# Introducing Symmetries to Black Box Meta Reinforcement Learning

**Louis Kirsch,**[1 2] **Sebastian Flennerhag,**[1] **Hado van Hasselt,**[1] **Abram Friesen,**[1]
**Junhyuk Oh,**[1] **Yutian Chen**[1]
[1] DeepMind
[2] The Swiss AI Lab IDSIA, USI, SUPSI
`louis@idsia.ch`, `{flennerhag,hado,abef,junhyuk,yutianc}@deepmind.com`

## Abstract

Meta reinforcement learning (RL) attempts to discover new RL algorithms automatically from environment interaction. In so-called black-box approaches, the policy and the learning algorithm are jointly represented by a single neural network. These methods are very flexible, but they tend to underperform in terms of generalisation to new, unseen environments. In this paper, we explore the role of symmetries in meta-generalisation. We show that a recent successful meta RL approach that meta-learns an objective for backpropagation-based learning exhibits certain symmetries (specifically the reuse of the learning rule, and invariance to input and output permutations) that are not present in typical black-box meta RL systems. We hypothesise that these symmetries can play an important role in meta-generalisation. Building off recent work in black-box supervised meta learning [12], we develop a black-box meta RL system that exhibits these same symmetries. We show through careful experimentation that incorporating these symmetries can lead to algorithms with a greater ability to generalise to unseen action & observation spaces, tasks, and environments.

## 1 Introduction

Recent work in meta reinforcement learning (RL) has begun to tackle the challenging problem of automatically discovering general-purpose RL algorithms [13, 1, 17]. These methods learn to reinforcement learn by optimizing for earned reward over the lifetimes of many agents in multiple environments. If the discovered learning principles are sufficiently general-purpose, then the learned algorithms should generalise to novel environments. Depending on the structure of the learned algorithm, these methods can be partitioned into backpropagation-based methods, which learn to use the backpropagation algorithm to reinforcement learn, and black-box-based methods, in which a single (typically recurrent) neural network jointly specifies the agent and RL algorithm [25, 6]. While backpropagation-based methods are more prevalent due to their relative ease of implementation and theoretical guarantees, black-box methods can be more expressive and have the potential to avoid some of the issues with backpropagation-based optimization, such as memory requirements, catastrophic forgetting, and differentiability.

Unfortunately, black-box methods have not yet been successful at discovering general-purpose RL algorithms. In this work, we show that black-box methods exploit fewer symmetries than backpropagation-based methods. We hypothesise that introducing more symmetries to black-box meta-learners can improve their generalisation capabilities. We test this hypothesis by introducing a number of symmetries into an existing black-box meta learning algorithm, including (1) the use of the same learned learning rule across all nodes of the neural network (NN), (2) the flexibility to work with any input, output, and architecture sizes, and (3) invariance to permutations of the inputs

5th Workshop on Meta-Learning at NeurIPS 2021.

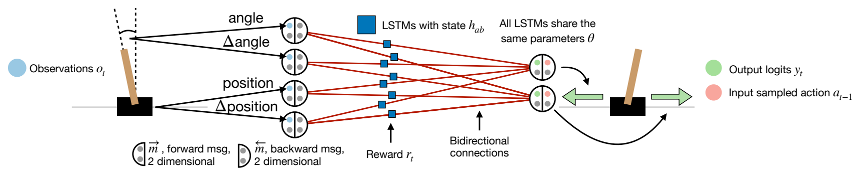
angle
$\Delta$angle
position
$\Delta$position
Observations $o_t$
LSTMs with state $h_{ab}$
All LSTMs share the same parameters $\theta$
Output logits $y_t$
Input sampled action $a_{t-1}$
$\overrightarrow{m}$, forward msg, 2 dimensional
$\overleftarrow{m}$, backward msg, 2 dimensional
Reward $r_t$
Bidirectional connections

Figure 1: The architecture for the proposed *symmetric learning agents* (SymLA) that we use to investigate black-box learning algorithm with symmetries. Weights in a neural network are replaced with small parameter-shared RNNs. Activations in the original network correspond to messages passed between RNNs, both in the forward $\overrightarrow{m}$ and backward $\overleftarrow{m}$ direction in the network. These messages may contain external information such as the environment observation, previously taken actions, and rewards from the environment.

and outputs (for dense layers). Permutation invariance implies that for any permutation of inputs and outputs the learning algorithm produces the same policy. We refer to such agents as *symmetric learning agents* (SymLA).

To introduce these symmetries, we build on variable shared meta learning (VSML) [12], which we adapt to the RL setting. VSML arranges multiple RNNs like weights in a NN and performs message passing between these RNNs. We then perform meta training and meta testing similar to black-box MetaRNNs [25, 6]. We experimentally validate SymLA on bandits, classic control, and grid worlds, comparing generalisation capabilities to MetaRNNs. SymLA improves generalisation when varying action dimensions, permuting observations and actions, and significantly changing tasks and environments.

## 2 Preliminaries: Meta Reinforcement Learning

The RL setting in this work follows the standard (PO)MDP formulation. At every time step, $t = 1, 2, \ldots$ the agent receives a new observation $o_t \in \mathcal{O}$ generated from the environment state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}$ sampled from its (recurrent) policy $\pi_\theta = p(a_t|o_{1:t}, a_{1:t-1})$. The agent receives a reward $r_t \in \mathcal{R} \subset \mathbb{R}$ and the environment transitions to the next state. This transition is defined by the environment dynamics $e = p(s_{t+1}, r_t|s_t, a_t)$. The initial environment state $s_1$ is sampled from the initial state distribution $p(s_1)$. The goal is to find the optimal policy parameters $\theta^*$ that maximise the expected return $R = \mathbb{E}[\sum_{t=1}^{T} \gamma^t r_t]$ where $T$ is the episode length, and $0 < \gamma \leq 1$ is a discount factor ($T = \infty$, $\gamma < 1$ for non-episodic MDPs).

The meta reinforcement learning setting is concerned with discovering novel agents that learn throughout their multi-episode lifetime ($L \geq T$) by making use of rewards $r_t$ to update their behavior. This can be formulated as maximizing $\mathbb{E}_{e \sim p(e)}[\mathbb{E}[\sum_{t=1}^{L} \gamma^t r_t]]$ where $p(e)$ is a distribution of meta-training environments. The objective itself is similar to a multi-task setting. In this work, we discuss how the structure of the agent influences the degree to which it *learns* and *generalises* in novel tasks and environments. We seek to discover *general-purpose* learning algorithms that generalise outside the meta-training distribution.

We can think of an agent that learns throughout its lifetime as a history-dependent map $a_t, h_t = f(h_{t-1}, o_t, r_{t-1}, a_{t-1})$ that produces an action $a_t$ and new agent state $h_t$ given its previous state $h_{t-1}$, an observation $o_t$, environment reward $r_{t-1}$, and previous action $a_{t-1}$. In the case of backpropagation-based learning, $f$ is decomposed into: (1) a *stationary* policy $\pi_\theta^{(s)}$ that maps the current state into an action, $a_t = \pi_\theta^{(s)}(o_t)$; and (2) a backpropagation-based update rule that optimizes a given objective $J$ by propagating the error signal backwards and updating the policy in fixed intervals (e.g. after each episode). In its simplest form, for any dense layer $k \in \{1, \ldots, K\}$ of a NN policy with size $A^{(k)} \times B^{(k)}$, inputs $x^{(k)}$, outputs $x^{(k+1)}$, and weights $w^{(k)} \subset \theta$, the backpropagation update rule is given by

$$x_b^{(k+1)} = \underbrace{\sum_a x_a^{(k)} w_{ab}^{(k)}}_{\text{forward pass}} \quad (1) \qquad \delta_a^{(k-1)} = \underbrace{\sum_b \delta_b^{(k)} w_{ab}^{(k)}}_{\text{backward pass}} \quad (2) \qquad \Delta w_{ab}^{(k)} = -\alpha \frac{\partial J}{\partial w_{ab}^{(k)}} = \underbrace{-\alpha x_a^{(k)} \delta_b^{(k)}}_{\text{update}} \quad (3)$$

where $a \in \{1, \ldots, A^{(k)}\}$, $b \in \{1, \ldots, B^{(k)}\}$, $\alpha$ is the learning rate, $\delta$ are error terms, and the agent state $h$ corresponds to parameters $\theta$. The initial error is given by the gradient at the NN outputs, $\delta^{(k)} = \frac{\partial J}{\partial x^{(K+1)}}$. Transformations such as non-linearities are omitted here. Works in meta-reinforcement learning that take this approach parameterise the objective $J_\phi$ and meta-learn its parameters [13, 17].

In contrast, black-box meta RL [6, 25] meta-learns $f$ directly in the form of a single *non-stationary* policy $\pi_\theta$ with memory. Parameters of $f$ represent the learning algorithm (no explicit $J_\phi$) while the state $h$ represents the policy. In the simplest form of an RNN representation of $f$, given a current hidden state $h$ and inputs $o, r, a$ (concatenated $[\cdot]$), updates to the policy take the form

$$a_b, h_b \leftarrow f_\theta(h, o, r, a)_b = \sigma(\sum_a [h, o, r, a]_a v_{ab}), \tag{4}$$

with parameters $\theta = v$ and activation function $\sigma$, omitting the bias term. We refer to this as the MetaRNN. The inputs must include, beyond the observation $o$, the previous reward $r$ and action $a$, so that the meta-learner can learn to associate past actions with rewards [21, 25]. Further, black-box systems do not reset the state $h$ between episode boundaries, so that the learning algorithm can accumulate knowledge through the agent's lifetime. For additional related work, refer to Appendix A.

## 3 Symmetries in Meta RL

In this section, we demonstrate how the learning dynamics in backpropagation-based systems (Equation 3) differ from the learning dynamics in black-box systems (Equation 4), and how this affects the generalisation of black-box methods to novel environments.

**Symmetries in backpropagation-based Meta RL** We first identify three symmetries that backpropagation-based systems exhibit and discuss how they affect the generalisability of the learned learning algorithms.

1. **Symmetric learning rule.** In Equation 3, each parameter $w_{ab}$ is updated by the same update rule based on information from the forward and backward pass. Meta-learning an objective $J_\phi$ affects the updates of each parameter symmetrically through backpropagation.
2. **Flexible input, output, and architecture sizes.** Because the same rule is applied everywhere, the learning algorithm can be applied to arbitrarily sized neural networks, including variations in input and output sizes. This involves varying $A$ and $B$ and the number of layers, affecting how often the learning rule is applied and how many parameters are being learned.
3. **Invariance to input and output permutations.** Given a permutation of inputs and outputs in a layer, defined by the bijections $\rho : \mathbb{N} \to \mathbb{N}$ and $\rho' : \mathbb{N} \to \mathbb{N}$, the learning rule is applied as $x^{(k+1)}_{\rho'(b)} = \sum_a x^{(k)}_{\rho(a)} w^{(k)}_{ab}$, $\delta^{(k-1)}_{\rho(a)} = \sum_b \delta^{(k)}_{\rho'(b)} w^{(k)}_{ab}$, and $\Delta w^{(k)}_{ab} = -\alpha x^{(k)}_{\rho(a)} \delta^{(k)}_{\rho'(b)}$. Let $w'$ be a weight matrix with $w'^{(k)}_{\rho(a)\rho'(b)} = w^{(k)}_{a,b}$, then we can equivalently write $x^{(k+1)}_{\rho'(b)} = \sum_a x^{(k)}_{\rho(a)} w'^{(k)}_{\rho(a)\rho'(b)}$, $\delta^{(k-1)}_{\rho(a)} = \sum_b \delta^{(k)}_{\rho'(b)} w'^{(k)}_{\rho(a)\rho'(b)}$, and $\Delta w'^{(k)}_{\rho(a)\rho'(b)} = -\alpha x^{(k)}_{\rho(a)} \delta^{(k)}_{\rho'(b)}$. If all elements of $w'^{(k)}$ are initialized i.i.d., we can interchangeably use $w$ in place of $w'$ in the above updates. By doing so, we recover the original learning rule equations for any $a, b$. Thus, the learning algorithm is invariant to input and output permutations.

While backpropagation has inherent symmetries, these symmetries would be violated if the objective function $J_\phi$ would be asymmetric. Formally, when permuting the NN outputs $y = x^{(K+1)}$ such that $y'_b = y_{\rho'(b)}$, $J_\phi$ should satisfy that the gradient under the permutation is also a permutation $\frac{\partial J_\phi(y')}{\partial y'_b} = \left[\frac{\partial J_\phi(y)}{\partial y}\right]_{\rho'(b)}$ where the environment accepts the action permuted by $\rho'$ in the case of $J_\phi(y')$. This is the case for policy gradients, for instance, if the action selection $\pi(a|s)$ is permuted according to $\rho'$. When meta-learning objective functions, prior work carefully designed the objective function $J_\phi$ to be symmetric. In MetaGenRL [13], taken actions were processed element-wise with the policy outputs and sum-reduced by the loss function. In LPG [17], taken actions and policy outputs were not directly fed to $J_\phi$, but instead only the log probability of the action distribution was used.

**Insufficient Symmetries in Black-box Meta RL** Black-box meta learning methods are appealing as they require few hard-coded biases and are flexible enough to represent a wide range of possible learning algorithms. We hypothesize that this comes at the cost of the tendency to overfit to the given meta training environment(s) resulting in overly specialized learning algorithms.

Learning dynamics in backpropagation-based systems (Equation 3) differ significantly from learning dynamics in black-box systems (Equation 4). In particular, meta-learning $J_\phi$ is significantly more constrained, since $J_\phi$ can only indirectly affect each policy parameter $w_{ab}^{(k)}$ through the *same* learning rule from Equation 3. In contrast, in black-box systems (Equation 4), each policy state $h_b$ is directly controlled by *unique* meta-parameters (vector $v_{\cdot b}$), thereby encouraging the black-box meta-learner to construct specific update rules for each element of the policy state. This results in sensitivity to permutations in inputs, outputs, and a general tendency to construct non-learning, biased solutions, over learning solutions. Furthermore, input and output spaces must retain the same size as those are directly dependent on the number of RNN parameters.

## 4 Adding Symmetries to Black-box Meta RL

A solution to the illustrated over-fitting problem with black-box methods is the introduction of symmetries into the parameterisation of the policy. This can be achieved by generalising the forward pass (Equation 1), backward pass (Equation 2), and element-wise update (Equation 3) to parameterized versions. We further subsume the loss computation into these parameterized update rules. Together, they form a single recurrent policy with additional symmetries. Prior work on variable shared meta learning (VSML) [12] used similar principles to meta-learn supervised learning algorithms. In the following, we extend their approach to deal with the RL setting.

**Variable Shared Meta Learning** VSML describes neural architectures for meta learning with parameter sharing. This can be motivated by meta learning how to update weights [4, 22] where the update rule is shared across the network. Instead of designing a meta network that defines the weight updates explicitly, we arrange small parameter-shared RNNs (LSTMs) like weights in a NN and perform message passing between those.

In VSML, each weight $w_{ab}$ with $w \in \mathbb{R}^{A \times B}$ in a NN is replaced by a small RNN with parameters $\theta$ and hidden state $h_{ab} \in \mathbb{R}^N$. We restrict ourselves to dense NN layers here, where $w$ corresponds to the weights of that layer with input size $A$ and output size $B$. This can be adapted to other architectures such as CNNs if necessary. All these RNNs share the same parameters $\theta$, defining both what information propagates in the neural network, as well as how states are updated to implement learning. Each RNN with state $h_{ab}$ receives the analogue to the previous activation, here called the vectorized forward message $\overrightarrow{m}_a \in \mathbb{R}^{\overrightarrow{M}}$, and the backward message $\overleftarrow{m}_b \in \mathbb{R}^{\overleftarrow{M}}$ for information flowing backwards in the network (asynchronously). The backward message may contain information relevant to credit assignment, but is not constrained to this. The RNN update equation (compare Equation 3 and 4) is then given by

$$h_{ab}^{(k)} \leftarrow f_{\text{RNN}}(h_{ab}^{(k)}, \overrightarrow{m}_a^{(k)}, \overleftarrow{m}_b^{(k)}) \tag{5}$$

for layer $k$ where $k \in \{1, \dots, K\}$ and $a \in \{1, \dots, A^{(k)}\}, b \in \{1, \dots, B^{(k)}\}$. Similarly, new forward messages are created by transforming the RNN states using a function $f_{\overrightarrow{m}} : \mathbb{R}^N \to \mathbb{R}^{\overrightarrow{M}}$ (compare Equation 1) such that

$$\overrightarrow{m}_b^{(k+1)} = \sum_a f_{\overrightarrow{m}}(h_{ab}^{(k)}) \tag{6}$$

defines the new forward message for layer $k+1$ with $b \in \{1, \dots, B^{(k)} = A^{(k+1)}\}$. The backward message is given by $f_{\overleftarrow{m}} : \mathbb{R}^N \to \mathbb{R}^{\overleftarrow{M}}$ (compare Equation 2) such that

$$\overleftarrow{m}_a^{(k-1)} = \sum_b f_{\overleftarrow{m}}(h_{ab}^{(k)}) \tag{7}$$

and $a \in \{1, \dots, A^{(k)} = B^{(k-1)}\}$. For simplicity, we use $\theta$ below to denote all of the VSML parameters, including those of the RNN and forward and backward message functions.

In the following, we derive a black-box meta reinforcement learner based on VSML (visualized in Figure 1).

**RL Agent Inputs and Outputs** At each time step in the environment, the agent's inputs consist of the previously taken action $a_{t-1}$, current observation $o_t$ and previous reward $r_{t-1}$. We feed $r_{t-1}$ as an additional input to each RNN, the observation $o_t \in \mathbb{R}^{A^{(1)}}$ to the first layer ($\overrightarrow{m}^{(1)}_{\cdot 1} := o_t$), and the action $a_{t-1} \in \{0,1\}^{B^{(K)}}$ (one-hot encoded) to the last layer ($\overleftarrow{m}^{(K)}_{\cdot 1} := a_{t-1}$). The index 1 refers to the first dimension of the $\overrightarrow{M}$ or $\overleftarrow{M}$-dimensional message. We interpret the agent's output message $y = \overrightarrow{m}^{(K+1)}_{\cdot 1}$ as the unnormalized logits of a categorical distribution over actions. While we focus on discrete actions only in our present experiments, this can be adapted for probabilistic or deterministic continuous control.

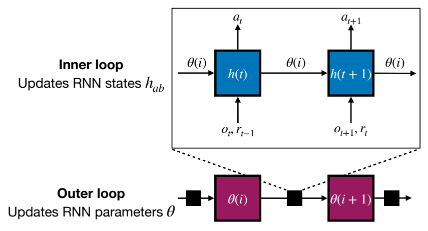

Figure 2: In SymLA, the inner loop recurrently updates all RNN states $h_{ab}(t)$ for agent steps $t \in \{1, \dots, L\}$ starting with randomly initialized states $h_{ab}$. Based on feedback $r_t$, RNN states can be used as memory for learning. The learning algorithm encoded in the RNN parameters $\theta$ is updated in the outer loop by meta-training using ES.

**Architecture Recurrence and Reward Signal** Instead of using multiple layers ($K > 1$), in this paper we use a single layer ($K = 1$). In Equation 5, RNNs in the same layer can not coordinate directly as their messages are only passed to the next and previous layer. To give that single layer sufficient expressivity for the RL setting, we make it 'recurrent' by processing the layer's own messages $\overrightarrow{m}^{(k+1)}_b$ and $\overleftarrow{m}^{(k-1)}_a$. The network thus has two levels of recurrence: (1) Each RNN that corresponds to a weight of a standard NN and (2) messages that are generated according to Equation 6 and 7 and fed back into the same layer. Furthermore, each RNN receives the current reward signal $r_{t-1}$ as input. The update equation is given by

$$h^{(k)}_{ab} \leftarrow f_{\text{RNN}}(h^{(k)}_{ab}, \underbrace{\overrightarrow{m}^{(k)}_a, \overleftarrow{m}^{(k)}_b, r_{t-1}}_{\text{environment inputs}}, \underbrace{\overrightarrow{m}^{(k+1)}_b, \overleftarrow{m}^{(k-1)}_a}_{\text{from previous step}}) \tag{8}$$

where $a \in \{1, \dots, A^{(k)}\}, b \in \{1, \dots, B^{(k)}\}$. As we only use a single layer, $k = 1$, we apply the update multiple times (multiple micro ticks) for each step in the environment. This can also be viewed as multiple layers with shared parameters, where parameters correspond to states $h$. For pseudo code, see Algorithm 1 in the appendix.

**Symmetries in SymLA** By incorporating the above changes to inputs, outputs, and architecture, we arrive at a black-box meta RL method with symmetries, here represented by our proposed *symmetric learning agents* (SymLA). By construction, SymLA exhibits the same symmetries as those described in Section 3, despite not using the backpropagation algorithm.

1. **Symmetric learning rule.** The learning rule as defined by Equation 8 is replicated across $a \in \{1, \dots, A\}$ and $b \in \{1, \dots, B\}$ with the same parameter $\theta$.
2. **Flexible input, output, and architecture sizes.** Changes in $A$, $B$, and $K$ correspond to input, output, and architecture size. This does not affect the number of meta-parameters and therefore these quantities can also be varied at meta-test time.
3. **Invariance to input and output permutations.** When permuting messages using bijections $\rho$ and $\rho'$, the state update becomes $h^{(k)}_{ab} \leftarrow f_{\text{RNN}}(h^{(k)}_{ab}, \overrightarrow{m}^{(k)}_{\rho(a)}, \overleftarrow{m}^{(k)}_{\rho'(b)}, r_{t-1}, \overrightarrow{m}^{(k+1)}_{\rho'(b)}, \overleftarrow{m}^{(k-1)}_{\rho(a)})$, and the message transformations are $\overrightarrow{m}^{(k+1)}_{\rho'(b)} = \sum_a f_{\overrightarrow{m}}(h^{(k)}_{ab})$ and $\overleftarrow{m}^{(k-1)}_{\rho(a)} = \sum_b f_{\overleftarrow{m}}(h^{(k)}_{ab})$. Similar to backpropagation, when RNN states $h_{ab}$ are initialized i.i.d., we can use $h_{\rho(a),\rho'(b)}$ in place of $h_{ab}$ to recover the original Equations 6, 7, 8.

**Learning / Inner Loop** Learning corresponds to updating RNN states $h_{ab}$ (see Figure 2). This is the same as the MetaRNN [25, 6] but with a more structured neural model. For fixed RNN parameters $\theta$ which encode the learning algorithm, we randomly initialize all states $h_{ab}$. Next, the agent steps through the environment, updating $h_{ab}$ in each step. If the environment is episodic with $T$ steps, the agent is run for a lifetime of $L \geq T$ steps with environment resets in-between, carrying the agent state $h_{ab}$ over.

**Meta Learning / Outer Loop** Each outer loop step unrolls the inner loop for $L$ environment steps to update $\theta$. The SymLA objective is to maximize the agent's lifetime sum of rewards, i.e. $\sum_{t=1}^{L} r_t(\theta)$. We optimize this objective using evolutionary strategies [26, 18] by following the gradient

$$\nabla_\theta \mathbb{E}_{\phi \sim \mathcal{N}(\phi|\theta,\Sigma)}[\mathbb{E}_{e \sim p(e)}[\sum_{t=1}^{L} r_t^{(e)}(\phi)]]. \tag{9}$$

with some fixed diagonal covariance matrix $\Sigma$ and environments $e \sim p(e)$. We chose evolution strategies due to its ability to optimize over long inner-loop horizons without memory constraints that occur due to backpropagation-based meta optimization. Furthermore, it was shown that meta-loss landscapes are difficult to navigate and the search distribution helps in smoothing those [14].

## 5 Experiments

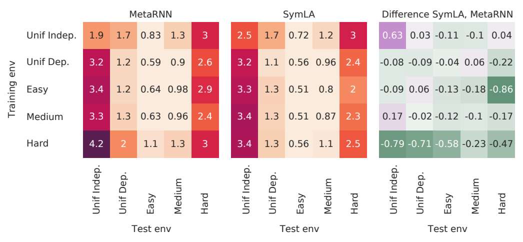

Figure 3: We compare SymLA to a standard MetaRNN on a set of bandit benchmarks from Wang et al. [25]. We train (y-axis) and test (x-axis) on two-armed bandits of varying difficulties. We report expected cumulative regret across 3 meta-training and 100 meta-testing runs with 100 arm-pulls (smaller is better). We observe that SymLA tends to perform comparably to the MetaRNN.

Equipped with a symmetric black-box learner, we now investigate how its learning properties differ from a standard MetaRNN. Firstly, we learn to learn on bandits from Wang et al. [25] where the meta-training environments are similar to the meta-test environments. Secondly, we demonstrate generalisation to unseen action spaces, applying the learned algorithm to bandits with varying numbers of arms at meta-test time—something that MetaRNNs are not capable of. Thirdly, we demonstrate how symmetries improve generalisation to unseen observation spaces by creating permutations of observations and actions in classic control benchmarks. Fourthly, we show how permutation invariance leads to generalisation to unseen tasks by learning about states and their associated rewards at meta-test time. Finally, we demonstrate how symmetries result in better learning algorithms for unseen environments, generalising from a grid world to CartPole. Hyper-parameters are in Appendix C.

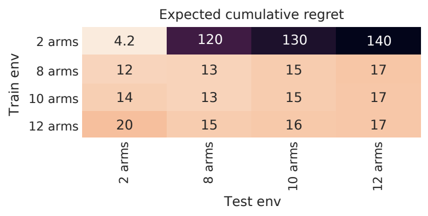

Figure 4: We meta-train and meta-test SymLA on varying numbers of independent arms to measure generalisation on unseen configurations. We do this by adding or removing RNNs to accommodate the varying output units. We report expected cumulative regret across 3 meta-training and 100 meta-testing runs with 100 arm-pulls (smaller is better). Particularly relevant are the out-of-distribution scenarios (off-diagonal).

**Learning to Learn on Similar Environments** We first compare SymLA and the MetaRNN on the two-armed (dependent) bandit experiments from Wang et al. [25] where there is no large variation in the meta-test environments. These consist of five different settings of varying difficulty that we use for meta-training and meta-testing (see Appendix B). There are no observations (no context), only two arms, and a meta-training distribution where each arm has the same marginal distribution of payouts. Thus, we expect the symmetries from SymLA to have no significant effect on performance . We meta-train for an agent lifetime of $L = 100$ arm-pulls and report the expected cumulative regret at meta-test time in Figure 3. We meta-train on each of the five settings, and meta-test across all settings. The performance of the MetaRNN reproduces the average performance of Wang et al. [25], here trained with ES instead of A2C. When using symmetries (as in SymLA), we recover a similar performance compared to the MetaRNN.

**Generalisation to Unseen Action Spaces** In contrast to the MetaRNN, in SymLA we can vary the number of arms at meta-test time. The architecture of SymLA allows to change the network size arbitrarily by replicating existing RNNs, thus adding or removing arms at meta-test time while retaining the same meta-parameters from meta-training. In Figure 4 we train on different numbers of arms and test on seen and unseen configurations. All arms are independently drawn from the uniform distribution $p_i \sim U[0, 1]$. We observe that SymLA works well within-distribution (diagonal) and generalises to unseen numbers of arms (off-diagonal). We also observe that for two arms a more specialized solution can be discovered, impeding generalisation when only training on this configuration.

**Generalisation to Unseen Observation Spaces** In the next experiments we want to specifically analyze the permutation invariance created by our architecture. In the previous bandit environments, actions occurred in all permutations in the training distribution. In contrast, RL environments usually have some structure to their observations and actions. For example in CartPole the first observation is usually the pole angle and the first action describes moving to the left. Human-engineered learning algorithms are usually invariant to permutations and thus generalise to new problems with different structure. The same should apply for our black-box agent with symmetries.

We demonstrate this property in the classic control tasks *CartPole*, *Acrobot*, and *MountainCar*. We meta-train on each environment respectively with the original observation and action order. We then meta-test on either (1) the same configuration or (2) across a permuted version. The results are visualized in Figure 5. Due to the built-in symmetries, the performance does not degrade in the shuffled setting. Instead, our method quickly learns about the ordering of the relevant observations and actions at meta-test time. In comparison, the MetaRNN baseline fails on the permuted setting where it was not trained on, indicating over-specialization. Thus, symmetries help to generalise to observation permutations that were not encountered during meta training.

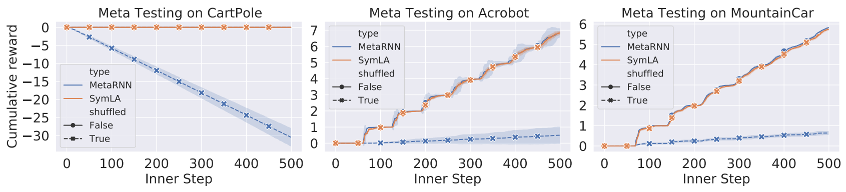

Figure 5: SymLA's architecture is inherently permutation invariant. When meta-training on standard CartPole, Acrobot, and MountainCar, the performance of the MetaRNN and SymLA are comparable. We then meta-test on a setting where both the observations and actions are shuffled. In this setting SymLA still performs well as it has meta-learned to identify observations and actions at meta-test time. In contrast, the MetaRNN fails to do so. Standard deviations are over 3 meta-training and 100 meta-testing runs.

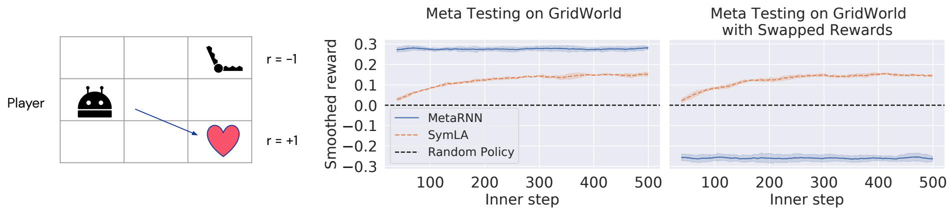

Figure 6: We extend the permutation invariant property to concepts - varying the rewards associated with different object types (+1 and -1) in a grid world environment (left). SymLA is forced to learn about the rewards of object types at meta-test time (starting at near zero reward and increasing the reward intake over time). When switching the rewards and running the same learner, the MetaRNN collects the wrong rewards, whereas SymLA still infers the correct relationships. Standard deviations are over 3 meta-training and 100 meta-testing runs.

**Generalisation to Unseen Tasks** The permutation invariance has further reaching consequences. It extends to learning about tasks at meta-test time. This enables generalisation to unseen tasks. We construct a grid world environment (see Figure 6) with two object types: A trap and a heart. The agent and the two objects (one of each type) are randomly positioned every episode. Collecting the heart gives a reward of +1, whereas the trap gives -1. All other rewards are zero. The agent observes its own position and the position of both objects. The observation is constructed as an image with binary channels for the position and each object type.

When meta-training on this environment, at meta-test time we observe in Figure 6 that the MetaRNN learns to directly collect hearts in each episode throughout its lifetime. This is due to having overfitted to the association of hearts with positive rewards. In comparison, SymLA starts with near-zero rewards and learns through interactions which actions need to be taken when receiving particular observations to collect the heart instead of the trap. With sufficient environment interactions $L$ we would expect SymLA to eventually (after sufficient learning) match the average reward per time of the MetaRNN in the non-shuffled grid world. Next, we swap the rewards of the trap and heart, i.e. the trap now gives a positive reward, whereas the heart gives a negative reward. This is equivalent to swapping the input channels corresponding to the heart and trap. We observe that SymLA still generalises, learning at meta-test time about observations and their associated rewards. In contrast, the MetaRNN now collects the wrong item, receiving negative rewards. These results show that black-box meta RL with symmetries discovers a more general update rule that is less specific to the training tasks than typical MetaRNNs.

**Generalisation to Unseen Environments** We have demonstrated how permutation invariance can lead to increased generalisation. But can SymLA also generalise between entirely different environments? We show-case how meta-training on a grid world environment allows generalisation to CartPole. To simplify credit-assignment, we use a dense-reward grid world where the reward is proportional to the change in distance toward a target position. Both the target position, as well as the agent position are randomized. The agent observes its own position, all obstacles, and the target position as a binary image with multiple channels. In the CartPole environment the agent is rewarded for being as upright and centered as possible [24]. Further, during meta-training, we randomly project observations linearly for each lifetime. This is necessary as in the grid world environment all observations are binary whereas the CartPole environment has continuously varying observations. This mismatch would inhibit generalisation. In Figure 7 we demonstrate that meta-training with SymLA only on the GridWorld environment allows reusing the same meta-learned learning algorithm to the CartPole environment. In contrast, the MetaRNN does not exhibit such generalisation. Thus, learning algorithms with symmetries can to some extent generalise between significantly different environments.

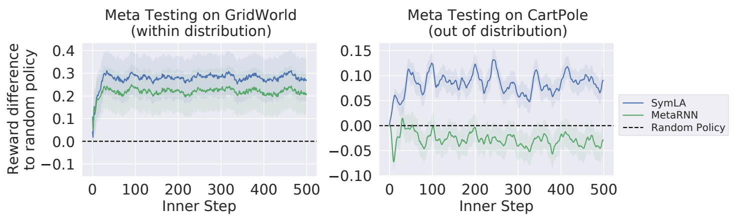

Figure 7: Generalisation capabilities of SymLA from GridWorld to CartPole. We meta-train the learning algorithm on GridWorld. We then meta-test on GridWorld and CartPole and report standard error of the mean and mean rewards (100 seeds) relative to a random policy - this highlights the learning process. While SymLA generalises from GridWorld to CartPole, the MetaRNN does not.

## 6 Conclusion

In this work, we identified symmetries that exist in backpropagation-based methods for meta RL but are missing from black-box methods. We hypothesized that these symmetries lead to better generalisation of the resulting learning algorithms. To test this, we extended a black-box meta learning method [12] that exhibits these same symmetries to the meta RL setting. This resulted in SymLA, a flexible black-box meta RL algorithm that is less prone to over-fitting. We demonstrated generalisation to varying numbers of arms in bandit experiments (unseen action spaces), permuted observations and actions with no degradation in performance (unseen observation spaces), and observed the tendency of the meta-learned RL algorithm to learn about states and their associated rewards at meta-test time (unseen tasks). Finally, we showed that the discovered learning behavior also transfers between grid world and (unseen) classic control environments.

## References

[1] F. Alet, M. F. Schneider, T. Lozano-Perez, and L. P. Kaelbling. Meta-learning curiosity algorithms. *arXiv preprint arXiv:2003.05325*, 2020.

[2] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[3] S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[4] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2, 1992.

[5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[6] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl$^2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[8] S. Flennerhag, A. A. Rusu, R. Pascanu, F. Visin, H. Yin, and R. Hadsell. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

[9] K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron. *IEICE Technical Report, A*, 62(10):658–665, 1979.

[10] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[11] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[12] L. Kirsch and J. Schmidhuber. Meta learning backpropagation and improving it. *arXiv preprint arXiv:2012.14905*, 2020.

[13] L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[14] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565. PMLR, 2019.

[15] T. Miconi, K. Stanley, and J. Clune. Differentiable plasticity: training plastic neural networks with backpropagation. In *International Conference on Machine Learning*, pages 3559–3568. PMLR, 2018.

[16] E. Najarro and S. Risi. Meta-learning through hebbian plasticity in random networks. *arXiv preprint arXiv:2007.02686*, 2020.

[17] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

[18] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[19] I. Schlag, T. Munkhdalai, and J. Schmidhuber. Learning associative inference using fast weight memory. *arXiv preprint arXiv:2011.07831*, 2020.

[20] J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[21] J. Schmidhuber. Steps towards self-referential neural learning: A thought experiment. 1992.

[22] J. Schmidhuber. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets. In *International Conference on Artificial Neural Networks*, pages 460–463. Springer, 1993.

[23] R. S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*, pages 171–176. San Jose, CA, 1992.

[24] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dm-control: Software and tasks for continuous control, 2020.

[25] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[26] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE, 2008.

[27] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[28] Z. Xu, H. P. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[29] Z. Xu, H. P. van Hasselt, M. Hessel, J. Oh, S. Singh, and D. Silver. Meta-gradient reinforcement learning with an objective discovered online. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15254–15264. Curran Associates, Inc., 2020.

[30] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017.

Table 1: A comparison between fixed reinforcement learning algorithms (REINFORCE), backpropagation-based meta RL (MAML, MetaGenRL, LPG), black-box (MetaRNN), and our black-box method with symmetries (SymLA). $\pi_\theta^{(s)}$ denotes a *stationary* policy that is updated at fixed intervals by backpropagation.

| | REINFORCE | MetaGenRL / LPG | MAML | MetaRNN | SymLA (ours) |
|---|---|---|---|---|---|
| Meta variables | / | $\phi$ | Initial $\theta_0$ | $\theta$ | $\theta$ |
| Learned variables | $\theta$ | $\theta$ | $\theta$ | RNN state $h$ | RNN states $h_{ab}^{(k)}$ |
| Learning algorithm | fixed loss func $L$ + Backprop | learned loss func $L_\phi$ + Backprop | fixed loss func $L$ + Backprop | $\pi_\theta$ | $\pi_\theta$ |
| Policy | $\pi_\theta^{(s)}$ | $\pi_\theta^{(s)}$ | $\pi_\theta^{(s)}$ | $\pi_\theta$ | $\pi_\theta$ |
| Black-box | ✗ | ✗ | ✗ | ✓ | ✓ |
| Symmetries in learning algorithm | ✓ | ✓ | ✓ | ✗ | ✓ |

## A Related Work

Learning to reinforcement learn can be implemented with varying degrees of inductive biases.

**Black-Box Meta RL** Black-box meta RL can be implemented by policies that receive the reward signal as input [21] and use memory to learn, such as recurrence in RNNs [10, 25, 6]. These approaches do not feature the symmetries discussed in this paper which leads to a tendency of overfitting.

**Learned Learning Rules & Fast Weights** In the supervised and reinforcement learning contexts, learned learning rules [4] or fast weights [20, 22, 15, 19, 16] describe (meta-)learned mechanisms (slow weights) that update fast weights to implement learning. This often involves outer-products and can be generalised to black-box meta learning with parameter sharing [12]. None of these approaches feature all of the symmetries we discuss above to meta learn RL algorithms.

**Backpropagation-based Meta RL** Alternatives to black-box meta RL include learning a weight initialization and adapting it with a human-engineered RL algorithm [7], warping computed gradients [8], meta-learning hyper-parameters [23, 28] or meta-learning objective functions corresponding to the learning algorithm [11, 13, 29, 17, 3].

**Neural Network Symmetries** Symmetries in neural networks have mainly been investigated to reflect the structure of the input data. This includes applications of convolutions [9], deep sets [30], graph neural networks [27], and geometric deep learning [5]. While many meta learning algorithms exhibit symmetries [4], in particular backpropagation-based meta learning [2, 7, 8, 13], the effects of these symmetries have not been discussed in detail. In this work, we provide such a discussion and experimental investigation in the context of meta RL.

**Algorithm 1** SymLA meta training

```
Require: Distribution over RL environment(s) $p(e)$
  $\theta \leftarrow$ initialize LSTM parameters
  while meta loss has not converged do    ▷ Outer loop in parallel over envs $e \sim p(e)$ and samples $\phi \sim \mathbb{N}(\phi|\theta, \Sigma)$
      $\{h_{ab}\} \leftarrow$ initialize LSTM states   $\forall a, b$
      $o_1 \sim p(o_1)$                                        ▷ Initialize environment $e$
      for $t \in \{1, \dots, L\}$ do                           ▷ Inner loop over lifetime in environment $e$
          $h_{ab} \leftarrow f_{\text{LSTM}}(h_{ab}, o_{t,a}, a_{t-1,b}, r_{t-1}, \overrightarrow{m}_b, \overleftarrow{m}_a)$   $\forall a, b$   ▷ Equation 8
          $\overrightarrow{m}_b \leftarrow \sum_a f_{\overrightarrow{m}}(h_{ab})$   $\forall b$        ▷ Create forward messages
          $\overleftarrow{m}_a \leftarrow \sum_b f_{\overleftarrow{m}}(h_{ab})$   $\forall a$        ▷ Create backward messages
          $y \leftarrow \overrightarrow{m}_{\cdot 1}$                                   ▷ Read out action
          $a_t \sim p(a_t; y)$                  ▷ Sample action from distribution parameterized by $y$
          Send action $a_t$ to environment $e$, observe $o_{t+1}$ and $r_t$
      $\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{\phi \sim \mathbb{N}(\phi|\theta,\Sigma)}[\mathbb{E}_{e \sim p(e)}[\sum_{t=1}^{L} r_t^{(e)}(\phi)]]$   ▷ Update $\theta$ using evolution strategies (Equation 9)
```

## B Bandits from Wang et al. [25]

In our experiments, we use bandits of varying difficulty from Wang et al. [25]. Let $p_1$ be the probability of the first arm for a payout of $r = 1$, $r = 0$ otherwise, and $p_2$ the payout for the second arm. Then, we define the

- uniform independent bandit with $p_1 \sim U[0, 1]$ and $p_2 \sim U[0, 1]$,
- uniform dependent bandit with $p_1 \sim U[0, 1]$ and $p_2 = 1 - p_1$,
- easy dependent bandit with $p_1 \sim U\{0.1, 0.9\}$ and $p_2 = 1 - p_1$,
- medium dependent bandit with $p_1 \sim U\{0.25, 0.75\}$ and $p_2 = 1 - p_1$,
- hard dependent bandit with $p_1 \sim U\{0.4, 0.6\}$ and $p_2 = 1 - p_1$.

## C Hyper-parameters

### C.1 SymLA Architecture

We use a single recurrent layer, $K = 1$, with a message size of $\overleftarrow{M} = 8$ and $\overrightarrow{M} = 8$. To produce the next state $h_{ab}$ according to Equation 8, we use parameter-shared LSTMs with a hidden size of $N = 16$ ($N = 64$ for bandits to match Wang et al. [25]) and run the recurrent cell for 2 micro ticks.

### C.2 Meta Learning / Outer Loop

We estimate gradients $\nabla_\theta$ using evolutionary strategies [18] with $10$ evaluations per population sample to estimate the fitness value (100 evaluations for bandits). Then, we apply those using Adam with a learning rate of $\alpha = 0.01$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ ($\alpha = 0.2$ for bandits). We use a fixed noise standard deviation of $\sigma = 0.035$ ($\sigma = 0.2$ for bandits) and a population size of $512$. Our inner loop has a length of $L = 500$ ($L = 100$ for bandits), concatenating multiple episodes. We meta-optimize for $4,000$ outer steps for bandit experiments, and $20,000$ otherwise.

### C.3 Generalisation to Unseen Environments

We apply a random linear transformation (Glorot normal) to environment observations, mapping those to a 16-dimensional vector.