nature portfolio

https://doi.org/10.1038/s41586-025-09761-x

Supplementary information

Discovering state-of-the-art reinforcement learning algorithms

In the format provided by the authors and unedited

Discovering state-of-the-art reinforcement learning algorithms (Supplementary information)

Junhyuk Oh[†], Greg Farquhar[†], Iurii Kemaev[†], Dan A. Calian[†], Matteo Hessel, Luisa Zintgraf, Satinder Singh, Hado van Hasselt, David Silver

All authors are affiliated with Google DeepMind.

[†]These authors contributed equally to this work.

Contents

List of Figures

List of Tables

Table 1: **Meta-network inputs.** For each input, we give examples of the RL algorithmic concepts that inspire it, and example algorithms that use such inputs in computing their objectives. To represent a scalar with y or z, the agent could use a sparse vector, or encode the scalar using a categorical representation (like the two-hot value representation used in prior work[3]). θ^- represents exponentially moving parameters.

Input	Inspiration
$y(s)$	$V(s)$ [A2C[1], PPO[2], MuZero[3], Muesli[4], Dreamer[5]] Distributional $V(s)$ [DA2C[6]]
$z(s,a)$	$Q(s,a)$ [DQN[7], MuZero, Muesli, Dreamer] Distributional $Q(s,a)$ [C51[8]] Successor Features [GPI[9]] Next-state reward [MuZero, Muesli, Dreamer]
$\pi(s,a)$	Policy losses and targets [PG, PPO, MPO[10]] Entropy regularisation [A2C, PPO]
$y_{\theta^-}(s)$, $z_{\theta^-}(s,a)$, $\pi_{\theta^-}(s,a)$	Target networks [DQN, C51, TD3[11]]
$\mu(s,a_{\text{taken}})$	Off-policy corrections [PPO, IMPALA[12], Retrace[13]]
$r(s,a_{\text{taken}})$	Necessary for RL
episode termination	Necessary for RL
$q(s,a)$	Avoid requirement to rediscover concept of action-value [DQN] Semantics are defined by the hardcoded L_{aux}

1 Design of Meta-Learned Rule

The architecture of the meta-learned rule was designed such that its search space encompasses many foundational concepts from existing reinforcement learning (RL) rules. Additionally, some components were designed to make meta-optimisation easier without compromising the generality of the meta-learned rule. The following sections discuss how the meta-learned rule relates to hand-crafted algorithms (Sec. 1.1), why meta-learning targets is more general than meta-learning loss (Sec. 1.2), and why the meta-learned rule is easier to optimise compared to alternatives (Sec. 1.3).

1.1 Search space

Our meta-learned rule requires the agent to produce an observation-conditional vector prediction ($y(s) \in \mathbb{R}^m$) and an action-conditional vector prediction ($z(s,a) \in \mathbb{R}^m$). The meta-network takes these predictions as input, along with other agent outputs, rewards, and an episode-termination indicator. The meta-network outputs are targets towards which the agent's predictions may be updated.

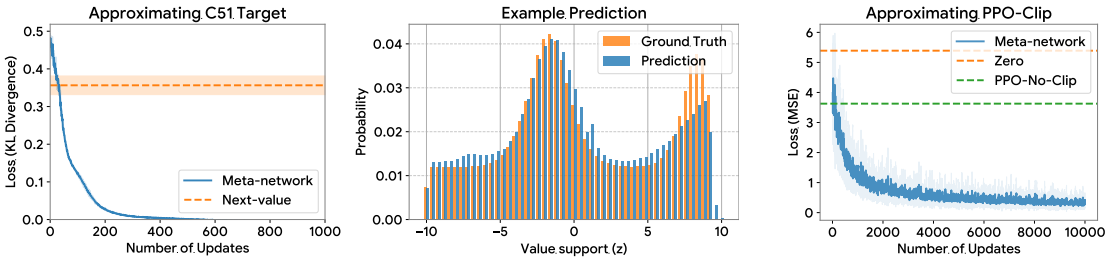
This basic structure, as well as the specific inputs chosen for the meta-network, are designed to incorporate the underlying principles of successful hand-crafted algorithms. Table 1 enumerates the full set of meta-network inputs and describes their relation to existing RL algorithms.

The expressiveness of this functional signature for the meta-network would allow DiscoRL, with an unbounded capacity of the meta-network, a sufficient dimensionality of y and z, and an appropriate distance function, to represent the objective functions of many standard algorithms, including A2C, DQN, and C51.

In practice, we have a limited capacity meta-network, fixed dimensionality of y and z, and choose to use a KL-divergence distance function for its favorable meta-optimisation behaviour (see below). Even with these constraints, we find that our meta-network can effectively represent sophisticated target-generating functions from hand-crafted algorithms, like the distributional bootstrap and projection of C51 (see Figure 1).

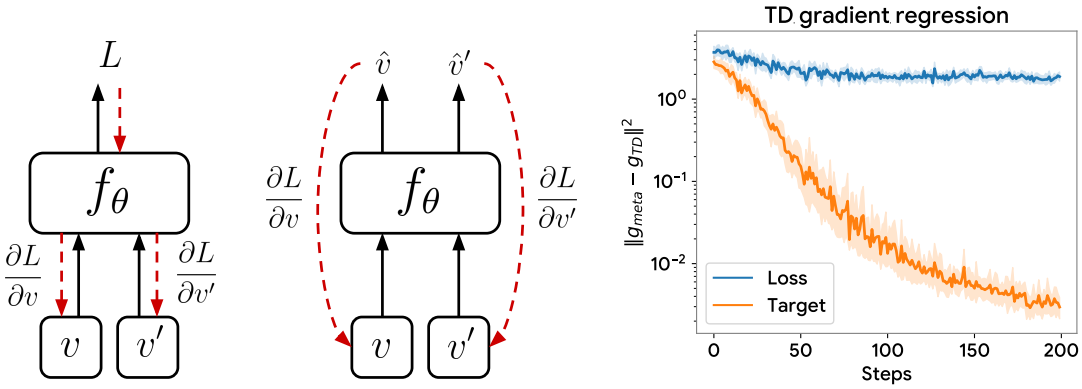
Limitations

As is common in human-led RL research, we focused on a broad but somewhat restricted class of algorithms. For example, the meta-network does not directly get access to observations of the environment. This promotes generalisation across environments, but precludes learning models of observations, inverse dynamics models, or observation-based auxiliary tasks like pixel control[14]. There is no dedicated transition-model



(a) **C51** (b) **PPO-Clip**

Fig. 1: **Approximation of C51 and PPO with meta-network.** (a) The meta-network was trained to predict C51 value targets on an online synthetic data. The data was generated by a mixture of Gaussians for the distributional value. (Left) The meta-network achieved a near-zero loss for approximating C51 targets. The next-value baseline uses v_{t+1} as a prediction. (Right) The meta-network approximately matched the ground-truth C51 target for a non-trivial multi-modal target after training. (b) The meta-network was trained to predict PPO-Clipped ratio on an online synthetic data. It was able to outperform two baselines: a constant zero and PPO ratio without clipping.



(a) **Loss** (b) **Target** (c) **TD gradient regression**

Fig. 2: **Target-based rules can approximate temporal-difference rules, whereas loss-based rules cannot.** Given two scalar input values ($\mathbf{x} = [v, v']$), we consider a simple temporal-difference (TD) loss, $L^*(\mathbf{x}) = \frac{1}{2}||v - \{v'\}||^2$, where $\{\cdot\}$ stops the gradient flow for value bootstrapping. We train two variants of neural networks to approximate the TD gradients with respect to each input value as follows: $\frac{\partial L^*}{\partial v} = v - v'$ and $\frac{\partial L^*}{\partial v'} = 0$. **(a)** In the loss network, the loss is defined as $L = f_\theta(\mathbf{x})$ where f_θ is a neural network parameterised by θ. **(b)** In the target network, the loss is given by $L = D(f_\theta(\mathbf{x}), \mathbf{x})$, where D is a L2 distance function, and $f_\theta(\mathbf{x}) : \mathbb{R}^2 \mapsto \mathbb{R}^2$ is a neural network that produces a target for each input value. **(c)** In both cases, θ is updated to approximate the TD gradients by minimising the loss $L(\theta) = \frac{1}{2}||g_{meta} - g_{TD}||^2$, where $g_{meta} = \nabla_{\mathbf{x}} L(\mathbf{x})$ and $g_{TD} = \nabla_{\mathbf{x}} L^*(\mathbf{x})$, for randomly sampled values $\mathbf{x} = [v, v']$. The target network approximates TD gradients better than the loss network.

component, so while DiscoRL can make multi-step predictions about the future, these may not be conditional on multi-action sequences, as in some model-based algorithms. Other more exotic pieces of information about the state of the agent, such as the internal activations of its network, are also not accessible to the meta-network, so objectives like feature control [14] cannot be represented. Lastly, the scope of discovery space does not include the data sampling and collection mechanism. DiscoRL was discovered with sampling actions from the softmax policy with a replay buffer for learning from off-policy experiences..

1.2 Generality of meta-learning targets

In this section, we show that meta-learning target functions instead of loss functions provides a strictly larger space of update rules for discovery. Our approach meta-learns a *target* function towards which the agent updates its predictions. To simplify notations without loss of generality, let us define the agent's loss function as follows:

$$L = D(\hat{y}, y), \qquad \nabla_y L = \nabla_y D(\hat{y}, y) \tag{1}$$

where $\hat{y} = f(y)$ is a target produced by meta-network f, and y is a prediction from the agent. $D(\cdot,\cdot)$ is a distance function between two vectors. The gradient with respect to the agent's prediction $\nabla_y L$ defines an *update rule*, i.e., how much the agent should adjust its prediction.

A conceptually simpler alternative would be to directly meta-learn a loss function (i.e., $L = f(y)$) such that backpropagation through the meta-network directly gives a gradient $\nabla_y L = \nabla_y f(y)$. This approach has been widely adopted in prior work [15–17].

The space of gradient fields that each approach can represent can be written as

$$G_{\text{target}} = \{\nabla_y D(\hat{y}, y) \mid \hat{y} = f(y), f \in F\} \qquad G_{\text{loss}} = \{\nabla_y f(y) \mid f \in F\}, \tag{2}$$

where F is the function space that neural networks can represent. We show that $G_{\text{loss}} \subseteq G_{\text{target}}$ by showing that for any gradient field $g_{\text{loss}} \in G_{\text{loss}}$, there exists $g_{\text{target}} \in G_{\text{target}}$ that can approximate g_{loss} arbitrarily well. The below is the sketch of the proof.

Let $D(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|^2$ be the squared L2 distance. For this choice of D, the effective gradient field for the target function is $g_{\text{target}} = \nabla_y D(\hat{y}, y) = y - \hat{y}$. Define a target function $\hat{y} = y - \nabla_y f(y)$. Since neural networks are universal function approximators, this can be approximated by a neural network $h \in F$ such that $h(y) = \hat{y} = y - \nabla_y f(y) = y - g_{\text{loss}}$. Therefore, for any $g_{\text{loss}} \in G_{\text{loss}}$, we can find $g_{\text{target}} \in G_{\text{target}}$ such that $g_{\text{target}} \approx g_{\text{loss}}$ by choosing an appropriate target and using the L2 distance as a loss. Therefore, $G_{\text{loss}} \subseteq G_{\text{target}}$. This argument is not specific to L2 distances, and can be extended to other smooth loss functions.

As an illustrative example, consider the policy gradient: $\nabla_\pi L_{PG} = Q(s,a)\nabla_\pi \log \pi(a|s)$. This can be represented as a target-based loss $L = \frac{1}{2}\|\hat{\pi} - \pi\|^2$ with $\hat{\pi} = \pi - Q(s,a)\nabla_\pi \log \pi(a|s)$ such that $\nabla_\pi L = \pi - \hat{\pi} = \nabla_\pi L_{PG}$.

Furthermore, the target-based rule can represent update rules that the loss-based rule cannot represent. By construction, the gradient vector fields of loss-based rules $\nabla_y f(y)$ are always conservative, as they are the gradients of scalar potential functions. On the other hand, the gradient vector fields of many existing RL losses are not necessarily conservative. For example, *semi-gradients* for value bootstrapping cannot be represented as the gradient of a potential function. To empirically show this, we trained two neural networks, a loss network and a target network, to approximate the gradient of a simplified temporal-difference (TD) loss. The result in Fig. 2 shows that the target-based approach can approximate TD gradients, whereas the loss-based approach cannot. Therefore, meta-learning targets can search over a strictly larger space of update rules, including semi-gradient update rules in TD, than the update rules that meta-learned losses can capture in previous work.

1.3 Distance function

We use KL-divergence as the distance function between targets from the meta-network and the agent's predictions and policy: $D(\hat{y}, y) = D_{KL}(\hat{y}||y)$. This is because the gradient norm of KL-divergence function with respect to its input is bounded, whereas alternative distance functions such as L2 distance have unbounded gradient norms. To illustrate this, we visualise the gradient norms of different loss functions using a scalar prediction and a scalar target in Fig. 3. We found that using alternatives such as L2 distance can lead to meta-gradient explosion, making meta-optimisation unstable. A similar result was reported in a prior work on meta-gradient [18].

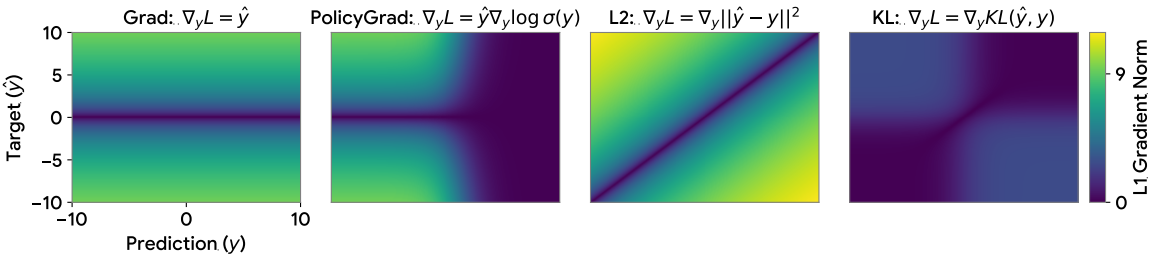
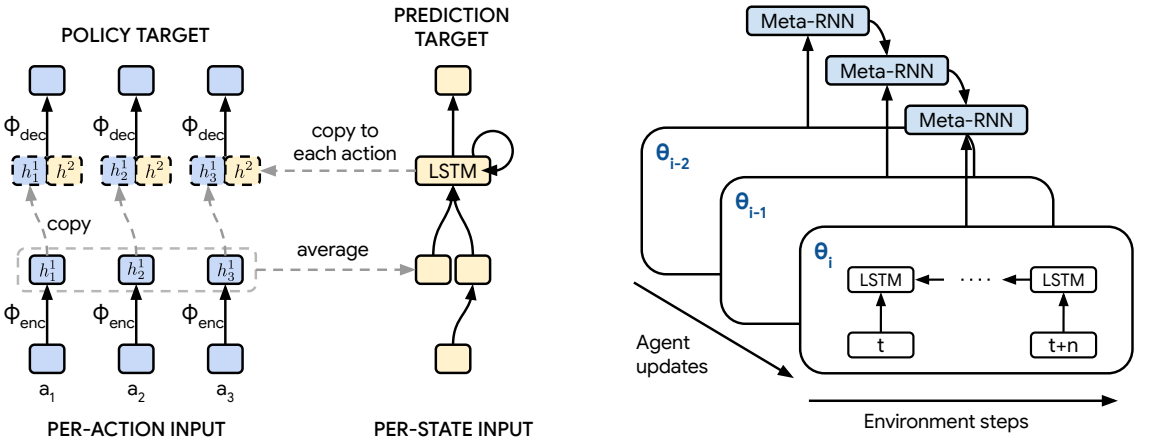


Fig. 3: **Gradient norm of different loss functions.** Each image visualises the L1 norm of the gradient with respect to the scalar prediction (y) given each loss function. In 'Grad' and 'PolicyGrad', the target $\hat{y}$ is directly used as the gradient with respect to the input y and $\log \sigma(y)$, which result in unbounded gradients. L2 distance also produces large gradients as the prediction is farther away from the target, whereas the gradients are bounded between $(-1, 1)$ in KL divergence.



(a) **Action-invariant architecture** (b) **Meta-RNN**

Fig. 4: **Meta-network illustration.** (a) Per-action inputs (e.g., $\pi(s,a), z(s,a)$) are projected to embeddings using an encoder network ϕ_{enc} whose weights are shared across action dimension. The action-specific embeddings are averaged and concatenated to the per-state embedding. This is given as input to the LSTM that is unrolled over environment steps. The LSTM state is copied and concatenated to each action embedding, which is transformed to a policy target $\hat{\pi}$ using a shared decoder network ϕ_{dec}. Meanwhile, the LSTM output is also decoded to prediction targets $(\hat{y}, \hat{z})$. (b) Meta-RNN is unrolled forward over agent update iterations. For each iteration, it takes as input the average of embeddings over all timesteps in each agent update. The output of meta-RNN is used as an additional input to the decoder when generating targets in (a) (not illustrated for simplicity).

2 Meta-Network Details

2.1 Action-invariant architecture

As illustrated in Fig. 4a our meta-network is action-invariant, as it can process inputs corresponding to any number of actions. To enable this, the meta-network processes the action-conditional inputs (e.g., $\pi(s,a_k)$, $z(s,a_k)$ for all action indices k) by first embedding each one into an (action-conditional) embedding $h^1_{a_k}$ using a neural network (ϕ_{enc}) with shared weights over the action dimension. Then, the meta-network extracts the embedding corresponding to the action taken in the environment h^1_a as well as the average embedding across all actions $\frac{1}{K}\sum_k h^1_{a_k}$, and feeds these as inputs into the LSTM. The resulting functional form can thus

process any number of actions, as the number of parameters of the meta-network does not depend on the number of actions. To produce outputs for each possible action (e.g. $\hat{\pi}$), the LSTM output along with the action-conditional input embedding are fed through a separate output module.

The procedure may be summarised as follows:

$$h^1_{a_k} = \phi_{\text{enc}}(\texttt{inputs}_{a_k}) \text{ for all } k \text{ action indices}, \tag{3}$$

$$h^2 = \text{LSTM}\left(..., h^1_a, \frac{1}{K}\sum_k h^1_{a_k}\right) \tag{4}$$

$$\hat{\pi}_{a_k} = \phi_{\text{out}}\left(h^2, h^1_{a_k}\right) \tag{5}$$

The full architecture is included in the released code (https://github.com/google-deepmind/disco_rl).

2.2 Meta-RNN architecture

Fig. 4b shows the conceptual design of the meta-RNN. It is an additional recurrent module that takes as each input the embedding of an entire trajectory of data used for an agent update. It is unrolled forward over sequential agent updates, rather than over timesteps in an episode like the core LSTM of the meta-network. The meta-RNN state is used as an additional input to the decoder for generating targets in Fig. 4a (not illustrated for simplicity).

The motivation behind the meta-RNN is to allow the meta-network to go beyond human-designed rules by taking into account the learning dynamics of the agent throughout its lifetime. Specifically, Meta-RNN allows the meta-network to capture information such as the sparsity and scale of rewards, and episode lengths to provide adaptive updates (e.g., reward normalisation).

3 Meta-Optimisation Details

When updating the meta-value function for A2C, we compute an exponential moving average of the first and second moment of the advantage with a decay of 0.99, and use this to perform advantage normalisation when calculating the A2C gradient.

During meta-optimisation, the agent parameters are updated using the Adam optimiser. We found it useful for meta-optimisation stability to suppress the dependence of the updated parameters θ on the second moment Adam statistics (implemented with a 'stop_gradient' on the optimiser statistics). This is reasonable as our meta-optimisation horizon is shorter than the effective horizon on which these statistics are updated.

References

[1] Mnih, V. *et al.* Asynchronous methods for deep reinforcement learning. *International conference on machine learning* (2016).

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[3] Schrittwieser, J. *et al.* Mastering atari, go, chess and shogi by planning with a learned model. *Nature* **588**, 604–609 (2020).

[4] Hessel, M. *et al.* Muesli: Combining improvements in policy optimization. *International conference on machine learning* (2021).

[5] Hafner, D., Pasukonis, J., Ba, J. & Lillicrap, T. Mastering diverse control tasks through world models. *Nature* (2025).

[6] Li, S., Bing, S. & Yang, S. Distributional advantage actor-critic. *arXiv preprint arXiv:1806.06914* (2018).

[7] Mnih, V. *et al.* Human-level control through deep reinforcement learning. *nature* **518**, 529–533 (2015).

[8] Bellemare, M. G., Dabney, W. & Munos, R. A distributional perspective on reinforcement learning. *International conference on machine learning* (2017).

[9] Barreto, A. *et al.* Successor features for transfer in reinforcement learning. *Advances in neural information processing systems* **30** (2017).

[10] Abdolmaleki, A. *et al.* Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920* (2018).

[11] Fujimoto, S., Hoof, H. & Meger, D. Addressing function approximation error in actor-critic methods. *International conference on machine learning* (2018).

[12] Espeholt, L. *et al.* IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. *International conference on machine learning* (2018).

[13] Munos, R., Stepleton, T., Harutyunyan, A. & Bellemare, M. Safe and efficient off-policy reinforcement learning. *Advances in neural information processing systems* **29** (2016).

[14] Jaderberg, M. *et al.* Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397* (2016).

[15] Houthooft, R. *et al.* Evolved policy gradients. *Advances in neural information processing systems* **31** (2018).

[16] Kirsch, L., van Steenkiste, S. & Schmidhuber, J. Improving generalization in meta reinforcement learning using learned objectives. *International conference on learning representations* (2020).

[17] Xu, Z. *et al.* Meta-gradient reinforcement learning with an objective discovered online. *Advances in neural information processing systems* **33** (2020).

[18] Oh, J. *et al.* Discovering reinforcement learning algorithms. *Advances in neural information processing systems* **33** (2020).