFish's PFC.

The lavaland problem. Like Dylan's IRD paper, we use lavaland as the test bed. Given a map with either dirt (brown), grass (green), lava (red) and target (yellow) tiles and the robot's starting point (blue x), as shown in x_{attn} of fig 2, the robot chooses to move either UP, DOWN, LEFT or RIGHT to reach the target. We treat lava as the "unrecognized tile" that the designer forgets to take into

account (imperfect designer), and show that our robot not only is capable of achieving the target *without training*, but it can (1) also be trained to step less on the grass tiles and (2) be designed to implicitly avoid the lava, or more generally, avoid the unknown. As a rule of thumb, we prefer to make the robot go through dirt tiles than grass tiles towards the target.

In Reward Design Problem, Singh et al. (2009) observes that, given bounded agents, proxy reward function can be more optimal than the true fitness function which is distinct from the proxy. Dylan's IRD, on the other hand, approximates the true fitness function given an observed proxy reward function, assuming the designer is the bounded agent, i.e. the designer is fallible. IRD performs inversion to compute $P(w = w^*|\tilde{w}, \tilde{M})$ from $P(\tilde{w}|w^*, \tilde{M})$, where w is the weight of the reward function $r(\xi; w)$, where ξ is the trajectory. We cannot fit the description of our model exactly in the same language, since we opt for an interpretable design with non-traditional RL reward. The closest we have to a reward is v_{plan} in Robot2NN, shown in fig. 2 indirectly as the values summed through orange arrows in v_Σ described in the next paragraph; also see appendix for remarks on *subtle and technical differences* compared to contemporary design.

We remind readers that this paper's main focus is on human-centric interpretable design, thus, we believe in creating humanly-understandable context-specific design that directly taylors to the problem at hand. The problem in 2D is more complex compared to 1D robot fish, and *favourability gradient* is our choice of explainable design. In summary, robot 2D chooses to move towards tiles that it thinks is the "better" way to reach the target (higher intensity "red" in its mental representation, see fig. 2). Before proceeding to the details, we mention here that the SRD idea is similar: the robot accumulates the values of its choices and uses the final "judgment" value to update itself, hence self-reward design. In robot fish, fig. 1, if the components of vector v_1 layer in PFC are strongly activated, then robot fish is inclined to think its choice of action is correct, i.e. larger T component, hence the accumulated final "judgment" vector z becomes higher. In Robot2NN (see fig. 2), rather than v_1, it is the dynamic vector corresponding to traversed tile values $[v_\Sigma]_{ij}$ that contributes to final "judgment" v_{plan}. We start with notational definitions.

Notations. ABA: approximate binary array, an array whose entries are expected to be ≈ 0 or 1. DS_t^n, or DeconvSeq, is the sequence of n deconvolutional layers for a tile type t. Deconvolutional layer, or deconv, is a regular DNN module. Each deconv is followed by Tanh activation for normalization and non-linearity. Normalization (to magnitude 1) ensures that DeconvSeq compares action choices in relative terms. Tile t denotes the name of a tile, e.g. grass, but it also denotes its $[0, 1]$ normalized RGB value, e.g. for grass, $t = [0, 128, 0]/255$ or an array of tile values (they should be obvious from the context). $\tau_{recog} = 10^{-4}$ is the recognition threshold. Designer needs to specify $P = \{p_t : t = target, grass, ...\}$. For now P is the set of untrainable parameters, each p_t roughly acting as the factor for scaling the true reward, inevitably affecting the robot's final preferences for or against different tiles. They induce the biases that designers input into the model in a simple, interpretable way. We also define the *unknown avoidance* parameter $u_a \geq 0$.

Tile-based modules. Now we introduce the tile-based module intended to explicitly map robot's response to each specific tile type. Particularly, unknown avoidance is clearly processed through $w_{unknown}$, thus providing a clear interpretability to our system. Get_ABA() function computes the ABA for each tile type: $w_t = \sigma_{sa} \circ \mu[(x_{attn} - t)^2]$, where $\mu[.]$ is the mean across RGB channel. This is reminiscent of eq. 1, though each neuron responds to a tile type at each x_{attn} coordinates. A key component to unknown avoidance is the Boolean array $w_{unknown} = [(1 - \Sigma_t w_t) > \tau_{recog}]$ to be treated as floating point numbers, to be used soon. Now, we compute v_1, whose components are $[v_1]_t$, as the following. For target tiles and "self" tiles (original positions), $[v_1]_t = p_t DS_t^5[w_t]$. For any other named tiles, we also create the *favourability gradients* based on the initial and target positions, so that, for example, a grass tile nearer the target can be valued more than a dirt tile. Thus,

$$[v_1]_t = p_t DS_t^5[(w_{target} - w_{self}) * w_t] \tag{3}$$

Each deconv in DeconvSeq consists of 1 input and 1 output channel, has kernel size 3 such that its center value is 1 and side value is 0.1 as seen in fig. 5(A). Fig. 5(B, C) show examples of trained weights. This choice of values is intended roughly to create a centred area of effect, where the center of the tile contributes most significantly to w_t; see for example $[v_1]_{target}$ in fig. 2. Highly positive contribution (dark red) is interpretable as the favourable destination, and thus the target is designed to be given the highest positive value through initial parameters. Likewise, unfavourable tiles are assigned blue, and neutral tiles are white. Throughout this paper, grass tiles are less desirable, and lava tiles should be avoided at all cost.

Table 1: Accuracy comparisons for Project A, project Compare A, project With-Lava-A and project Lava NOAV A. $Acc = n_{reached\ target}/4096$. "No SRD" indicates no training: reasonable accuracy is attainable purely with design. SRD training generally increases the accuracies. Starred values reflect failure modes, matching their respective histograms; see fig. 3,4.

	Project A		Compare A		With Lava A		Lava NOAV A	
Expt no.	No SRD	SRD	No SRD	SRD	No SRD	SRD	No SRD	SRD
1	0.767	0.856	0.862	0.941	0.826	0.893	0.841	0.884
2	0.765	0.845	0.871	0.589*	0.821	0.893	0.850	0.909
3	0.763	0.817	0.874	0.930	0.819	0.837	0.844	0.868
4	0.760	0.868	0.863	0.422*	0.823	0.762*	0.835	0.907

Making plans. 2D robot makes plans for different trajectories to reach the target, then judges their qualities before choosing one of the trajectories. Each plan is a series of action, and each action is chosen as the following. Compute $v_\Sigma \leftarrow \Sigma_t[v_1]_t$. Set the specific values for the target tile and tile of the original position to v_0 and $-v_0$ respectively, where $v_0 = max\{|v_\Sigma|\}$ is computed without back-propagation gradient to prevent any complications. Finally, to incorporate lava avoidance, or generally avoidance of anything previously unseen, $v_\Sigma \leftarrow v_\Sigma * (1 - w_{unknown}) + -u_a * v_0 * w_{unknown}$. From the current position, robot's neighbouring $[v_\Sigma]_i$ where $i = up, down, left, right$ values are collected, and neighbours with the top two values are chosen. From the top two choices, randomly choose one of them with a 9 to 1 odds, favouring the action with higher value. The randomness is to encourage exploration. After each action, the tile the robot leaves will be assigned $0.9v_0$ to prevent the robot from oscillating back and forth, where v_0 is separately computed inside step_update() function. A series of actions are chosen in this manner until the target is reached or a maximum of 36 iterations are reached. Thus we have obtained a *plan*, where $v_{plan} = \frac{1}{N_\xi}\Sigma_{i,j\in\xi}[v_\Sigma]_{i,j}$, the mean of all values assigned to the chosen tiles where ξ is the trajectory (see fig. 2, orange arrows in v_Σ). The robot then makes $n_{plans} = 4$ plans and executes the plan with the highest v_{plan}.

Imagination and SRD optimization. Like Kalweit & Boedecker (2017), we use all n_{plans} imagination branches for training (SRD optimization) with possibly novel loss minimization:

$$loss = \Sigma_{i=1}^{n_{plans}}(1 - Tanh(N[v_{plan,i}]))^2 \tag{4}$$

where $N[.]$ normalizes v_{plan} to the magnitude of 1, where normalization factor is computed without gradient also to prevent complication. Thus, Robot2NN SRD always undervalues its reward relative to an abstract, ideal value 1. The intended effect is to always maximize v_{plan}. A standard Stochastic Gradient Descent with learning rate 10^{-4} is used for optimization. In our experimental results section, we will see how adjusting P will change the robot's behaviour. We conclude the section with a summary: our robot sample trajectories and values them through a self reward mechanism. It is a self-supervised learning in the sense that no ground-truth rewards (or labels) are needed.

4 Experiment and Results

Reaching Targets without training, optimized by training. Table 1 shows that even without training ("No SRD" columns), our interpretable designs have enabled relatively high rate of problem solving. With SRD, the accuracies are further improved. Note: we repeat the experiments for different project codes, such as Project A, Compare A, Project B, C, D, Compare B, C, D but will only show Project A, Compare A in the main text; for maps with lava tiles, only project Lava A and Lava NOAV A are shown. Other similar results, like project B, C, D, are available either in the supp. materials or the appendix. The results shown in the paper are generally representative. All codes are included, including codes for creating animated gifs of robot traversing the lavaland.

Project A: standard design and SRD on maps without lava tiles. We randomly generate and save 4096 maps to evaluate the performance of both standard design and SRD trained model, and similarly 4096 maps for SRD training. The fraction of grass tiles is 0.3. No robot will be spawned at the target immediately. Here, we use $p_{dirt,grass} = 0.2, -0.8$ chosen empirically. For training, we only run through 1 epoch, taking a very short time to complete (less than 0.5 hour per project without GPU). Fig. 3(B) shows the cumulative histogram of the number of tiles traversed with $p_{target} = 2$.

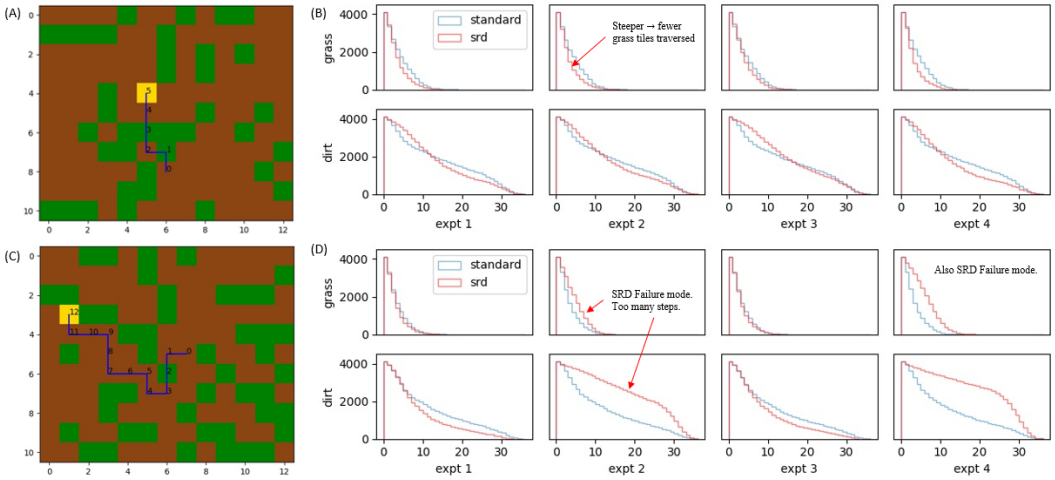

Figure 3: (A) A sample trajectory from Project A. (B) Cumulative histogram of the no. of tiles traversed by robot in Project A. Steeper histogram indicates that less tiles of the type are being traversed (for grass, steeper is better) (C) A sample trajectory from Compare A. (D) Similar to B, but for robot in Compare A.

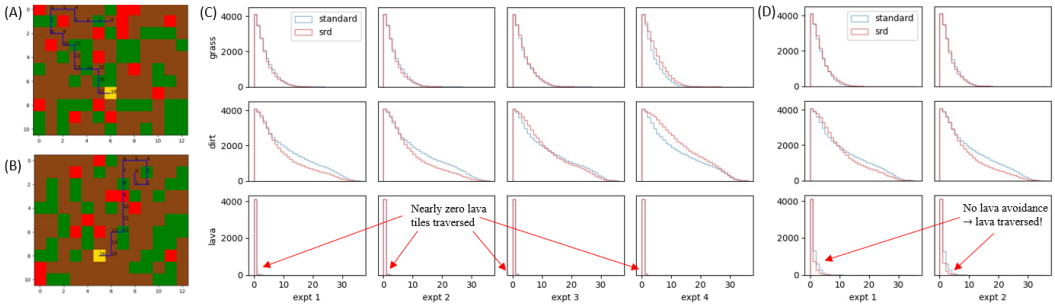

Figure 4: (A) A sample trajectory from Lava A. (B) Cumulative histogram of the no. of tiles traversed by robot in Lava A. Almost zero lava tiles are traversed. (C) A sample trajectory from Lava NOAV A. (D) Similar to (B) but for Lava NOAV A, where some lava tiles are traversed.

Compare A. This is similar to Project A, but we set $p_{target} = 10$. The local reward of the target tile is increased, and from table 1, we see that this generally increases the accuracies of both standard and SRD trained model (more likely to reach the target tile). The value seems to produce more unstable results as well, i.e. we empirically observe more failure modes.

Project Lava A and Project Lava NOAV A. Project Lava A is similar to Compare A, i.e. $p_{target} = 10$, thus we see higher accuracy but some failure modes as well. However, we now have 0.1 of the tiles randomly assigned as lava and unknown avoidance $u_a = 2$. The model is designed to not recognize a lava tile, so a lava tile will activate the robot's Robot2NN $w_{unknown}$ at lavas' positions. Fig. 4(A) is just an example of how robot successfully avoids lava, while fig. 4(C) lava cumulative histograms show that there are nearly zero lava tiles traversed. Project Lava NOAV A is similar to Project Lava A, but we set $u_a = 0$, i.e. no unknown avoidance. The results are clear, we see in 4(D) that robot will traverse the lava tiles without much regards.

Robot2NN weights. Robot2NN model consists of only 180 trainable parameters, as shown fully in fig. 5(B,C). As expected of relatively simple problems, there is no need for millions of parameters required to achieve high accuracy. We will not obtain perfect accuracy because we allow less than optimal choice from explorations. Generally, the weights of target deconv have been trained towards higher positive values while the center value remains the most prominent. Looking into individual variations, fig. 5(B) shows the weights from Robot2NN model of project A expt 1; 5(C) from project

Compare A expt 1. It seems that project A with smaller p_{target} results in relatively less preference for the grass tiles, which in turn leads to negative values (blue) for deconv for grass tiles. By comparison, project Compare A seems to have no negative values for grass tiles deconv. Unfortunately, the weights are shown only for demonstration; *there is no definite conclusion since other experiments similar to project A also can result in all positive deconv weights with different patterns*. They still yield high accuracy, thus possible variations within even this small set of parameters can still produce similar performance. Other results are shown in the appendix. Lava A project does not yield particularly distinct patterns. We see that even failure modes can yield weights profile that look similar to non-failure modes. Further investigations may be necessary.

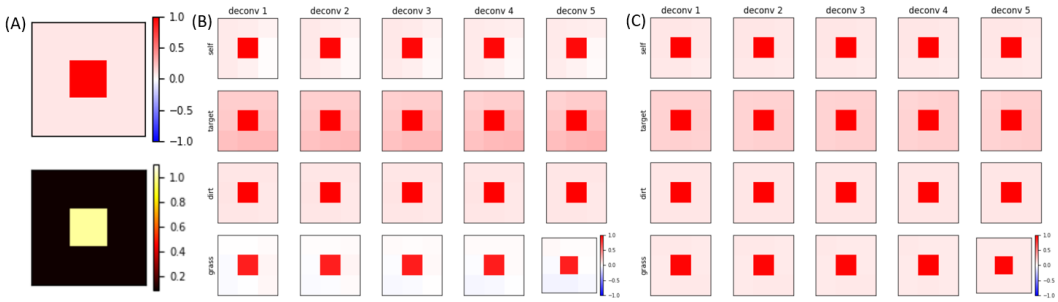

Figure 5: (A) Initial parameters; all deconvs in all DeconvSeq are initialized to the same 3x3 weights with center max value of 1 and 0.1 elsewhere. (B) Once trained, variations of weights are observed for Project A expt 1 (all 180 parameters are shown). (C) Similar to (B) but for project Compare A.

5 Conclusion, Limitation and Future Directions

Limitations and Future directions. 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. Another technical limitation includes the lack of APIs that perform the exact operations we may need to parallelize the imaginative portions of SRD optimization. Finally, 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. So far, there is no guarantee for convergence either, and it may be difficult to provide a general proof, since the design is heavily dependent on the context. Thus, SRD may have many rooms for improvements; future studies aimed at obtaining a reliable design pipeline regardless of context, new modular parallelization and interpretable weight regularization may be desirable.

Conclusion. Finally, 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 introduced as a way to cope with the designers' imperfection. With SRD concept, models have trainable parameters, and optimization process is performed to alleviate that imperfection in a self-supervised manner. Such learning process has been shown to make up for the limited performance due to imperfect human design (e.g. accuracies increased). Interpretability is fine-grained, e.g. each module is meaningful in Robot2NN as it recognizes a specific tile. The recognition is manifested as localized responses (intense values) within w_t, unlike DNN that predicts output through a collection of neuron activations that may require hefty analysis for interpretability. Furthermore, unknown avoidance is explicitly mapped into $w_{unknown}$, creating an interpretable formulation that captures the robots' ignorance that reflect designers' imperfection.

Ethics Statements

There is no specific ethical statement present. We demonstrate methods for designing neural networks for RL problem in a transparent way. We believe each part is transparently presented, and,

if the paper is insufficient, the code in supp. materials should fully complements each description made in this paper.

REPRODUCIBILITY STATEMENTS

All codes are available in the supp. materials, to be released to public repository in case of acceptance. Results are tested on different random number seeds, and irregularities or mode collapses have been presented clearly in the paper as well (see for example the annotations made on the cumulative histograms).

ACKNOWLEDGMENTS

Anonymous now

REFERENCES

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82 – 115, 2020. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2019.12.012. URL http://www.sciencedirect.com/science/article/pii/S1566253519308103.

Jack Clark and Dario Amodei. Faulty reward functions in the wild, 2016. URL https://openai.com/blog/faulty-reward-functions/. Accessed: 2021-08-21.

L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 80–89, 2018.

Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding Atari agents. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1792–1801. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/greydanus18a.html.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca D. Dragan. Inverse reward design. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6768–6777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214:106685, 2021. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2020.106685. URL https://www.sciencedirect.com/science/article/pii/S0950705120308145.

Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *in proceedings at the International Joint Conference on Artificial Intelligence. A Workshop on Explainable Artificial Intelligence.*, 2019.

Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5129–5136, 2018. doi: 10.1109/ICRA.2018.8460655.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 195–206. PMLR, 13–15 Nov 2017. URL https://proceedings.mlr.press/v78/kalweit17a.html.

Earl K. Miller, David J. Freedman, and Jonathan D. Wallis. The prefrontal cortex: categories, concepts and cognition. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 357(1424):1123–1136, Aug 2002. ISSN 0962-8436. doi: 10.1098/rstb.2002.1099. URL https://pubmed.ncbi.nlm.nih.gov/12217179. 12217179[pmid].

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/ffbd6cbb019a1413183c8d08f2929307-Paper.pdf.

Erika Puiutta and Eric M. S. P. Veith. Explainable reinforcement learning: A survey. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl (eds.), *Machine Learning and Knowledge Extraction*, pp. 77–95, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57321-8.

Sébastien Racanière, Théophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 5694–5705, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

Satinder Singh, A. Barto, and Richard L. Lewis. Where do rewards come from. 2009.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning : an introduction*. 2018.

Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020. doi: 10.1109/TNNLS.2020.3027314.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkxaFoC9KQ.

A APPENDIX

A.1 SUBTLE AND TECHNICAL DIFFERENCES

In the main text, we mention that our we use non-traditional reward function. Deep Q-learning loss function $L_i(\theta_i)$ computes differences between current Q value (value of the current state-action pair) and its temporal difference (TD)[1] given by $r + \gamma max_{a'} Q(s', a', \theta_{i-1})$. The reward r can be fixed by the designer; this can be seen on OpenAI's official github[2], and the values are then used to update Q, an expected value of total future rewards, technically also a reward. SRD is similar to DQN in this sense: there is a future cumulative reward. In turn, DQN is similar to standard RL except Q is predicted by the neural network. However, SRD does not apply the loss function $L_i(\theta_i)$ in DQN since we do not specify TD, which is dictated by the reward that explicitly adds into the sum of future reward. As mentioned, the reward value r in DQN is specified by the designer and thus DQN robot aims to maximize values whose absolute magnitude of worth is known. By contrast, SRD lets the robot measure the worth of every series of actions. Since the target Tanh score is 1, every measurement of worth is relative with respect to the learning process. Even then, when the robot compares the multiple plans it makes at once, it has its own yardstick for rewarding its own decisions. The absolute magnitude of reward based on this yardstick is decided by the robot's own neural network, thus the "self reward". Finally, our robot self reward works by always undervaluing its perceived reward w.r.t the ideal plan with score 1, as mentioned in the main text.

More technical side of the implementation includes the adjustment of traversed tile to prevent oscillating behaviour: the tile robot 2D just left is assigned a highly negative value (blue).

A.2 DECONV WEIGHTS

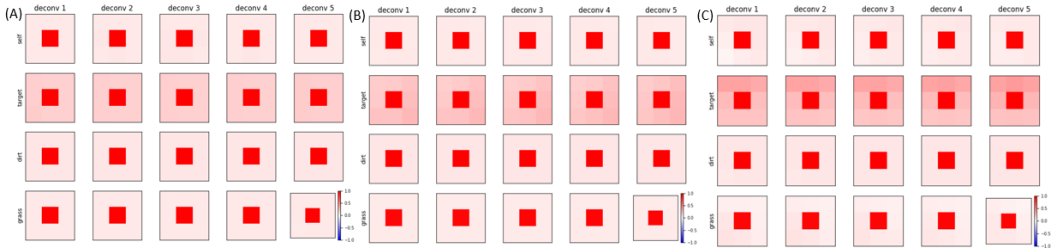

Figure 6: Weights for (A) Project Lava A expt 1 (B) Project Lava NOAV A expt 1 (C) Lava A expt 4.

[1] see for example `https://www.tensorflow.org/agents/tutorials/0_intro_rl`

[2] `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`

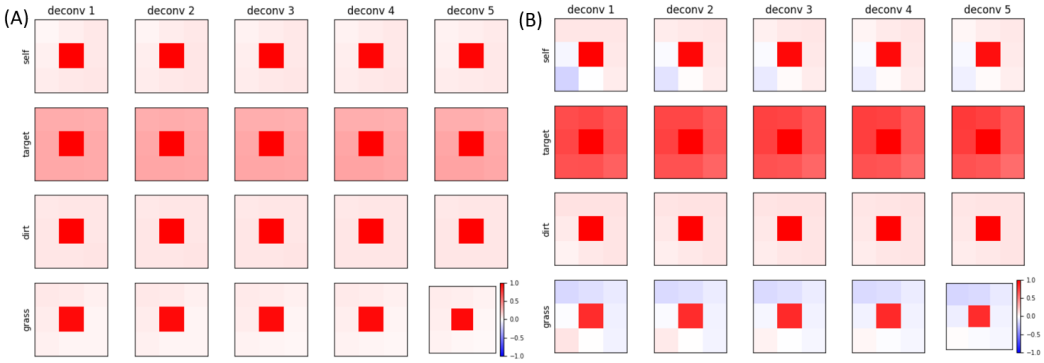

Figure 7: Weights for failure modes (A) Project Compare A expt 2 (B) Project Compare A expt 4. (A) still shows standard-looking weights.