# MODULAR CONTINUAL LEARNING IN A UNIFIED VISUAL ENVIRONMENT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A core aspect of human intelligence is the ability to learn new tasks quickly and switch between them flexibly. Here, we describe a modular continual reinforcement learning paradigm inspired by these abilities. We first introduce a visual interaction environment that allows many types of tasks to be unified in a single framework. We then describe a reward map prediction scheme that learns new tasks robustly in the very large state and action spaces required by such an environment. We investigate how properties of module architecture influence efficiency of task learning, showing that a module motif incorporating specific design principles (e.g. early bottlenecks, low-order polynomial nonlinearities, and symmetry) significantly outperforms more standard neural network motifs, needing fewer training examples and fewer neurons to achieve high levels of performance. Finally, we present a meta-controller architecture for task switching based on a recurrent neural voting scheme, which allows new modules to use information learned from previously-seen tasks to substantially improve their own learning efficiency.

## INTRODUCTION

In the course of everyday functioning, people are constantly faced with real-world environments in which they are required to shift unpredictably between multiple, sometimes unfamiliar, tasks (Botvinick & Cohen, 2014). They are nonetheless able to flexibly adapt existing decision schemas or build new ones in response to these challenges (Arbib, 1992). How humans support such flexible learning and task switching is largely unknown, both neuroscientifically and algorithmically (Wagner et al., 1998; Shadlen & Newsome, 2001).

We investigate solving this problem with a *neural module* approach in which simple, task-specialized decision modules are dynamically allocated on top of a largely-fixed underlying sensory system (Andreas et al., 2015; Hu et al., 2017). The sensory system computes a general-purpose visual representation from which the decision modules read. While this sensory backbone can be large, complex, and learned comparatively slowly with significant amounts of training data, the task modules that deploy information from the base representation must, in contrast, be lightweight, quick to be learned, and easy to switch between. In the case of visually-driven tasks, results from neuroscience and computer vision suggest the role of the fixed general purpose visual representation may be played by the ventral visual stream, modeled as a deep convolutional neural network (Yamins & DiCarlo, 2016; Razavian et al., 2014). However, the algorithmic basis for how to efficiently learn and dynamically deploy visual decision modules remains far from obvious.

In standard supervised learning, it is often assumed that the output space of a problem is prespecified in a manner that just happens to fit the task at hand — e.g. for a classification task, a discrete output with a fixed number of classes might be determined ahead of time, while for a continuous estimation problem, a one-dimensional real-valued target might be chosen instead. This is a very convenient simplification in supervised learning or single-task reinforcement learning contexts, but if one is interested in the learning and deployment of decision structures in a rich environment defining tasks with many different natural output types, this simplification becomes cumbersome.

To go beyond this limitation, we build a *unified environment* in which many different tasks are naturally embodied. Specifically, we model an agent interacting with a two-dimensional touchscreen-like GUI that we call the *TouchStream*, in which all tasks (discrete categorization tasks, continuous estimation problems, and many other combinations and variants thereof) can be encoded using a

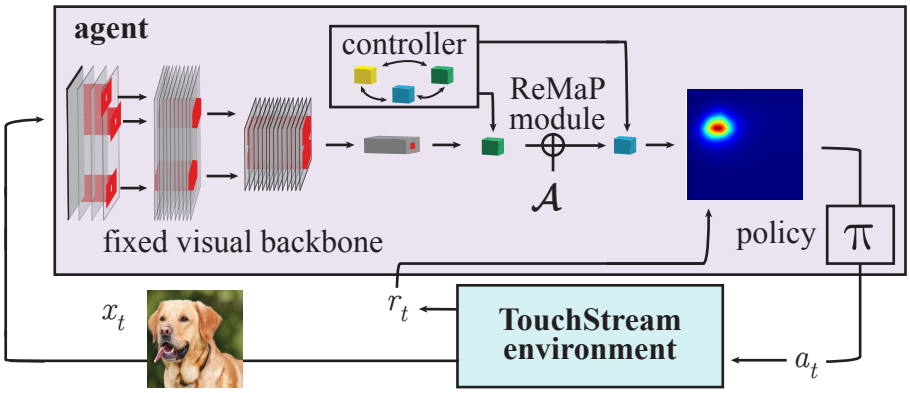



Figure 1: **Modular continual learning in the TouchStream environment** The TouchStream is a GUI-like environment for continual learning agents, used for posing visual reasoning tasks in a large but unified action space. The agent for this work is a series of interdependent neural networks, consisting of a fixed visual backbone (e.g. a deep convolutional neural network), a set of learned neural modules, and a recurrent meta-controller which mediates the deployment of these learned modules for task solving. The modules use the ReMaP algorithm to produce an estimate anticipated rewards for a finite future horizon, conditional on the agent's recent history, over the entire action space. Using a sampling policy on this reward map, the agent chooses an optimal action to maximize its aggregate reward.

single common and intuitive — albeit large — output space. This choice frees us from having to hand-design or programmatically choose between different output domain spaces, but forces us to confront the core challenge of how a naive agent can quickly and emergently learn the implicit "interfaces" required to solve different tasks.

We then introduce Reward Map Prediction (ReMaP) networks, an algorithm for continual reinforcement learning that is able to discover implicit task-specific interfaces in large action spaces like those of the TouchStream environment. We address two major algorithmic challenges associated with learning ReMaP modules. First, what module architectural motifs allow for *efficient* task interface learning? We compare several candidate architectures and show that those incorporating certain intuitive design principles (e.g. early visual bottlenecks, low-order polynomial nonlinearities and symmetry-inducing concatenations) significantly outperform more standard neural network motifs, needing fewer training examples and fewer neurons to achieve high levels of performance. Second, what system architectures are effective for *switching* between tasks? We present a meta-controller architecture based on a recurrent neural voting scheme, allowing new modules to use information learned from previously-seen tasks to substantially improve their own learning efficiency.

In § 1 we formalize the Touchstream environment. In § 2, we introduce the ReMaP algorithm. In § 3, we describe and evaluate comparative performance of multiple ReMaP module architectures on a variety of Touchstream tasks. In § 4, we describe the Recurrent Neural Voting meta-controller, and evaluate its ability to efficiently transfer knowledge between ReMaP modules on task switches.

### Related Work

Modern deep convolutional neural networks have had significant impact on computer vision and artificial intelligence (Krizhevsky et al. (2012)), as well as in the computational neuroscience of vision (Yamins & DiCarlo (2016)). There is a recent but growing literature on convnet-based *neural modules*, where they have been used for solving compositional visual reasoning tasks (Andreas et al., 2015; Hu et al., 2017). In this work we apply the idea of modules to solving visual learning challenges in a continual learning context. Existing works rely on choosing between a menu of pre-specified module primitives, using different module types to solve subproblems involving specific input-output datatypes, without addressing *how* these modules' forms are to be discovered in the first place. In this paper, we show a single generic module architecture is capable of automatically learning to solve a wide variety of different tasks in a unified action/state space, and a simple controller scheme is able to switch between such modules.

Our results are also closely connected with the literature on lifelong (or continual) learning (Kirkpatrick et al., 2016; Rusu et al., 2016). A part of this literature is concerned with learning to solve new

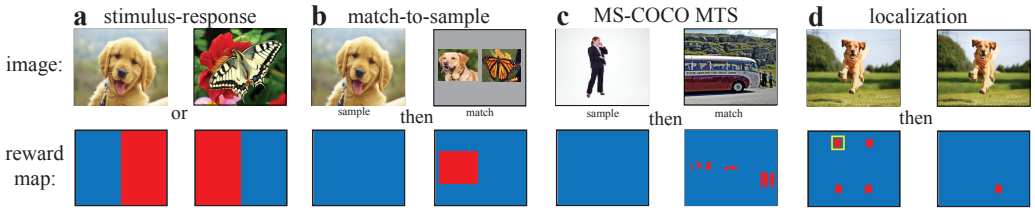



Figure 2: **Exemplar TouchStream tasks.** Illustration of several task paradigms explored in this work using the TouchStream Environment. The top row depicts observation $x_t$ and the bottom shows the ground truth reward maps (with red indicating high reward and blue indicating low reward). **a.** Binary Stimulus-Response task. **b.** stereotyped Match-To-Sample task. **c.** The Match-To-Sample task using the MS-COCO dataset. **d.** Object localization.

tasks without catastrophically forgetting how to solve old ones (Zenke et al., 2017; Kirkpatrick et al., 2016). The use of modules obviates this problem, but instead shifts the hard question to one of how newly-allocated modules can be learned effectively. The continual learning literature also directly addresses knowlege transfer to newly allocated structures (Chen et al., 2015; Rusu et al., 2016; Fernando et al., 2017), but largely addresses how transfer learning can lead to higher performance, rather than addressing how it can improve learning speed. Aside from reward performance, we focus on issues of speed in learning and task switching, motivated by the remarkably efficient adaptability of humans in new task contexts. Existing work in continual learning also largely does not address which specific architecture types learn tasks efficiently, independent of transfer. By focusing first on identifying architectures that achieve high performance quickly on individual tasks (§ 3), our transfer-learning investigation then naturally focuses more on how to efficiently identify when and how to re-use components of these architectures (§ 4). Most of these works also make explicit *a priori* assumptions about the structure of the tasks to be encoded into the models (e.g. output type, number of classes), rather than address the more general question of emergence of solutions in an embodied case, as we do.

Meta-reinforcement learning approaches such as Wang et al. (2016); Duan et al. (2016), as well as the schema learning ideas of e.g. Arbib (1992); McClelland (2013) typically seek to address the issue of continual learning by having a complex meta-learner extract correlations between tasks over a long timescale. In our context most of the burden of environment learning is placed on the individual modules, so our meta-controller can thus be comparatively light-weight compared to typical meta-reinforcement approaches. Unlike our case, meta-learning has mostly been limited to small state or action spaces. Some recent work in general reinforcement learning (e.g. Ostrovski et al. (2017); Dulac-Arnold et al. (2015)) has addressed the issue of large action spaces, but has not sought to address multitask transfer learning in these large action spaces.

## 1 THE TOUCHSTREAM ENVIRONMENT

Agents in a real-world environment are exposed to many different implicit tasks, arising without predefined decision structures, and must learn *on the fly* what the appropriate decision interfaces are for each situation. Because we are interested in modeling how agents can do this on-the-fly learning, our task environment should mimic the unconstrained nature of the real world. Here, we describe the TouchStream environment, which attempts to do this in a simplified two-dimensional domain.

Our problem setup consists of two components, an "environment" and an "agent," interacting over an extended temporal sequence (Fig. 1). At each timestep $t$, the environment emits an RGB image $x_t$ of height $H$ and width $W$, and a scalar reward $r_t$. Conversely, the agent accepts images and rewards as input and chooses an action $a_t$ in response. The action space $\mathcal{A}$ available to the agent consists of a two-dimensional pixel grid $\{0, \ldots, H-1\} \times \{0, \ldots, W-1\} \subset \mathbb{Z}^2$, of the same height and width as its input image. The environment is equipped with a policy (unknown to the agent) that on each time step computes image $x_t$ and reward $r_t$ as a function of the history of agent actions $\{a_0, \ldots, a_{t-1}\}$, images $\{x_0, \ldots, x_{t-1}\}$ and rewards $\{r_0, \ldots, r_{t-1}\}$. In this work, the agent is a neural network, composed of a visual backbone with fixed weights, together with a recurrent controller module whose parameters are learned by interaction with the environment. The agent's goal is to learn to enact a policy that maximizes its reward obtained over time.

By framing the action space $\mathcal{A}$ of the agent as all possible pixel locations and the state space as any arbitrary image, a very wide range of possible tasks are unified in this single framework, at the cost of requiring the agents' action space to be congruent to its input state space, and thus be quite large. This presents two core efficiency challenges for the agent: on any given task, it must be able to both quickly recognize what the "interface" for the task is, and *transfer* such knowledge across tasks in a smart way. Both of these goals are complicated by the fact that both the large size of agent's state and action spaces.

Although we work with modern large-scale computer vision-style datasets and tasks in this work, e.g. ImageNet (Deng et al. (2009)) and MS-COCO (Lin et al. (2014)), we are also inspired by visual psychology and neuroscience, which have pioneered techniques for how controlled visual tasks can be embodied in real reinforcement learning paradigms (Horner et al., 2013; Rajalingham et al., 2015). Especially useful are three classes of task paradigms that span a range of the ways discrete and continuous estimation tasks can be formulated — including Stimulus-Response, Match-To-Sample, and Localization tasks (Fig. 2).

**Stimulus-Response Tasks:** The Stimulus-Response (SR) paradigm is a common approach to physically embodying discrete categorization tasks (Gaffan & Harrison, 1988). For example, in the simple two-way SR discrimination task shown in Fig. 2a, the agent is rewarded if it touches the left half of the screen after being shown an image of a dog, and the right half after being shown a butterfly. SR tasks can be made more difficult by increasing the number of image classes or the complexity of the reward boundary regions. In our SR experiments, we use images and classes from the ImageNet dataset (Deng et al., 2009).

**Match-To-Sample Tasks:** The Match-to-Sample (MTS) paradigm is another common approach to assessing visual categorization abilities (Murray & Mishkin, 1998). In the MTS task shown in Fig. 2b, trials consist of a sequence of two image frames — the "sample" screen followed by the "match" screen — in which the agent is expected to remember the object category seen on the sample frame, and then select an onscreen "button" (really, a patch of pixels) on the match screen corresponding to the sample screen category. Unlike SR tasks, MTS tasks require some working memory and more localized spatial control. More complex MTS tasks involve more sophisticated relationships between the sample and match screen. In Fig. 2c, using the MS-COCO object detection challenge dataset (Lin et al., 2014), the sample screen shows an isolated template image indicating one of the 80 MS-COCO classes, while the match screen shows a randomly-drawn scene from the dataset containing at least one instance of the sample-image class. The agent is rewarded if its chosen action is located inside the boundary of an instance (e.g. the agent "pokes inside") of the correct class. This MS-COCO MTS task is a hybrid of categorical and continuous tasks.

**Localization:** Fig. 2d shows a two-step continuous localization task in which the agent is supposed to mark out the bounding box of an object by touching opposite corners on two successive timesteps, with reward proportionate to the Intersection over Union (IoU) value of the predicted bounding box relative to the ground truth bounding box $IoU = \frac{Area(B_{GT} \cap \hat{B})}{Area(B_{GT} \cup \hat{B})}$. In localization, unlike the SR and MTS paradigms, the choice made at one timestep constrains the agent's optimal choice on a future timestep (e.g. picking the upper left corner of the bounding box on the first step contrains the lower right opposite corner to be chosen on the second).

Although these tasks can become arbitrarily complex along certain axes, the tasks presented here require only fixed-length memory and future prediction. That is, each task requires only knowledge of the past $k_b$ timesteps, and a perfect solution always exists within $k_f$ timesteps from any point. The minimal required values of $k_b$ and $k_f$ are different across the various tasks in this work. However, in the investigations below, we take $k_b = 1$ and $k_f = 2$ – the maximum required values across these tasks – and thus require the agent to learn for itself when it is safe to ignore information from the past and when it is irrelevant to predict past a certain point in the future.

We will begin by considering a restricted case where the environment runs one semantic task indefinitely, showing how different architectures learn to solve such individual tasks with dramatically different levels of efficiency (§ 2-3). We will then expand to considering the case where the environment's policy consists of a sequence of tasks with unpredictable transitions between tasks, and exhibit a meta-controller that can cope effectively with this expanded domain (§ 4).

## 2 REWARD MAP PREDICTION

The TouchStream environment necessarily involves working with large action and state spaces. Methods for handling this situation often focus on reducing the effective size of action/state spaces, either via estimating pseudo-counts of state-action pairs, or by clustering actions (Ostrovski et al., 2017; Dulac-Arnold et al., 2015). Here we take another approach, using a neural network to directly approximate the (image-state modulated) mapping between the action space and reward space, allowing learnable regularities in the state-action interaction to implicitly reduce the large spaces into something manageable by simple choice policies. We introduce an off-policy algorithm for efficient multitask reinforcement learning in large action and state spaces: Reward Map Prediction, or ReMaP.

### 2.1 REMAP NETWORK ALGORITHM

As with any standard reinforcement learning situation, the agent seeks to learn an optimal policy $\pi = p(a_t \mid x_t)$ defining the probability density $p$ over actions given image state $x_t$. The ReMaP algorithm is off-policy, in that $\pi$ is calculated as a simple fixed function of the estimated reward.

A *ReMaP network* $M_\Theta$ is a neural network with parameters $\Theta$, whose inputs are a history over previous timesteps of (i) the agent's own actions, and (ii) an activation encoding of the agent's state space; and which explicitly approximates the expected reward map across its action space for some number of future timesteps. Mathematically:

$$M_\Theta : [\mathbf{\Psi}_{t-k_b:t}, \mathbf{h}_{t-k_b:t-1}] \longmapsto \left[m_t^1, m_t^2, \ldots, m_t^{k_f}\right]$$

where $k_b$ is the number of previous timesteps considered; $k_f$ is the length of future horizon to be considered; $\mathbf{\Psi}_{t-k_b:t}$ is the history $[\psi(x_{t-k_b}), \ldots, \psi(x_t)]$ of state space encodings produced by fixed backbone network $\psi(\cdot)$, $\mathbf{h}_{t-k_b:t-1}$ is the history $[a_{t-k_b} \ldots, a_{t-1}]$ of previously chosen actions, and each $m_i \in \mathbf{map}(\mathcal{A}, \mathcal{R})$ — that is, a map from action space to reward space. The predicted reward maps are constructed by computing the expected reward obtained for a subsample of actions drawn randomly from $\mathcal{A}$:

$$m_t^j : a_t \mapsto E\left[r_{t+j} \mid a_t, \mathbf{h}_{t-k_b:t-1}, \mathbf{\Psi}_{t-k_b:t}\right] = \int_{\mathcal{R}} r_{t+j} p(r_{t+j} \mid a_t, \mathbf{h}_{t-k_b:t-1}, \mathbf{\Psi}_{t-k_b:t}). \tag{1}$$

where $r_{t+j}$ is the predicted reward $j$ steps into the future horizon. Having produced $k_f$ reward prediction maps, one for each timestep of its future horizon, the agent needs to determine what it believes will be the single best action over all the expected reward maps $\left[m_t^1, m_t^2, \ldots, m_t^{k_f}\right]$. The ReMaP algorithm formulates doing so by normalizing the predictions across each of these $k_f$ maps into separate probability distributions, and sampling an action from the distribution which has maximum variance. That is, the agent computes its policy $\pi$ as follows:

$$\pi = \mathbf{VarArgmax}_{j=1}^{k_f}\{\mathbf{Dist}[\mathbf{Norm}[m_t^j]]\}, \tag{2}$$

where

$$Norm[m] = m - \min_{x \in \mathcal{A}} m(x) \tag{3}$$

is a normalization that removes the minimum of the map,

$$Dist[m] = \frac{f(m)}{\int_{\mathcal{A}} f(m(x))} \tag{4}$$

ensures it is a probability distribution parameterized by functional family $f(\cdot)$, and $\mathbf{VarArgmax}$ is an operator which chooses the input with largest variance. This procedure proposes two solutions to exploration of large action spaces containing large spatial and temporal non-uniformity. First, within each future predicted frame, we *sample* actions proportionate to the expected reward prediction, narrowing the spatial distribution more than would (e.g.) an $\epsilon$-greedy policy. Second, we combine across time frames by selecting actions from a single frame with maximum predicted reward variance, corresponding to the idea that an action at one of the timesteps in the horizon will impact the outcome of the remainder of the episode most. This will be the case if e.g. there is a high uncertainty associated with $Dist\,[\cdot]$ on this timestep (encouraging exploration), or that $Dist\,[\cdot]$ has several localized intervals

containing disproportionate probability mass (regions to be exploited). Although any standard action selection strategy can be used in place of the one in (2) (e.g. pseudo $\epsilon$-greedy over all $k_f$ maps), we have empirically found that this policy is effective at efficiently exploring our large action space.

The parameters $\Theta$ of a ReMaP network are learned by gradient descent on the loss of the reward prediction error $\Theta^* = \text{argmin}_\Theta L\left[m_t, r_t; \Theta\right]$ with map $m_t^j$ compared to the true reward $r_{t+j}$. Only the actual action chosen at timestep $t$ actually participates in loss calculation and backpropagation of error signals.

The ReMaP algorithm is summarized in 1.

**Algorithm 1: ReMaP – Reward Map Prediction**

Initialize ReMaP network $M$
Initialize state and action memory buffers $\mathbf{\Psi}_{t-k_b:t}$ and $\mathbf{h}_{t-k_b:t-1}$
**for** *timestep t = 1,T* **do**
  Observe $x_t$, encode with state space network $\psi(\cdot)$, and append to state buffer
  Subsample set of potential action choices $a_t$ uniformly from $\mathcal{A}$
  Produce $k_f$ expected reward maps of $a_t$ from eq. (1)
  Select action according to policy $\pi$ as in (2)
  Execute action $a_t$ in environment, store in action buffer, and receive reward $r_t$
  Calculate loss for this and previous $k_f - 1$ timesteps
  **if** $t \equiv 0 \mod$ *batch size* **then**
    Perform parameter update

Throughout this work, we take our fixed backbone state space encoder to be the VGG-16 convnet, pretrained on ImageNet (Simonyan & Zisserman, 2014). Because the resolution of the input to this network is 224x224 pixels, our action space $\mathcal{A} = \{0, \ldots, 223\} \times \{0, \ldots, 223\}$. By default, the functional family $f$ used in the action selection scheme in Eq. (4) is the identity, although on tasks benefiting from high action precision (e.g. Localization or MS-COCO MTS), it is often optimal to sample a low-temperature Boltzmann distribution with $f(x) = e^{-x/T}$.

## 3 Efficient Neural Modules for Task Learning

The main question we seek to address in this section is: what specific neural network structure(s) should be used in ReMaP modules? The key considerations are that such modules (i) should be easy to learn, requiring comparatively few training examples to discover optimal parameters $\Theta^*$, and (ii) easy to learn from, meaning that an agent can quickly build a new module by reusing components of old ones.

**Intuitive Example:** As an intuition-building example, consider the case of a simple binary Stimulus-Response task, as in Fig. 2a ("if you see a dog touch on the right, if a butterfly touch on the left"). One decision module that is a "perfect" reward predictor on this task is expressed analytically as:

$$M[\Psi](a_x, a_y) = \text{sgn}(\mathbf{ReLu}(W\mathbf{\Psi}) \cdot \mathbf{ReLu}(a_x) + \mathbf{ReLu}(-W\mathbf{\Psi}) \cdot \mathbf{ReLu}(-a_x)) \tag{5}$$

where $a_x$ is the $x$-component of the action $a \in \mathcal{A}$, and $W$ is a length-$|\mathbf{\Psi}|$ vector expressing the class boundary (bias term omitted for clarity). If $W\Psi$ is positive $a_x$ must also be positive to predict positive reward; conversly, if $W\Psi$ is negative, $a_x$ must be negative to predict reward.

Three basic principles are evident from the "perfect" formula:

- there is an **early visual bottleneck**, in which the high-dimensional general purpose feature representation $\Psi$ is greatly reduced in dimension (in this case, from the 4096 features of VGG's FC6 layer, to 1) prior to combination with action space,
- there is a **multiplicative interaction** between the action vector and (bottlenecked) visual features, and
- there is **symmetry**, e.g. the first term of the formula is the sign-antisymmetric partner of the second term, reflecting something about the spatial structure of the task.

But how can this one example be generalized into a parameterized structure from which the "right" visual bottleneck (the $W$ parameters), and decision structure (the *form* of equation (5) modified for the task at hand) can emerge naturally and efficienty via learning for any given task of interest?

### 3.1 THE EMS MODULE

In this section we define a generic ReMaP module which is lightweight, encodes all three generic design principles from the "perfect" formula, and uses only a small number of learnable parameters.

Define the *concatenated square* nonlinearity as

$$\mathbf{Sq} : x \longmapsto x \oplus x^2$$

and the *concatenated ReLu* nonlinearity (Shang et al. (2016)) as

$$\mathbf{CReLu} : x \longmapsto \mathbf{ReLu}(x) \oplus \mathbf{ReLu}(-x)$$

where $\oplus$ denotes vector concatenation. The $\mathbf{CReS}$ nonlinearity is then defined as the composition of $\mathbf{CReLu}$ and $\mathbf{Sq}$, e.g.

$$x \mapsto \mathbf{ReLu}(x) \oplus \mathbf{ReLu}(-x) \oplus \mathbf{ReLu}^2(x) \oplus \mathbf{ReLu}^2(-x) \coloneqq \mathbf{CReS}(x).$$

The $\mathbf{CReS}$ nonlinearity introduces multiplicative interactions between its arguments via its $\mathbf{Sq}$ component and symmetry via its use of $\mathbf{CReLu}$.

**Definition.** *The* $(n_0, n_1, \ldots, n_k)$-Early Bottleneck-Multiplicative-Symmetric (EMS) *module is the ReMaP module given by*

$$B = \mathbf{CReLu}(W_0 \cdot \Psi + b_0)$$
$$l_1 = \mathbf{CReS}(W_1(B \oplus a) + b_1)$$
$$l_i = \mathbf{CReS}(W_i l_{i-1} + b_i) \quad \textit{for} \quad i > 1$$

*where* $W_i$ *and* $b_i$ *are learnable parameters,* $\Psi$ *are features from the fixed visual encoding network, and* $a$ *is the action vector in* $\mathcal{A}$*.*

The EMS structure builds in each of the three principles described above. The $B$ stage represents the early bottleneck in which visual encoding inputs are bottlenecked to size $n_0$ before being combined with actions, and then performs $k$ $\mathbf{CReS}$ stages, introducing multiplicative symmetric interactions between visual features and actions. From this, the "perfect" module definition for the binary SR task in eq. (5) then becomes a special case of a two-layer EMS module. Note that the visual features to be bottlenecked can from any encoder; in practice, we work with both fully connected and convolutional features of the VGG-16 backbone.

In the experiments that follow, we compare the EMS module to a wide variety of alternative control motifs, in which the early bottleneck, multiplicative, and symmetric features are ablated. Multiplicative nonlinearity and bottleneck ablations use a spectrum of more standard activation functions, including $\mathbf{ReLu}$, $\tanh$, sigmoid, elu (Clevert et al., 2015), and $\mathbf{CReLu}$ forms. In late bottleneck (fully-ablated) architectures – which are, effectively, "standard" multi-layer perceptrons (MLPs) – action vectors are concatenated directly to the output of the visual encoder before being passed through subsequent stages. In all, we test 24 distinct architectures. Detailed information on each can be found in the Supplementary material.

### 3.2 EXPERIMENTS

We compared each architecture across 12 variants of visual SR, MTS, and localization tasks, using fixed visual encoding features from layer FC6 of VGG-16. Task variants ranged in complexity from simple (e.g. a binary SR task with ImageNet categories) to more challenging (e.g. a many-way ImageNet MTS task with result buttons appearing in varying positions on each trial). The most complex tasks are two variants of localization, either with a single main salient object placed on a complex background (similar to images used in Yamins & DiCarlo (2016)), or complex scenes from MS-COCO (see Fig. 3b). Details of the tasks used in these experiments can be found in the Supplementary material. Module weights were initialized using a normal distribution with $\mu = 0.0$, $\sigma = 0.01$, and optimized using the ADAM algorithm (Kingma & Ba (2014)) with parameters $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 1e-8$. Learning rates were optimized on a per-task, per-architecture

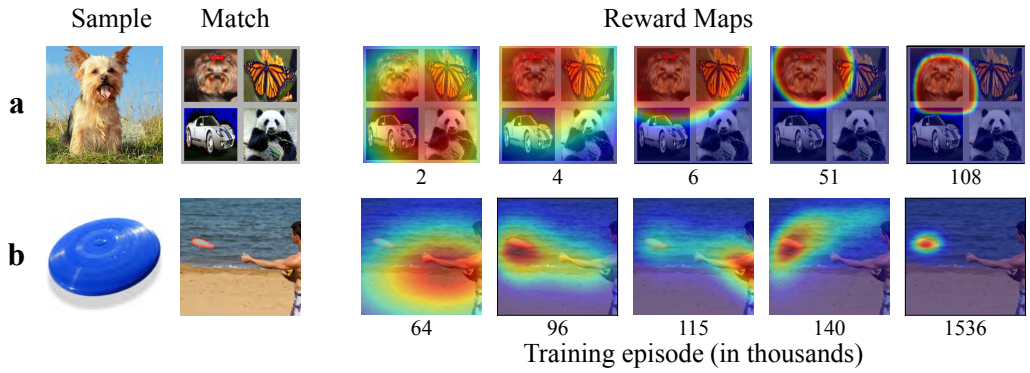



Figure 3: **Decision interfaces emerge naturally over the course of training.** The ReMaP modules capture the emergence of natural physical constructs as task decision interfaces over the course of learning. Examples of this are **a.** buttons on the match screen of a challenging MTS task and **b.** that object salience is related to the actual physical *shape* of objects inside a complex real-world scene. Best viewed in color.

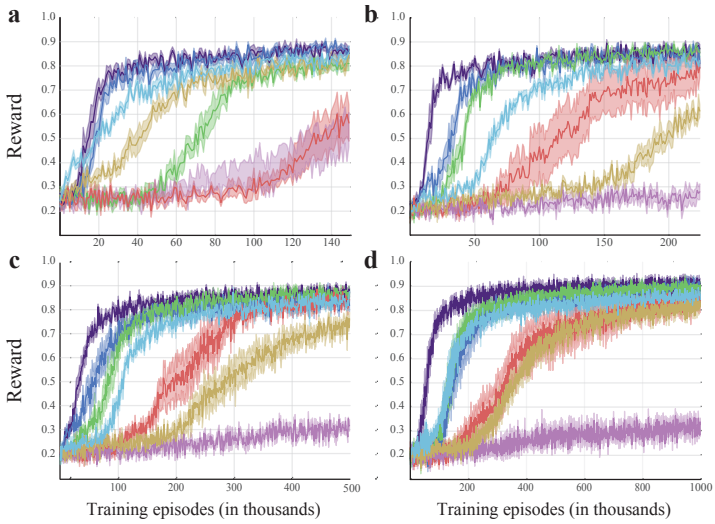
basis in a cross-validated fashion. For each architecture and task, we ran optimizations from five different initialization seeds to obtain mean and standard error due to initial condition variability. For fully-ablated "late-bottleneck" modules, we measured the performance of modules of three different sizes (small, medium, and large), where the smallest version is equivalent in size to the EMS module, and the medium and large versions are much larger (Table S1).

**Emergence of Decision Structures:** A key feature of ReMaP modules is that they are able to discover highly interpretable implicit decision structures across a variety of visual tasks starting from no knowledge (Fig. 3). In general we observe that the modules typically discover the underlying "physical structures" needed to operate the implicit task interface before learning the specific decision rules needed to solve the task. For example, in the case of a discrete MTS categorization task (Fig. 3a), this involves the quick discovery of onscreen "buttons" corresponding to discrete action choices before these buttons are mapped to their semantic meaning. In the case of hybrid category-coordinated continuous localization in MS-COCO images (Fig. 3b) this corresponds to the initial discovery of high salience object boundaries and then the refinement by category type. The specific temporal patterns of how the system discovers and refines its estimate of the implicit interfaces for each task are highly characteristic and replicable across initial seedings, and could serve as a strong candidate model of patterns of interface use and learning in humans.

**Efficiency of the EMS module:** The efficiency of learning was measured by computing the task-averaged, normalized area under the learning curve (TA-N-AUC) for each of the 24 modules tested, across all 12 task variants. Fig. 4a-d shows characteristic learning curves for several tasks, summarized in the table in Fig. 4e. Results for all architectures for all tasks are shown in Supplementary Figure S1. We find that the EMS module is the most efficient across tasks (0.997 TA-N-AUC). Moreover, the EMS architecture always achieves the highest final reward level on each task.

Increasing ablations of the EMS structure lead to increasingly poor performance, both in terms of learning efficiency and final performance. Ablating the low-order polynomial interaction (replacing $\mathbf{Sq}$ with $\mathbf{CReLu}$) had the largest negative effect on performance (0.818 TA-N-AUC), followed in importance by the symmetric structure (0.944 TA-N-AUC). Large fully-ablated models (no bottleneck, using only $\mathbf{ReLu}$ activations) performed significantly worse than the smaller EMS module and the single ablations (0.717 TA-N-AUC), but better than the module with neither symmetry nor multiplicative interactions (0.566 TA-N-AUC). Small fully-ablated modules with the same number of parameters as EMS were by far the least efficient (0.403 TA-N-AUC) and oftentimes achieved much lower final reward. In summary, the main conceptual features by which the special-case architecture in eq. (5) solves the binary SR task are both individually helpful, combine usefully, and can be parameterized and efficiently learned for a variety of visual tasks. These properties are critical to achieving effective task learning compared to standard MLP structures.

In a second experiment focusing on localization tasks, we tested an EMS module using convolutional features from the fixed VGG-16 feature encoder, reasoning that localization tasks could benefit from finer spatial feature resolution. We find that using visual features with explicit spatial information substantially improves task performance and learning efficiency on these tasks (Fig. 5). To our knowledge, our results on MS-COCO are the first demonstrated use of reinforcement learning to




e

| TA-N-AUC | Module |
|---|---|
| **0.997 ± 0.006** | **EMS** |
| 0.944 ± 0.059 | No symm |
| 0.818 ± 0.122 | No mult |
| 0.717 ± 0.127 | None (large) |
| 0.566 ± 0.184 | No mult/symm |
| 0.522 ± 0.213 | None (medium) |
| 0.403 ± 0.170 | None (small) |

Figure 4: **EMS modules as components of an efficient visual learning system.** Validation reward obtained over the course of training for modules on **a.** 4-way stimulus-response with a reward map split into four quadrants, **b.** 2-way MTS with randomly moving match templates, **c.** 4-way MTS with two randomly moving class templates shown at a time, and **d.** 4-way MTS with four randomly positioned images shown at a time. Lines indicate mean reward over five different weight initializations. For clarity, only one subset of the total 24 tested ablation modules are displayed (see remaining modules in Supplementary). **e.** Area Under the Curve metric normalized to the highest performing module within a task (over all 24 modules), averaged across all 12 tasks.

achieve instance-level object segmentations. Reward curves (measuring bounding box IoU) in Fig. 5a show little difference between any of the late bottleneck modules at any size. The only models to consistently achieve an IoU above 0.4 are the EMS-like variants, especially with convolutional features. For context, a baseline SVR trained using supervised methods to directly regress bounding boxes using the same VGG features results in an IoU of 0.369.

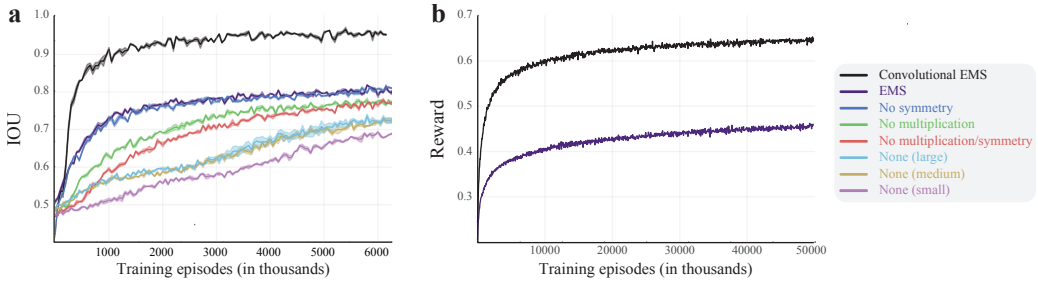



Figure 5: **Convolutional bottlenecks allow for fine resolution localization and detection in complex scenes.** **a.** Mean Intersection over Union (IoU) obtained on the localization task. **b.** Reward obtained on the MS-COCO match-to-sample variant. Both of these require their visual systems to accomodate for finer spatial resolution understanding of the scene, and more precise action placement than the SR or (non-COCO) MTS tasks. The convolutional variant of the EMS module uses skip connections from the conv5 and FC6 layers of VGG-16 as input, whereas the standard EMS uses only the FC6 layer as input.

## 4 Recurrent Neural Voting for Task Switching

We now extend the study of efficient modular learning to the case where the environment is now a sequence of tasks $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_\Omega\}$, each of which may last for an indeterminate period of time. A continual learning system should be capable of switching between tasks in $\mathcal{T}$ by e.g. containing a set of task-specific modules $\mathcal{M}$, where each module corresponds to a *task-specific* policy $\pi_\omega(a \mid x) = p(a_t \mid x_t; \tau_\omega)$. We accomplish this through augmenting the agent with one additional component: a meta-controller that mediates module deployment. This controller is in itself a neural

network, and must learn a meta-policy $\pi_{\mathcal{M}} = p\left(M \mid x_t; \tau_\omega\right)$ defining the distribution over modules (or parts of modules) that should be used in action selection given the task context (Wang et al., 2016).

Specifically, the circuitry of the meta-controller should be designed such that it can (i) determine which – if any – modules $M \in \mathcal{M}$ are most suitable for deployment through the controller meta-policy $\pi_{\mathcal{M}}$, (ii) transfer learned decision interfaces between these modules, and (iii) update meta-knowledge in $\pi_{\mathcal{M}}$ such that future task switches of similar form become quicker. Below, we introduce one such meta-controller and demonstrate its efficiency in handling a wide variety of task transitions.

Since the goal of this work is to discover optimal module and controller architectures for task performance and switching, we cue the agent when task transitions occur, although in general these transitions should also be inferred by the agent. However, we do not tell the agent *which* task is active at any given time, nor tell it how many different tasks to expect.

### 4.1 Recurrent Neural Voting

We formulate this probabilistically, where given a new task, the controller's meta-policy $\boldsymbol{\pi}_{\mathcal{M}}$ assigns subcomponents of each module in $\mathcal{M}$ a probability mass corresponding to how useful they are on the new task. Specifically, we introduce two different mechanisms which "vote" on reusing (or replacing entirely) either *layers* or *individual-units* of prexisting modules for solving a new task. This voting procedure occurs at every timestep, such that the controller continously evaluates the optimal module for deployment. We implement the controller which encodes these two concepts as a soft-voting and fully-differentiable recurrent neural network, where the activations of the layers (or neurons) themselves are the participants involved in calculating $\boldsymbol{\pi}_{\mathcal{M}}$.

**Layer Voting** Under the assumption that $\mathcal{M}$ is a set of modules with identical structure, then similar representations might be built across the *layers* of $\mathcal{M}$ at equal depths. We can create $\boldsymbol{\pi}_{\mathcal{M}}$ such that it sequentially computes the optimal $ith$ layer $l^{(i)}$ to deploy in the new module, conditioned on what layers it has already deployed:

$$\boldsymbol{\pi}_{\mathcal{M}}^{(i)} = p\left(l_{\mathcal{M}}^{(i)} \mid l_{\mathcal{M}}^{(k)}\right), \quad \text{for } k < i \tag{6}$$

where $l^{(0)} := \psi(x_t)$ is the original encoded input state.

We consider the case where this distribution is encoded as a learnable function inside the meta-controller, where $\boldsymbol{\pi}_{\mathcal{M}}^{(i)}$ is a Boltzmann distribution over the layer activations themselves, conditioned on the lower level activations from which they are calculated. That is,

$$\tilde{p}\left(l_{\mathcal{M}}^{(i)} \mid l_{\mathcal{M}}^{(k)}\right) = \text{softmax}\left(W^{(i)}\boldsymbol{\Gamma}^{(\boldsymbol{i})} + b^{(i)}\right), \quad \text{for } k < i \tag{7}$$

where $\boldsymbol{\Gamma}^{(\boldsymbol{i})}$ is the concatenation of all $ith$ level layer activations across modules in $\mathcal{M}$, and $W^{(i)} \in \mathbb{R}^{(T \cdot L) \times T}$ is a learnable weight matrix.

The controller approximates sampling from this distribution with a "soft-voting" mechanism, which computes the optimal layer to use at depth $i$ as the expected value over all layers in $\mathcal{M}$ at this depth:

$$\tilde{l}_{\omega}^{(i)} = \sum_{M \in \mathcal{M}} \tilde{p}_M^{(i)} l^{(i)} \tag{8}$$

**Single-Unit Voting** A useful refinement of the above mechanism involves voting across the *units* of $\mathcal{M}$ at the same position at each layer. Specifically, this can be sequentially constructed from the total population of neurons:

$$\boldsymbol{\pi}_{\mathcal{M}}^{(i,j)} = p\left(n_{\mathcal{M}}^{(i,j)} \mid n_{\mathcal{M}}^{(k,j)}\right), \quad \text{for } k < i \tag{9}$$

where $n_{\mathcal{M}}^{(i,j)}$ is the $jth$ neuron in layer $i$. The generalizations of eqs. (7) and (8) are:

$$\tilde{p}\left(n_{\mathcal{M}}^{(i,j)} \mid n_{\mathcal{M}}^{(k,j)}\right) = \text{softmax}\left(W^{(i,j)}\boldsymbol{\eta}^{(\boldsymbol{i},\boldsymbol{j})} + b^{(i,j)}\right), \quad \text{for } k < i \tag{10}$$

$$\tilde{n}_{\omega}^{(i,j)} = \sum_{M \in \mathcal{M}} \tilde{p}_{M}^{(i,j)} n^{(i,j)} \tag{11}$$

where $\boldsymbol{\eta}^{(i,j)}$ is the concatenation of all $i, jth$ neurons at position $j$ at depth $i$, and $W^{(i,j)} \in \mathbb{R}^{T \times T}$ is a learnable weight matrix.

Empirically, we find that the initialization schemes of the learnable controller parameters are important an consideration in the design itself, and that two specialized transformations also contribute slightly to its overall efficiency. For details on these, please refer to the Supplementary.

### 4.2 Switching Experiments

Using EMS module and (for control) large fully-ablated module, the recurrent neural voting controller was evaluated on 12 switching experiments using several variants of SR and MTS tasks (Table at the bottom of Fig. 6), using the same number units per layer and cross-validated learning rates as were found in § 3.2. Using these 12 task-switching scenarios, we test the ability of both module and controller to handle five switch paradigms: addition of new classes to the dataset (switch indexes 2, 7, 11 in the table of Fig. 6), replacing the current class set entirely with a new non-overlapping class set (switch ids. 1, 3), addition of motion to an MTS match screen (switch id. 6), additional MTS match screen distractor classes or SR reward boundary shifts (switch ids. 8, 12), or transitions between SR and MTS tasks (switch ids. 4, 5, 9, 10). These paradigms are not mutually exclusive, and overlap occurs between several. Controller hyperparameters were optimized in a cross-validated fashion (see Appendix F.1), and optimizations for three different initialization seeds were run to obtain mean and standard error.

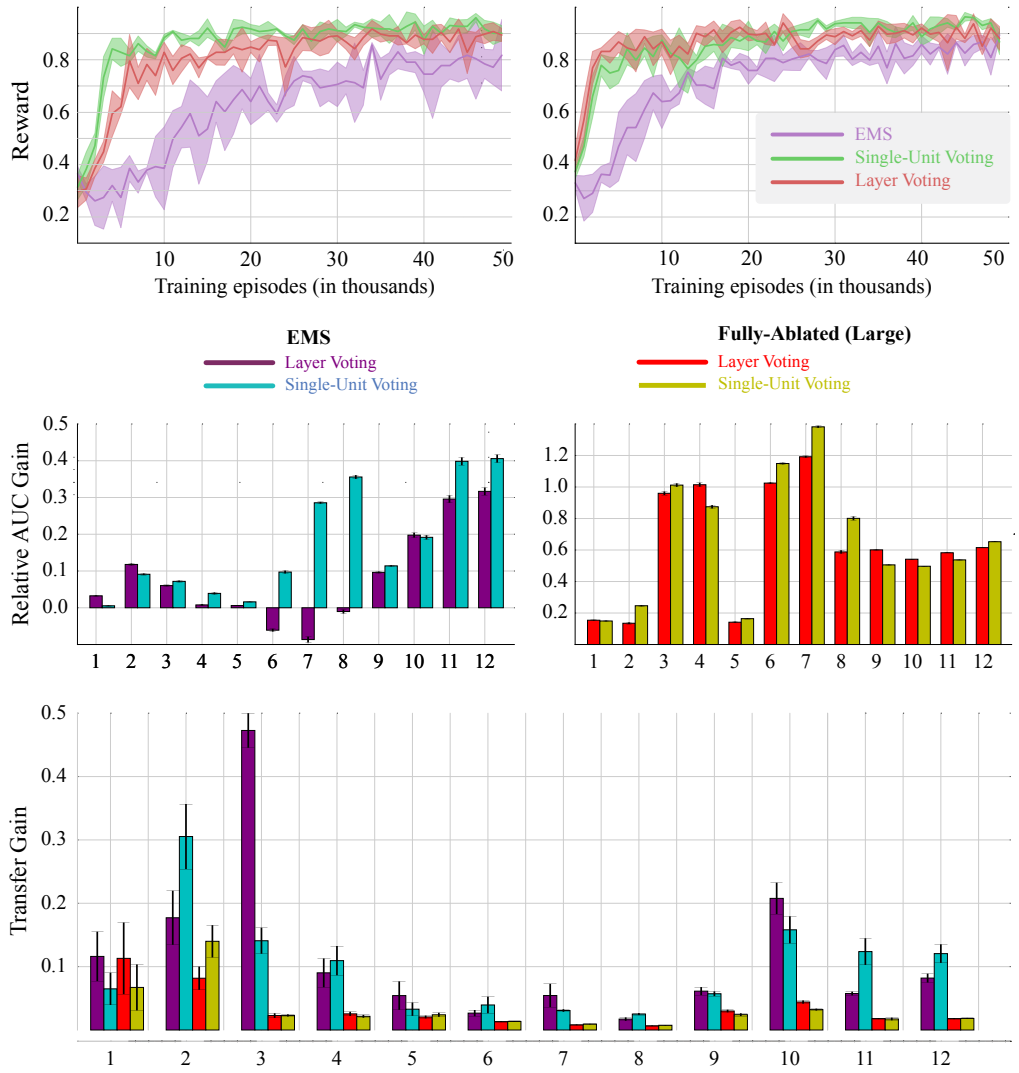
Figures 6a and b show characteristic switching curves for the EMS module for both the Layer Voting and Single-Unit Voting methods. Additional switching curves can be found in the Supplementary. Switching performance for each module and task was quantified with two metrics (see Figure S5 for a graphical illustration). The impact of switching on the module was measured by Relative Gain in AUC: $RGain = \frac{AUC(M^{switch}) - AUC(M)}{AUC(M)}$, where $M$ is the module trained from scratch on the second task, and $M^{switch}$ is the module transferred from an initial task using the recurrent voting controller. The temporal efficiency of transfer on a module architecture was quantified by Transfer Gain: $TGain = \frac{\Delta_{max}}{T_{\Delta_{max}}}$, where $T_{\Delta_{max}} = \text{argmax}(\Delta_t)$ is the time of maximum transfer $\Delta_{max}$.

We find that the recurrent voting controller allows for rapid positive transfer of both module types across all 12 task switches, where the general Single-Unit voting method is often a more powerful transfer mechanism than the Layer Voting method (Fig. 6 c). The large fully-ablated module, which was shown to be inefficient on single-task performance in § 3.2, benefits greatly from the introduction of the recurrent voting controller (Fig. 6 d). Nonetheless, the EMS-style module motif is still found to be significantly more transferable than the large fully-ablated module (Fig. 6e).

In other words, encoding a distribution over subcomponents of preexisting modules as a recurrent voting controller allowed for quick reuse of the learned decision structures which emerged naturally on the base task (e.g. the concept of an MTS button as a generic interface, or repurposing class knowledge in an SR task for use in an MTS task). Moreover, the same architectural motifs which were intuited to solve a relatively simple example (in (5)) and then were shown to generically solve a wide space of tasks in § 3.2, happen to be the *same* principles that allow it to quickly transfer its knowledge and be flexibly redeployed.

## 5 Conclusion and Future Directions

In this work, we introduce the Touchstream environment, a continual reinforcement learning framework that unifies a wide variety of spatial decision-making tasks within a single context. We describe a general algorithm (ReMaP) for learning light-weight neural modules that discover implicit task interfaces within this large-action/state-space environment. We show that a particular module architecture (EMS) is able to remain compact while retaining high task performance, and thus is especially suitable for flexible task learning and switching. We also describe a simple but general recurrent




| | | | | | |
|---|---|---|---|---|---|
| **1.** | 2-way SR | 2-way SR new classes | **7.** | 2-way vert-motion horiz-flip MTS | 4-way 2-shown vert-motion MTS |
| **2.** | 2-way SR | 4-way double-binary SR | **8.** | 4-way 2-shown vert-motion MTS | 4-way 4-shown permuted MTS |
| **3.** | 2-way stationary MTS | 2-way stationary MTS new classes | **9.** | 4-way double-binary SR | 4-way 4-shown stationary MTS |
| **4.** | 2-way SR | 2-way stationary MTS | **10.** | 4-way 4-shown stationary MTS t | 4-way quadrant SR |
| **5.** | 2-way stationary MTS | 2-way SR | **11.** | 2-way SR | 4-way quadrant SR |
| **6.** | 2-way stationary MTS | 2-way vert-motion horiz-flip MTS | **12.** | 4-way double-binary SR | 4-way quadrant SR |

**Task switching with Recurrent Neural Voting.** Post-Switching learning curves for the EMS module on the 4-way Quadrant SR task after learning **a.** 2-way SR task and **b.** a 4-way MTS task with 4 match screen class templates. Both the Layer Voting method and Single-Unit Voting method are compared against a baseline module trained on the second task from scratch. Across all twelve task switches, we evaluate the Relative Gain in AUC over baseline (RGain) using both voting methods for **c.** the EMS module and **d.** the large-sized fully-ablated late bottleneck MLP. **e.** Transfer Gain metrics are compared for both module types for each of the voting mechanisms. Colors are as in c. (EMS module) and d. (fully-ablated module).

Figure 6

task-switching architecture that shows substantial ability to transfer knowledge when modules for new tasks are learned.

A crucial future direction will be to expand insights from the current work into a more complete continual-learning agent. We will need to show that our approach scales to handle dozens or hundreds of task switches in sequence. We will also need to address issues of how the agent determines *when* to build a new module and how to *consolidate* modules when appropriate (e.g. when a series of tasks previously understood as separate can be solved by a single smaller structure). It will also be critical to extend our approach to handle visual tasks with longer horizons, such as navigation or game play with extended strategic planning, which will likely require the use of recurrent memory stores as part of the feature encoder.

From an application point of view, we are particularly interested in using techniques like those described here to produce agents that can autonomously discover and operate the interfaces present in many important real-world two-dimensional problem domains, such as on smartphones or the internet (Grossman, 2007). We also expect many of the same spatially-informed techniques that enable our ReMaP/EMS modules to perform well in the 2-D Touchstream environment will also transfer naturally to a three-dimensional context, where autonomous robotics applications (Devin et al., 2016) are very compelling.

## REFERENCES

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799, 2015. URL http://arxiv.org/abs/1511.02799.

Michael A Arbib. Schema theory. *The Encyclopedia of Artificial Intelligence*, 2:1427–1443, 1992.

Matthew M Botvinick and Jonathan D Cohen. The computational and neural basis of cognitive control: charted territory and new frontiers. *Cognitive science*, 38(6):1249–1285, 2014.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. URL http://arxiv.org/abs/1511.07289.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE CVPR*, 2009.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016. URL http://arxiv.org/abs/1609.07088.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$^2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL http://arxiv.org/abs/1611.02779.

Gabriel Dulac-Arnold, Richard Evans, Peter Sunehag, and Ben Coppin. Reinforcement learning in large discrete action spaces. *CoRR*, abs/1512.07679, 2015. URL http://arxiv.org/abs/1512.07679.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017. URL http://arxiv.org/abs/1701.08734.

David Gaffan and Susan Harrison. Inferotemporal-frontal disconnection and fornix transection in visuomotor conditional learning by monkeys. *Behavioural Brain Research*, 31(2):149 – 163, 1988. ISSN 0166-4328. doi: https://doi.org/10.1016/0166-4328(88)90018-6. URL http://www.sciencedirect.com/science/article/pii/0166432888900186.

Lev Grossman. Invention of the year: The iphone. *Time Magazine Online*, 1, 2007.

Alexa E Horner, Christopher J Heath, Martha Hvoslef-Eide, Brianne A Kent, Chi Hun Kim, Simon RO Nilsson, Johan Alsiö, Charlotte A Oomen, Andrew Holmes, Lisa M Saksida, et al. The touchscreen operant platform for testing learning and memory in rats and mice. *Nature protocols*, 8(10): 1961–1984, 2013.

R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. Learning to Reason: End-to-End Module Networks for Visual Question Answering. *ArXiv e-prints*, April 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL `http://arxiv.org/abs/1612.00796`.

A Krizhevsky, I Sutskever, and G Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2012.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL `http://arxiv.org/abs/1405.0312`.

James L McClelland. Incorporating rapid neocortical learning of new schema-consistent information into complementary learning systems theory. *Journal of Experimental Psychology: General*, 142 (4):1190, 2013.

Elisabeth A. Murray and Mortimer Mishkin. Object recognition and location memory in monkeys with excitotoxic lesions of the amygdala and hippocampus. *Journal of Neuroscience*, 18(16):6568–6582, 1998. ISSN 0270-6474. URL `http://www.jneurosci.org/content/18/16/6568`.

Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *CoRR*, abs/1703.01310, 2017. URL `http://arxiv.org/abs/1703.01310`.

R. Rajalingham, K. Schmidt, and J. J. DiCarlo. Comparison of object recognition behavior in human and monkey. *J. Neurosci.*, 35(35), 2015.

Ali S Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pp. 512–519. IEEE, 2014.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL `http://arxiv.org/abs/1606.04671`.

Michael N Shadlen and William T Newsome. Neural basis of a perceptual decision in the parietal cortex (area lip) of the rhesus monkey. *Journal of neurophysiology*, 86(4):1916–1936, 2001.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. *CoRR*, abs/1603.05201, 2016. URL `http://arxiv.org/abs/1603.05201`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Anthony D Wagner, Daniel L Schacter, Michael Rotte, Wilma Koutstaal, Anat Maril, Anders M Dale, Bruce R Rosen, and Randy L Buckner. Building memories: remembering and forgetting of verbal experiences as predicted by brain activity. *Science*, 281(5380):1188–1191, 1998.

Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *CoRR*, abs/1611.05763, 2016. URL `http://arxiv.org/abs/1611.05763`.

D Yamins, H Hong, C F Cadieu, E A Solomon, D Seibert, and J J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 2014.

Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Improved multitask learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*, 2017.

# SUPPLEMENTARY MATERIAL

## A TASK VARIANTS

The EMS module and all ablation controls were evaluated on a suite of 13 stimulus-response, match-to-sample, localization, and MS-COCO MTS variants:

1. 2-way SR - standard binary SR task
2. 4-way double binary SR - four class variant of SR, where each class is assigned either to the right or left half of the action space
3. 4-way quadrant SR - four class variant of SR, where each class is assigned to only a quadrant of the action space
4. 2-way stationary MTS - standard binary MTS task with stereotyped and non-moving match screens
5. 2-way stationary horiz-flip MTS - two class variant MTS task where the match templates' horizontal placement is randomly chosen, but confined within the same vertical plane
6. 2-way stationary vert-motion MTS - two class variant MTS task where the match templates' vertical position is randomly chosen, but each class is confined to a specific side
7. 2-way stationary vert-motion horiz-flip MTS - two class variant MTS task where the match templates' positions are completely random
8. 4-way 2-shown MTS - four class variant MTS task where only two class templates are shown on the match screen (appearing with random horizontal location as well)
9. 4-way 2-shown vert-motion MTS - same as above, but with random vertical motion for the templates
10. 4-way 4-shown stationary MTS - four class variant MTS task where all four class templates are shown on the match screen, but with fixed positions.
11. 4-way 4-shown permuted MTS - same as above, but with randomly permuted locations of all match templates
12. Localization - Localization task
13. MS-COCO MTS - 80-way MTS task using the MS-COCO detection challenge dataset, where match screens are randomly samples scenes from the dataset

## B EXPERIMENT DETAILS AND DATASETS

**Stimulus-Response Experiment Details:** Image categories used are drawn from the Image-Net 2012 ILSVR classification challenge dataset Deng et al. (2009). Four unique object classes are taken from the dataset: Boston Terrier, Monarch Butterfly, Race Car, and Panda Bear. Each class has 1300 unique training instances, and 50 unique validation instances.

**Match-To-Sample Experiment Details:** Sample screen images drawn from the same Image-Net class set as the Stimulus-Response tasks. One face-centered, unobstructed class instance is also drawn from the Image-Net classification challenge set and used as a match screen template image for that class. Class template images for the match screen were held fixed at 100x100 pixels. For all variants

Table S1: Number of units per layer for investigated modules

| Base-task | EMS | No symm | No Mult | No mult/symm | None-Small | None-Med | None-Large |
|---|---|---|---|---|---|---|---|
| SR | 8 | 8 | 8 | 8 | 8 | 128 | 512 |
| MTS | 32 | 32 | 32 | 32 | 32 | 128 | 512 |
| LOC | 128 | 128 | 128 | 128 | 128 | 512 | 1024 |

of the MTS task, we keep a six pixel buffer between the edges of the screen and the match images, and a twelve pixel buffer between the adjascent edges of the match images themselves. Variants without vertical motion have the match images vertically centered on the screen.

**Localization Experiment Details:** The Localization task uses synthetic images containing a single main salient object placed on a complex background (similar to images used in Yamins & DiCarlo (2016); Yamins et al. (2014)). There are a total of 59 unique classes in this dataset. In contrast to other single-class localization datasets (e.g. Image-Net) which are designed to have one large, face-centered, and centrally-focused object instance and for which a trivial policy of "always poke in image corners" could be learned, this synthetic image set offers larger variance in instance scale, position, and rotation so the agent is forced into learning non-trivial policies requiring larger precision in action selection.

**MS-COCO MTS Experiment Details** This task uses the entire MS-COCO detection challenge dataset Lin et al. (2014). On every timestep, a sample screen chosen from one of the 80 MS-COCO classes. These are constructed to be large, unobstructed, face centered representations of the class. For the match screen, we sample a random scene from MS-COCO containing any number of objects, but containing at least a single instance of the sample class. The agent is rewarded if its action is located inside any instance of the correct class. Both modules use sample actions from a low-temperature Boltzmann policy from eq. (4), which was empirically found to result in more precise reward map prediction.

# C MODULES

## C.1 UNITS PER LAYER

Table S1 aggregates the number of units per layer for the EMS and ablated modules which was used when conducting single-task and task-switching experiments. Only fully-connected modules' layer sizes are shown here. For details on the convolutional bottleneck EMS module, please refer to C.2.

## C.2 THE CONVOLUTIONAL-EMS MODULE

This is a "Convolutional Bottleneck" extension of the EMS module shown in the paper, where skip connections link the conv5 and the FC6 representation of the visual backbone. Here, the "scene-level" representation stored in the FC6 ReMaP memory buffer is tiled spatially to match the present convolution dimensions (here 14x14), and concatenated onto its channel dimension. A series of 1x1 convolutions plays the role of a shallow visual bottleneck, before the activations are vectorized and concatenated with $\mathcal{A}$ as input to the **CReS** layers of the standard EMS module.

The results in the paper are shown for a bottleneck consisting of a single tanh and two **CReS** convolutions, with 128 units each. The Downstream layers use 128 units each as well.

The motivation for the convolutional bottleneck is that lower-level features are useful for complex spatial tasks such as Localization and Object Detection, and hence may result in a more precise policy. By tiling the entire scene-level representation along the convolution layer's channel dimension, a form of multiplicative template-matching is possible between objects that must be memorized (e.g. MS-COCO MTS templates) and what is inside the present scene.

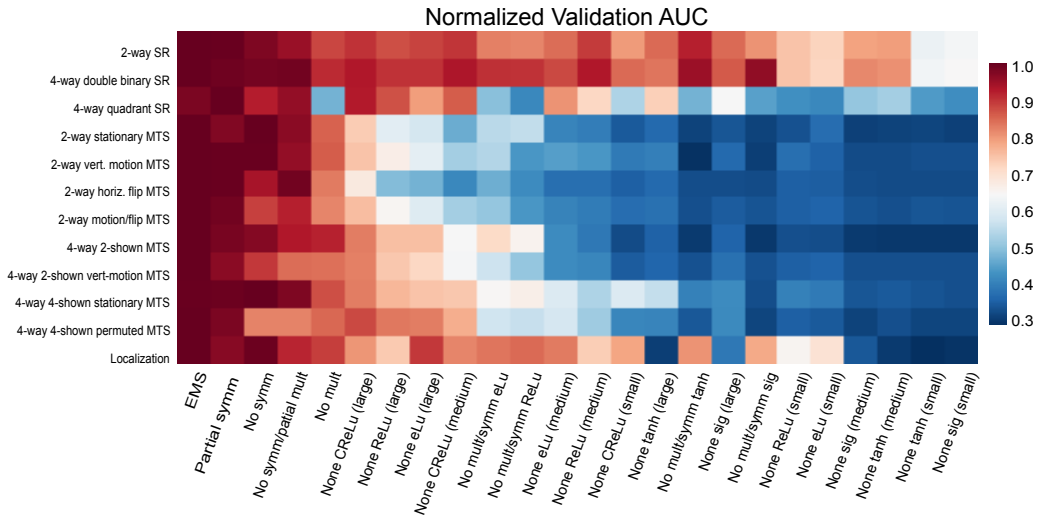



Figure S1: **Exhaustive module performance study** of the EMS module and 23 ablation control modules, measured as the Area Under the Curve for all SR, MTS, and LOC task variants. Shown is the AUC normalized to the highest performing module in a task. Results in fig. 4 have further averaged this over the vertical task axis, and report only a salient subset of the ablations.

# D EXHAUSTIVE ABLATION STUDY

In all, we investigated 23 distinct ablations on the EMS module, across all twelve task variants outlined in sec A (Fig. S1). Symmetry ablations replace **CReS** with the activation $x \mapsto \mathbf{ReLu}(x) \oplus x^2$ Multiplicative ablations are denoted by specifying the nonlinearity used in place of **CReS** (where this is one of $\mathbf{ReLu}$, $\tanh$, sigmoid, elu Clevert et al. (2015), or $\mathbf{CReLu}$ Shang et al. (2016)). This additionally includes one partial symmetry ablation (denoted "partial symm") where only the visual bottleneck is symmetric, and one which ablates the **ReLu** from the "no symm" module (denoted "no symm/partial-mult").

## D.1 HYPERPARAMETERS

Learning rates for the ADAM optimizer were chosen on a per-task basis through cross-validation on a grid between $[10^{-4}, 10^{-3}]$ for each architecture. Values used in the present study may be seen in Table S2.

# E ADDITIONAL LEARNING CURVES

## E.1 SINGLE-TASK ABLATION EXPERIMENTS

Learning trajectories for eight additional tasks are provided in Figure S2. Modules capable of convergence on a task were run until this was acheived, but AUC values for a given task are calculated at the point in time when the majority of models converge.

## E.2 RECURRENT VOTING CONTROLLER AND EMS MODULE TASK-SWITCHING EXPERIMENTS

Additional trajectories for ten unshown switching curves are provided in Figure S3.

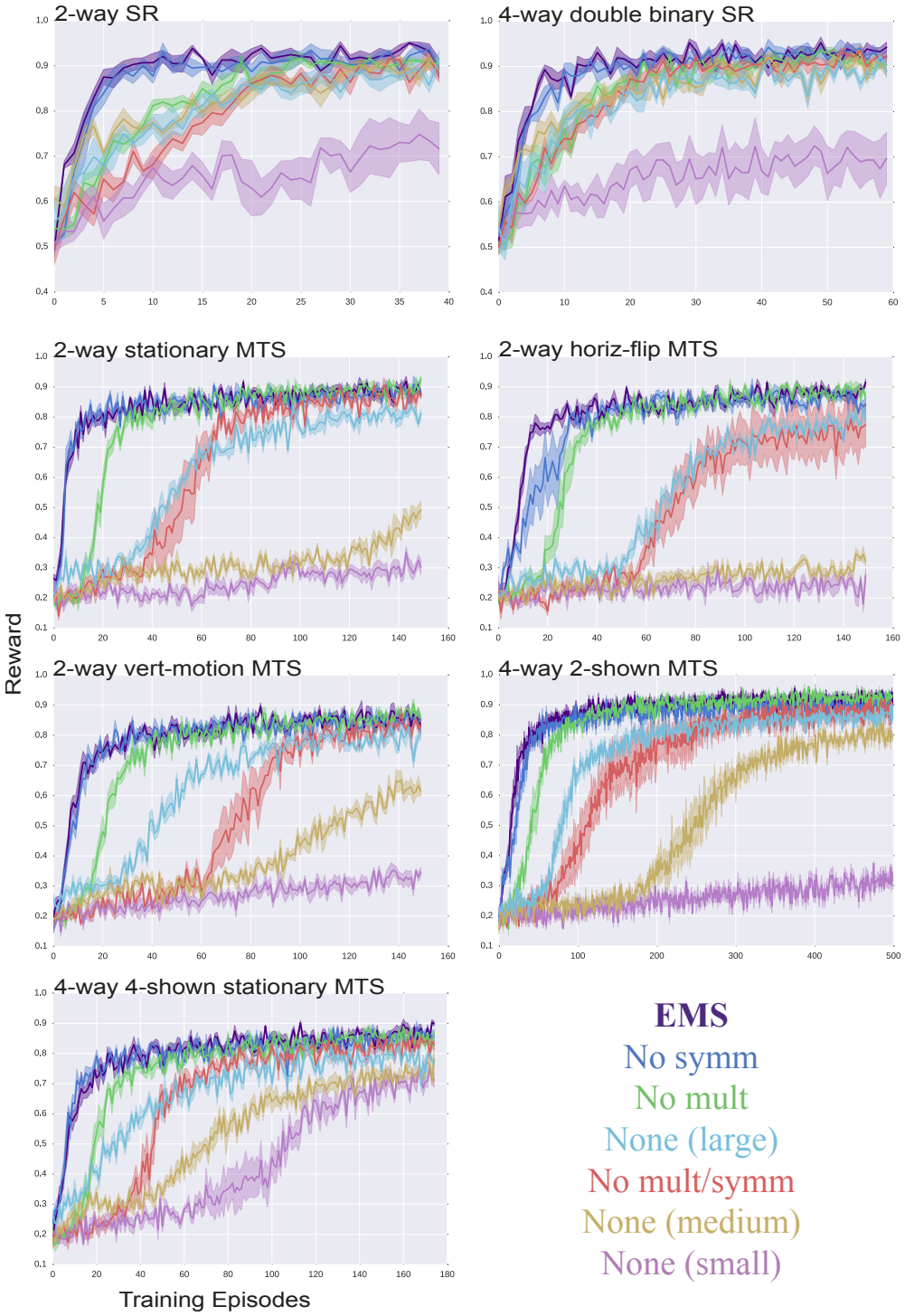



Figure S2: **Additional Single-task performance ablation Learning curves.** Seven learning curves shown for task variants not seen in the main text body. Shown are the same ablations as the main text.

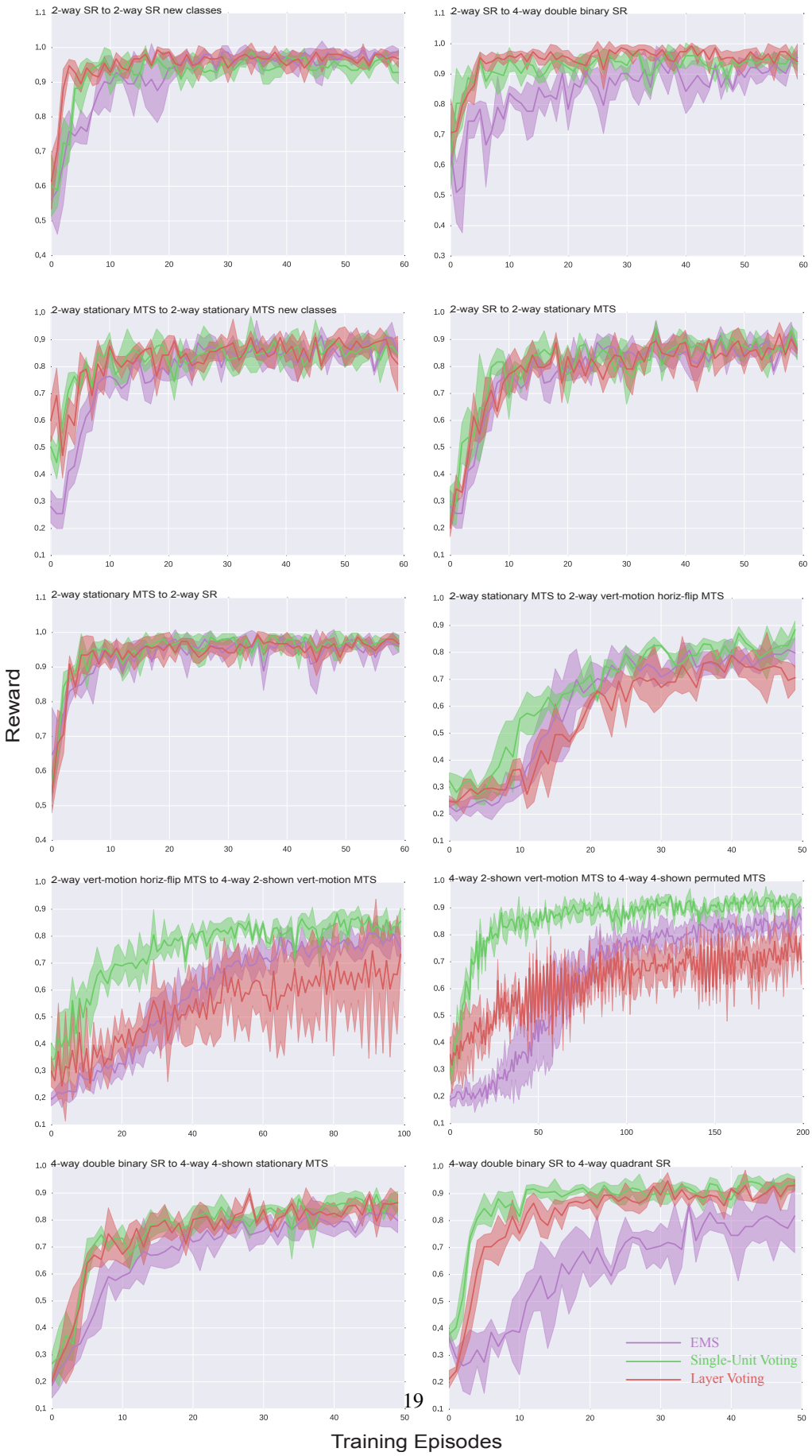

2-way SR to 2-way SR new classes
2-way SR to 4-way double binary SR
2-way stationary MTS to 2-way stationary MTS new classes
2-way SR to 2-way stationary MTS
2-way stationary MTS to 2-way SR
2-way stationary MTS to 2-way vert-motion horiz-flip MTS
2-way vert-motion horiz-flip MTS to 4-way 2-shown vert-motion MTS
4-way 2-shown vert-motion MTS to 4-way 4-shown permuted MTS
4-way double binary SR to 4-way 4-shown stationary MTS
4-way double binary SR to 4-way quadrant SR
EMS
Single-Unit Voting
Layer Voting
Reward
Training Episodes

Table S2: Module learning rates

| | 2-way SR | 4-way double binary SR | 4-way stationary SR | 2-way stationary MTS | 2-way vert-motion MTS | 2-way horiz flip MTS | 2-way motion/flip MTS | 4-way 2-shown MTS | 4-way 2-shown vert-motion MTS | 4-way 4-shown stationary MTS | 4-way 4-shown permuted MTS | LOC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EMS | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $10^{-4}$ |
| Partial symm | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $5{\cdot}10^{-4}$ | $10^{-4}$ |
| No symm | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $2{\cdot}10^{-4}$ | $10^{-4}$ |
| No symm/partial mult | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $2{\cdot}10^{-4}$ | $10^{-4}$ |
| No mult/symm ReLu | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ |
| No mult/symm tanh | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| No mult/symm sig | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| No mult/symm eLu | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ |
| No mult/symm CReLu | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ |
| None ReLu(small) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ |
| None ReLu(medium) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None ReLu(large) | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None tanh(small) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None tanh(medium) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None tanh(large) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None sig(small) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-3}$ | $10^{-4}$ |
| None sig(medium) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None sig(large) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None eLu(small) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ |
| None eLu(medium) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ |
| None eLu(large) | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| None CReLu(small) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ |
| None CReLu(medium) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-3}$ | $10^{-4}$ | $10^{-4}$ |
| None CReLu(large) | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |

# F RECURRENT VOTING CONTROLLER AUGMENTATIONS

## F.1 LEARNABLE PARAMETER INITIALIZATIONS

Here we describe the weight initialization scheme that was found to be optimal for use with the recurrent coting controller. For simplicity, consider the layer-voting mechanism, with learnable weight matricies $W^{(i)}$ and biases $b^{(i)}$. The intended biasing scheme is achieved through initializing the elements of these parameters to:

$$W_{\omega}^{(i)} \sim \begin{cases} |\mathcal{N}(\mu^{(0)}, 0.001)| & \text{if } i = 0,\ \omega < \Omega \\ |\mathcal{N}(\mu^{(1)}, 0.001)| & \text{if } i > 0,\ \omega < \Omega \\ |\mathcal{N}(0.01, 0.001)| & \text{if } \omega = \Omega \end{cases} \tag{12}$$

$$b_{\omega}^{(i)} = \begin{cases} b^{(0)} & \text{if } i = 0,\ \omega < \Omega \\ b^{(1)} & \text{if } i > 0,\ \omega < \Omega \\ 0.1 & \text{if } \omega = \Omega \end{cases} \tag{13}$$

This initialization technique was also generalized for use with the single-unit voting mechanism.

For the switching experiments presented in section § 4.2, we sweep the hyperparameters on a narrow band around the default scheme. The ranges for these are: $\mu^{(0)} \in [0.01, 0.005]$ , $b^{(0)} \in [0.1, 0.01]$ , $\mu^{(1)} \in [0.01, 0.02]$ , and $b^{(1)} \in [0.1, 0.2, 0.5, 1.0]$.

## F.2 TARGETED TRANSFORMATIONS

Two additional switching mechanisms were added to the controller to augment its ability to switch between taks which are remappings of the action space or reward policy of a preexisting module.

### F.2.1 ACTION TRANSFORMATIONS

we note that efficient modules are those which can effectively produce a minimal representation of the interaction between action space $\mathcal{A}$ and observation $x_t$. If the agent's optimal action space shifts

to $\mathcal{A}'$ while the remainder of the task context remains fixed, the controller should allow for rapid targeted remapping $\mathcal{A} \mapsto \mathcal{A}'$. Since we formulate the modules as ReMaP Networks, and $\mathcal{A}$ is an input feature basis, we can achieve remappings of this form through a fully-connected transformation:

$$\mathbf{a}'_\tau = f(W_a \mathbf{a}_\tau + b) \tag{14}$$

where $\mathbf{a}_\tau = [\mathbf{h}_{t-kb:t-1}, a_t]$ is the vector of action histories, and $W_a$ and $b$ embed $\mathbf{a}_\tau$ into new action space $\mathcal{A}'$ using only a small number of learnable parameters.

**Pseudo Identity-Preserving Transformation** In practice, we initialize the parameters in eq. (14) such that the transformation is *pseudo identity-preseving*, meaning that the representation learned at this level in the original module is not destroyed prior to transfer.

This is done by initializing $W_a$ to be an identity matrix $I_{|\mathbf{a}_\tau|}$ with a small amount of Gaussian noise $\epsilon \sim \mathcal{N}(0.0, \sigma^2)$ added to break symmetry. $b$ is initialized to be a vector of ones of size $|\mathbf{a}_\tau|$.

#### F.2.2 Reward Map Transformations

Each of the $k_f$ maps $m_t(x)$ reflects the agent's uncertainty in the environment's reward policy. If the task context remains stationary, but the environment transitions to new reward schedule $\mathcal{R}'$ that no longer aligns with the module's policy $\pi$, the controller could to this transition by e.g. containing a mechanism allowing for targeted transformation of $m(x)$ and hence also $\pi$.

One complication that arises under ReMaP is that since each task-module *learns* its optimal action space internally, $m(x)$ are in the basis of $\mathcal{R}$ rather than $\mathcal{A}$. Therefore, transformations on the map distribution must also re-encode $\mathcal{A}$ before mapping to $\mathcal{R}'$.

In this work, we investigate a shallow "adapter" neural network that lives on top of the existing module and maps $\mathcal{R} \mapsto \mathcal{R}'$. Its first and second layers are defined by

$$l_1(x) = f(W_1[m(x) \odot g(\mathbf{a}_\tau), \mathbf{a}_\tau] + b_1 \tag{15}$$

$$m(x)' \propto W_2 l_1 + b_2 \tag{16}$$

where $g(\mathbf{a}_\tau)$ is a similar transformation on $\mathcal{A}$ as above, $\odot$ denotes elementwise multiplication, $W_1$ is a learnable matrix embedding into a hidden state, and $W_2 \in \mathbb{R}^{|l_1| \times |\mathcal{R}'|}$ is a learnable matrix embedding into $\mathcal{R}'$

**Pseudo Identity-Preserving Transformation** Similar to the transformation on the action space, we modify the reward-map transformation to be pseudo identity-preserving as well. This is done by modifying eq. (15) such that the original maps are concatenated on to the beginning of the transformation input vector:

$$l_1(x) = f(W_1[m(x), m(x) \odot g(\mathbf{a}_\tau), \mathbf{a}_\tau] + b_1 \tag{17}$$

The intended map-preserving transformation is accomplished via initializing $W_1$ and $W_2$ as:

$$W^{(i,j)} \sim \begin{cases} 1.0 + \mathcal{N}(0.0, \epsilon) & \text{if } i = j,\ i < \mathcal{R} \\ \mathcal{N}(0.0, \epsilon) & otherwise \end{cases} \tag{18}$$

### F.3 Targeted Transformation Hyperparameters

Both of the targeted transformations have several hyperparameters. We conducted a grid search to optimize these in a cross-validated fashion, on a set of test task switches designed to be solved by one of the targeted transformations. Each was conducted independently of the recurrent voting controller, and independently of the other transformation. Optimal hyperparameters found in these experiments were fixed for use in the integrated recurrent voting controller, and were not further optimized afterwards.

**Action Transformation Hyperparameters** We conducted three tests using the stimulus-response paradigm: class reversal (in which the left class becomes the right class and vice-versa), a horizontal

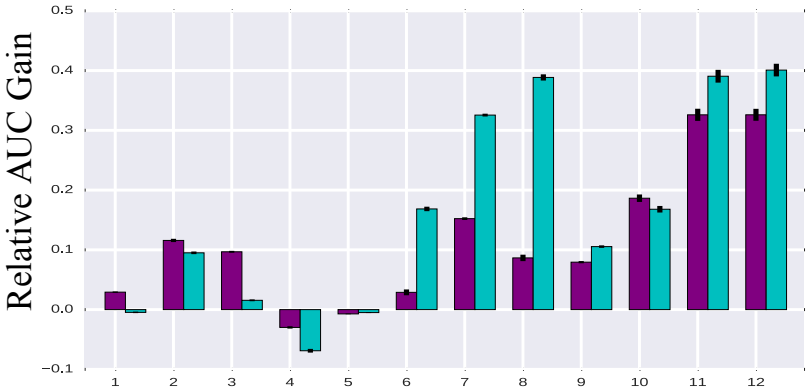



Figure S4: **Targeted controller transform ablation.** Relative AUC Gain for the EMS module over the same switching snearios in the paper, but with the targeted transformations ablated.

rotation of the reward boundaries (such that right becomes up and left becomes down), and a "switch" to the original task (intended to test the identity-preserving component).

In this work, we find that a single, non-activated linear transformation ($f$ in (14)) is optimal for this new state-space embedding, using $k_b * 2$ units, and initialized such that the idendity-preserving transformation weights have $\sigma = 0.01$. The learning rate for this transformation was found to be optimal at $0.1$.

**Reward Map Transformation Hyperparameters** We conducted two tests using the stimulus-response paradigm: a "squeezing" task (where there is no longer any reward dispensed on the lower half of the screen), and a "switch" to the original task (intended to test the identity-preserving component).

In this work, we find the optimal activations in (17) to be $f(\cdot) =$ **CReS** and $g(\cdot) =$ **ReLu**, with 4 units in the hidden layer. $\epsilon$ in the weight initialization scheme was found optimal at 0.001, and an initial bias of 0.01. The optimal learning rate for this transformation was found to be 0.01.

### F.4 Transform Ablation

A study was conducted to determine the relative benefit of the targeted transformations (Fig. S4), where it was determined that the primary contribution of the recurrent neural controller was in fact the voting mechanism (although the transformations did supplement this as well).

### F.5 Deployment scheme of task modules

When cued into task transition, the controller freezes the learnable parameters of the old task-module, and deploys a new unitialized task-module. The controller then initializes the action and reward map transformation networks as described in F.2 on top of the old module. These transformations are also voted on inside the recurrent neural controller at every timestep.

## G Switching Metrics

Figure S5 graphically illustrates the metrics used inside the paper to quantify switching performance: $RGain$ and $TGain$.

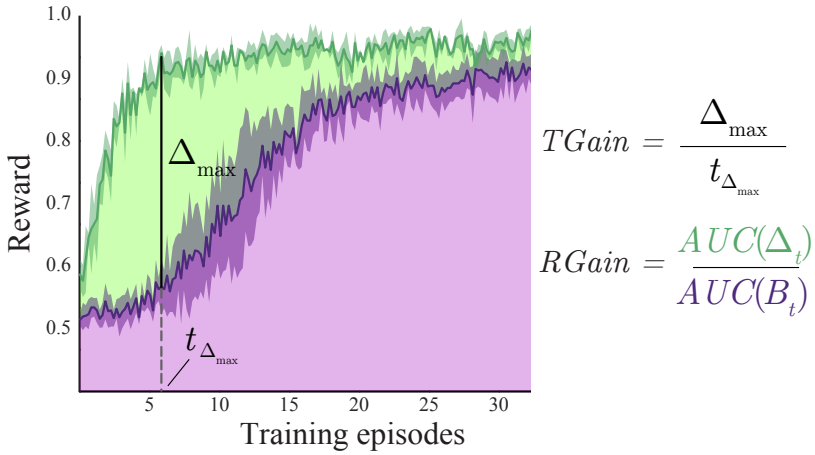



Figure S5: **Illustration of switching performance metrics.** We quantify the switching performance of the recurrent neural controller and task-modules by two metrics: "relative gain in AUC" (ratio of green to purple shaded regions), and "transfer" gain (difference of reward at $T_{\Delta\max}$). Relative AUC measures the overall gain relative to scratch, and the transfer gain measures the speed of transfer. Curve shown is the EMS module with Single-Unit voting method evaluated on a switch from a 4-way MTS task with two randomly moving class templates to a 4-way MTS task with four randomly moving templates.