Transforming task representations to perform novel tasks

Andrew K. Lampinen[a,1] and James L. McClelland[a]

[a]Department of Psychology, Stanford University, Stanford CA 94305

Edited by Terrence J. Sejnowski, Salk Institute for Biological Studies, La Jolla, CA, and approved November 6, 2020 (received for review May 11, 2020)

An important aspect of intelligence is the ability to adapt to a novel task without any direct experience (zero shot), based on its relationship to previous tasks. Humans can exhibit this cognitive flexibility. By contrast, models that achieve superhuman performance in specific tasks often fail to adapt to even slight task alterations. To address this, we propose a general computational framework for adapting to novel tasks based on their relationship to prior tasks. We begin by learning vector representations of tasks. To adapt to new tasks, we propose metamappings, higher-order tasks that transform basic task representations. We demonstrate the effectiveness of this framework across a wide variety of tasks and computational paradigms, ranging from regression to image classification and reinforcement learning. We compare to both human adaptability and language-based approaches to zero-shot learning. Across these domains, metamapping is successful, often achieving 80 to 90% performance, without any data, on a novel task, even when the new task directly contradicts prior experience. We further show that metamapping can not only generalize to new tasks via learned relationships, but can also generalize using novel relationships unseen during training. Finally, using metamapping as a starting point can dramatically accelerate later learning on a new task and reduce learning time and cumulative error substantially. Our results provide insight into a possible computational basis of intelligent adaptability and offer a possible framework for modeling cognitive flexibility and building more flexible artificial intelligence systems.

cognitive science | artificial intelligence | transfer | zero-shot

Adaptability is a key feature of biological intelligence—adaptation is necessary for a system to efficiently handle all of the vagaries of its environment (1). An advantage of neural networks over ordinary computer programs is that they can adapt by learning from training examples. Yet this is only a limited form of adaptability. An intelligent system should be able to transform its behavior on a task in accordance with a change in goals, and humans often exhibit this form of adaptability (2). For example, if we are told to try to lose at poker, we can perform quite well on our first try, even if we have always tried to win previously. If we are shown an object and told to find the same object in a new color or texture, we can do so. By contrast, this type of first-try adaptation is quite difficult for standard deep-learning models (2–4). How could models reuse their knowledge more flexibly?

We suggest that this ability to adapt can arise from exploiting the relationship between the adapted version of the task and the original. In this work, we propose a computational model of adaptation based on task relationships and demonstrate its success across a variety of domains, ranging from regression to classification to reinforcement learning. Our approach could provide insights into the flexibility of human cognition and allow for more flexible artificial intelligence systems.

Our model incorporates several key cognitive insights. First, to perform different tasks, it is useful for the system to constrain its behavior by an internal task representation (5). Prior work in machine learning and cognitive science has constructed task representations from a natural language instruction (6–8) or by learning to infer task representations from examples, a procedure called metalearning (9, 10). We extend these ideas, proposing that the model can adapt to a novel task by transforming its representation for a prior task into a representation for the new task, thereby exploiting the task relationship to perform the new task.

We refer to these transformations of task representations as metamappings. That is, metamappings are higher-order functions over tasks—functions that take a task as input and transform it to produce an adapted version of that task. Metamappings allow the model to adapt to a new task zero shot (i.e., without requiring any data from that new task), based on the relationship between the new task and prior tasks. We propose that metamapping is a powerful way to promote adaptation, because the task relationships it exploits are the fundamental conceptual structure on which systematic generalization can be predicated.

As a concrete example, our model is able to switch to losing at poker on its first try. To do so, it constructs a representation of poker from experience with trying to win the game. It then infers a "try-to-lose" metamapping, either from language or from examples of winning and losing at other games, such as blackjack. It then applies this metamapping to transform its representation of poker, thereby yielding a representation for losing at poker. This adapted task representation can then be used to perform the task of trying to lose at poker zero shot—that is, without any prior experience of losing at poker.

Our main contributions are 1) to propose metamapping as a computational framework for zero-shot adaptation to novel tasks and 2) to provide a parsimonious architecture for metamapping.

Significance

An intelligent system should be able to adapt to a novel task without any data and achieve at least moderate success. Humans can often do so, while models often require immense datasets to reach human-level performance. We propose a general computational framework by which models can adapt to new tasks based only on their relationship to old tasks. Our approach is based on transforming learned task representations. Our approach allows models to perform well on novel tasks, even using novel relationships not encountered during training. This adaptation can substantially accelerate later learning. Our work could contribute to understanding the computational basis of intelligence, to cognitive modeling, and to building more flexible forms of artificial intelligence.

Author contributions: A.K.L. and J.L.M. designed research; A.K.L. performed research; A.K.L. analyzed data; and A.K.L. and J.L.M. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]To whom correspondence may be addressed. Email: andrewlampinen@gmail.com.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2008852117/-/DCSupplemental.

First published December 10, 2020.

We demonstrate the success of metamapping across a variety of task domains, ranging from visual classification to reinforcement learning, and show that the model can even adapt using new metamappings not encountered during training. We further show that adapting by metamapping provides a useful starting point for later learning. This work proposes transforming a task representation to adapt zero shot. We consider related work and implications for cognitive science and artificial intelligence in *Discussion*.

Task Transformation via Metamappings

Basic Tasks Are Input–Output Mappings. We take as a starting point the construal of basic tasks as mappings (functions) from inputs to outputs. For example, poker can be seen as a mapping from hands to bets (Fig. 1*A*), chess as a mapping of board positions to moves, and object recognition as a mapping from images to labels. This perspective is common in machine-learning approaches, which generally try to infer a task mapping from many input/output examples or metalearn how to infer it from fewer examples. We use the phrase "basic task" to refer to any elementary task a system performs (e.g., any card game), including both standard tasks ("poker") and variants that can be produced by a transformation ("lose at poker").

Tasks Can Be Transformed via Metamappings. We propose metamappings as a computational approach to the problem of transforming a basic task mapping. A metamapping is a higher-order task, which takes a task representation as input and outputs a representation of the transformed version of the task. For example, we might have a "lose" metamapping (Fig. 1*D*) that would transform the representation of poker into a representation of losing at poker.

How can a metamapping be performed? We exploit an analogy between metamappings and basic task mappings—both are simply functions from inputs to outputs. Thus, to perform a metamapping we use approaches analogous to those we use for basic tasks. We infer a metamapping from examples (e.g., winning and losing at a set of example games) or natural language (e.g., "try to switch to losing"). We can then apply this metamapping to other basic tasks, to infer losing variations of those tasks. Importantly, the system can generalize to new metamappings—task transformations never seen in training—as well as to new basic tasks.

Model Architecture and Training Methods

We propose a class of architectures that can both perform basic tasks and adapt to task alterations via metamappings. In this section, we describe the general features of our architectures and their training. See *SI Appendix*, section A for details, including a formal model description, all hyperparameters, etc.

Constructing a Task Representation (Fig. 1*B*). When humans perform a task, we need to know what the task is. In our model, we specify the task using a task representation, which we derive from language, from supporting examples of appropriate behavior, or from metamapping (see *Transforming Task Representations via Metamappings*). To construct a task representation from language, we process the language through a deep recurrent network (long-short term memory), as in other work (7, 8, 11). To construct a task representation from examples, as in other work (12), we process each example (i.e., an input and its

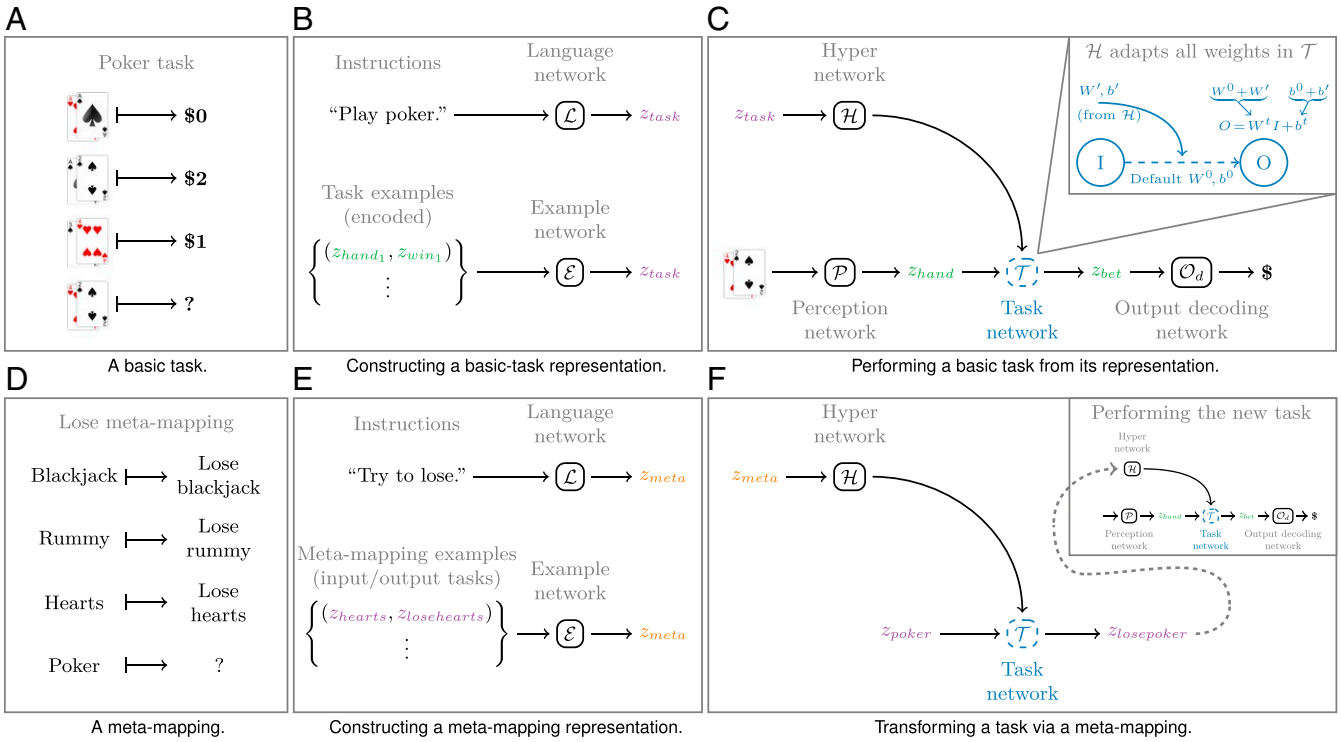


Fig. 1. Performing and transforming tasks with a metamapping architecture. (*A*) Basic tasks are mappings from inputs to outputs, which can be generalized from examples. (*D*) Metamappings are mappings from tasks to other tasks, which can be generalized from examples. (*B* and *E*) The architecture performs basic tasks and metamappings from a task representation, which can be constructed from a language cue or examples. (*C*) The task representation is used to alter the parameters of a task network (*Inset*) which executes the appropriate task mapping. (*F*) The metamapping representation is used to parameterize the task network to transform a task representation. The transformed representation can then be used to perform the new task zero shot (*Inset*). Our architecture exploits a deep analogy between basic tasks and metamappings—both can be seen as mappings of inputs to outputs. This analogy is reflected in the parallels between *A–C* and *D–F*.

corresponding target) to construct an appropriate representation of the example and then aggregate across those representations by taking an elementwise maximum, to combine examples in a nonlinear but order-invariant way. This aggregated representation then receives further processing to produce the task representation.

Performing a Task from Its Representation (Fig. 1*C*). Once we have a task representation, we use it to perform the task. We allow a large part of the input processing (perception) and output decoding (action) to be shared across the tasks within each domain we consider, so that the task-specific computations can be relatively simple and abstract.* For example, if a human is playing card games, the cards will be identical whether the game is poker or bridge, and the task-specific computations will be performed over abstract features such as suit and rank relationships. We thus allow the system to learn a general basis of perceptual features over all tasks within a domain.

The system then uses these features in a task-specific way to perform task-appropriate behavior. Specifically, the model uses a HyperNetwork (13, 14) which takes as input the representation of a task. This network adapts the values of learned "default" connection weights, to make the network task sensitive (Fig. 1*C*, *Inset*). The adapted network transforms the perceptual features into task-appropriate output features, which can then be decoded to outputs via a shared output decoding network. The whole model (including the construction of the task representations) can be trained end to end, just as a standard metalearning system would be. Our approach outperforms an alternative architecture, in which the task representation is provided as another input to a feedforward task network (*SI Appendix*, Fig. S10.)

Transforming Task Representations via Metamappings (Fig. 1 *E* and *F*). We defined a metamapping to be a higher-order task, which takes as input a task representation and outputs a transformed task representation. Thus, we need a way of transforming the task representations constructed in *Constructing a Task Representation*. To do so, we exploit the functional analogy between basic tasks and metamappings. We infer a representation for a metamapping from examples of that metamapping or from a language description, just like we infer a basic task representation from examples or language. We use this metamapping representation to adapt the parameters of the task network to transform other task representations. This approach is analogous to how we used a representation of a basic task to adapt the task network to perform that task. (See *SI Appendix*, section G.1 for proof that a simpler vector-analogy metamapping approach is inadequate.)

Homoiconicity. Our architectures exploit the analogy between basic tasks and metamappings by using exactly the same networks (with exactly the same parameters) to infer and perform a metamapping as for inferring and performing a basic task. To allow this, the system embeds individual data points, task representations, and metamapping representations into a shared representational space. This means that all task- or metamapping-specific computations can be seen as operations on objects in this shared space and can be inferred using the same processes regardless of object type. (Note that sharing of the space is only enforced implicitly in that the same networks are processing different entities.) This approach is in keeping with the idea that humans have a single mind that implements computations of all types. Our approach is also inspired by the computational notion of homoiconicity. In a homoiconic programming language programs can be manipulated just as data can. Our task representations are like programs that perform tasks, and our implementation is thus homoiconic in the sense that it operates on data and tasks in the same way.

Homoiconicity is parsimonious, in that it does not require adding new networks for each new type of computation. Furthermore, in many cases, functions have some common structure with the entities they act over. For example, both numbers and functions can have inverses. For another example, the set of linear maps over a vector space is itself a vector space. If the different levels of abstraction share structural features, sharing computation should improve generalization. Homoiconicity could also support the ability to build abstractions recursively on top of prior abstractions, as humans do in mathematical cognition (15–17). Although homoiconicity is not a necessary part of metamapping, we suggest that homoiconic approaches will be beneficial and verify this empirically, see *Polynomials* section).

Classifying Task Representations. In several domains, we also trained the model to classify task representations by relevant attributes (for example, whether a game was a variation of poker), again using the same architectural components. See *SI Appendix*, section A.3 for details. This may improve generalization by helping the model constrain its representation of the task space, but is not essential (*SI Appendix*, Fig. S11).

Training the Model. We train the system in epochs, during which it receives one training step on each trained basic task and one training step on each trained metamapping, interleaved in a random order. To train the system to perform the basic tasks, we compute a task-appropriate loss at the output of the output decoding network and then minimize this loss with respect to the parameters in all networks. This includes the networks used to construct the task representation and even the representations of the examples or language input. That is, we train the system end to end to perform the basic tasks.

When constructing a task representation from examples, we do not allow the example network to see every example in the training batch. This forces the model to generalize in a standard metalearning fashion. Specifically, we separate the batch of examples into a support set which is provided to the example network and a probe set which is only passed through the task network to compute an output, from which a loss can be computed against a task target. For example, in a card game the system will have to construct a task representation from the support hands that will be useful for playing the probe hands. This approach encourages the task representations to capture the task structure, rather than just memorizing examples. We randomly split the training examples into support and probe sets on each step, so that over the course of training every training example would fill both roles. In this approach, the task representation is constructed anew at each training step. However, to stabilize learning in difficult domains, it can be useful to maintain a persistent task representation which updates slowly with each new set of examples (*SI Appendix*, section A.3).

Training the system to construct basic task representations from language is similar, except that a language description (e.g., "play poker") is provided rather than examples. Thus, no support set is needed, so all examples can be used as probes.

To train the system to perform metamappings from examples, we start with a training set of example task representation pairs, where each pair consists of a source task representation and the corresponding transformed task representation. Again, on each training step, a subset of these examples is used as a support set to construct a metamapping representation. The remaining examples are used as a probe set to train the system to transform

*Of course, with different input types, this type of processing will be different. While the core model components are similar across experiments, the input and output systems can therefore differ.

the source representation for each pair to its corresponding target. Specifically, we present the source task embedding as input to the task network and minimize an ℓ_2 loss on the difference between the output embedding the task network produces and the task representation for the target transformed task. For example, suppose the system has been trained to play winning and losing variations of blackjack, hearts, and rummy. We might use the representations of winning and losing hearts and rummy as support-set examples to instantiate the metamapping, then input the task representation for winning blackjack as a probe, and try to match the output to the task representation for losing blackjack. Again, we randomly chose which examples were used as support or probes on each training step. On the next training step, we might use hearts and blackjack as examples and train the metamapping to generalize to losing at rummy.

Training the system to perform metamappings from language is similar, except that again a language description (e.g., "switch to losing") is provided rather than examples of the transformation. Thus, as when using language rather than examples to perform basic tasks, all pairs can be used as probes.

Evaluating Base-Task and Metamapping Performance. After training, we can evaluate the model's base-task performance using held-out examples unseen during training. To test generalization of a metamapping (e.g., try to lose), we can pass in the representation for a task that has never been used for any training on this metamapping (either as a support example or as a probe for generalization), for example, poker. We construct a metamapping representation using all of the training examples of the lose metamapping as a support set. We then apply the lose metamapping to the task representation of poker (i.e., pass it through the task network parameterized by the lose metamapping representation) to produce a transformed representation. We then actually perform the losing variation of poker with this transformed representation. Metamapping performance is always evaluated by zero-shot performance on held-out tasks that the system has never performed during training.

In metamapping, generalization is possible at different levels of abstraction. The paragraph above refers to basic generalization—applying a metamapping seen during training to a basic task that metamapping has not been applied to during training, to perform a held-out transformed version of that task. However, if the system has experienced sufficiently many metamappings during training, we can also test its ability to generalize to held-out metamappings. For example, if the system has been trained to switch various pairs of colors in a classification task (red for blue, green for yellow, etc.), it should be able to generalize to switching held-out pairs (red for yellow, green for blue, etc.) from an appropriate cue (examples or instructions). That is, even if a metamapping has never been encountered during training, we can construct a representation for it by providing a support set of transformation examples or a language instruction that is systematically related to those used for trained metamappings. We view this as an important part of intelligent adaptability—the system should be able not only to adapt to tasks via metamappings that it has directly experienced, but also to infer and use novel metamappings based on specific instructions or examples. We demonstrate this ability in the subset of our experimental domains where we can instantiate sufficiently many metamappings.

Experiments

Metamapping is an extremely general framework. Because the assumptions are simply that the basic tasks are mappings from inputs to outputs and that metamappings transform basic tasks, the approach can be applied to most paradigms of machine learning with minor modifications. We demonstrate our results in four experimental domains. We summarize the contributions of each domain in Table 1.

Polynomials. As a proof of concept we first apply metamapping to polynomial regression (Fig. 2). We construct basic tasks that are polynomial functions (of degree ≤ 2) in four variables (i.e., from $\mathbb{R}^4 \rightarrow \mathbb{R}$). These polynomials can be inferred from a support set of (input, output) examples, where the input is a point in $\mathbb{R}^4$ and the output is the evaluation of that polynomial at that point. For details, and to see that the system performs this simple metalearning regression problem extremely well, see *SI Appendix*, section B.1 and Fig. S4.

These basic tasks/polynomials can be transformed by various metamappings—we considered squaring a polynomial, permuting its variables, or adding or multiplying by a constant. We considered 36 metamappings in total, of which we trained the model to perform 20, and held out the remaining 16 to evaluate the model's ability to generalize to held-out metamappings (see *Evaluating Base-Task and Metamapping Performance*). The held-out metamappings included some of the possible permutation, addition, and multiplication transformations. We used 60 example (source polynomial, transformed polynomial) mapping pairs as a training set for each metamapping and held out another 40 transformed polynomials per metamapping for evaluation. The source and transformed polynomials for all 60 example pairs were trained for each trained or held-out metamapping. This results in a total of 2,260 polynomials trained and 1,440 held out for evaluation. For the 20 trained metamappings, the 60 trained (source polynomial, transformed polynomial) pairs were used to train the metamapping and as the support set for evaluation. For the 16 held-out metamappings, these pairs were used only as the support set for evaluation.

In Fig. 3, we show the success of our metamapping approach in this setting. We plot a normalized performance measure, $100\%(1 - \text{loss}/c)$, where the loss is the mean squared error, and c is the loss for a baseline model that always outputs zero. This measure is 0% for a model which outputs all zeros and 100% if the system performs perfectly. See *SI Appendix*, Table S4 for raw losses. Metamapping achieves good performance on the support set examples that are used to instantiate the mapping, with 98.3% performance (bootstrap 95% CI across runs [97.3, 99.0]) on trained metamappings and 92.1% [91.3, 93.0] on held-out metamappings. More importantly, on

Table 1. The contributions of our four experimental domains

Domain	Held-out MMs	Lang. comp.	Type	Input	Output
Polynomials	✓		Regression	Vector ($\mathbb{R}^4$)	Scalar ($\mathbb{R}$)
Cards		✓	Regression	Binary features	Bet values ($\mathbb{R}^3$)
Visual concepts	✓	✓	Classification	50×50 image	Label ($\{0, 1\}$)
RL		✓	RL	91 × 91 image	Action Q values ($\mathbb{R}^4$)

Our results span various computational paradigms and data types. (Note that "Held-out MMs" refers to held-out metamappings, and "Lang. comp." refers to a comparison to language alone; see *Language and Metamapping* section.)

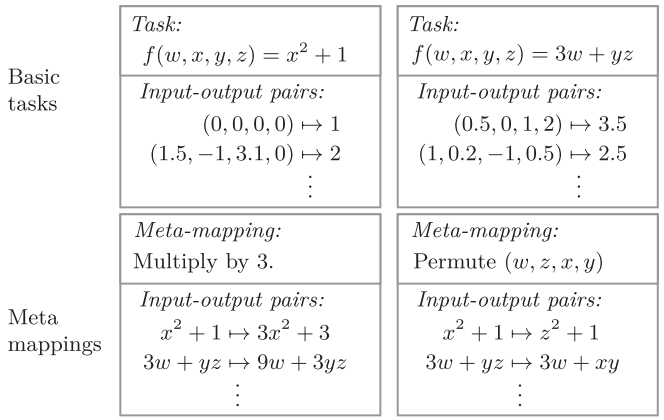


Fig. 2. The polynomial task domain. A basic polynomial task consists of regressing a single polynomial; i.e., the inputs are points in $\mathbb{R}^4$ and the outputs are the value of the polynomial at that point. These basic regression tasks can be transformed by various metamappings, such as multiplying by a constant, or permuting their variables.

polynomials never experienced during training, metamapping achieves 89.0% [89.3, 89.8] zero-shot performance on average based on a trained metamapping and 85.5% [85.1, 85.9] performance based on a held-out metamapping. We also show the performance the model obtains when it is scored on the new task using the untransformed source task representation (no adaptation). This baseline yields only 4.3 and 19.3% performance, respectively. In summary, metamapping is able to achieve good performance on a new task without any data, based only on its relationship to prior tasks. This success is consistent across all of the metamapping types we evaluated (*SI Appendix*, Fig. S6).

We show in Fig. 4 that polynomial and metamapping representations are systematically organized and transform in systematic ways. In general, the transformed representations are close to the nominal targets where targets are known. (Note that even missing the nominal target does not necessarily mean the model is incorrect; just as we could write $2(x+1)$ instead of $2x+2$, the model may have different representations for the same function.) The model is sample efficient at inferring both polynomials and metamappings (*SI Appendix*, Figs. S5 and S7).

Finally, our homoiconic approach significantly outperforms a nonhomoiconic baseline, which differs from the homoiconic architecture only in that separate example networks and hypernetworks are used for the basic tasks and metamappings (*SI Appendix*, Fig. S9). Sharing these networks improves generalization. Why is homoiconicity beneficial? We show that there is nontrivial overlap between the basic-task and metamapping representations (*SI Appendix*, Figs. S17–S19) and that some of this overlap reflects structural isomorphisms (*SI Appendix*, Fig. S20). While this may not fully explain the benefits of homoiconicity, it suggests that the model may exploit the shared structure between basic tasks and metamappings. By contrast, there is little alignment between the representations of numerical polynomial inputs and task representations, perhaps because there are fewer constraints encouraging such alignment.

Card Games. We motivated our work in part by observations about human flexibility, so we next compare our model to human adaptation in a simple card game. Performing a basic task consists of receiving a hand of two cards and making a bet. The human (or model) plays against an opponent and wins (or loses) the bet if the human's (or model's) hand beats (loses to) the opponent's.

We trained human participants to play one poker-like game with two-card hands (card rank 1 to 4, suit red or black). We evaluated their ability to play that game and then to switch strategy when told to try to lose. We evaluated on multiple trials without feedback, to get multiple "zero-shot" measurements from each participant. (Our protocol was approved by the Stanford University Institutional Review Board Panel on Human Subjects in Non-Medical Research, and all subjects provided informed consent. See *SI Appendix*, section C for details.)

We compare human adaptation to that of a metamapping model trained on poker and four other card games. The specific rules vary from game to game. We created eight variations of each game, by applying any subset of three transformations, each of which could be learned as a metamapping (*SI Appendix*, section B.2). The most dramatic transformation is switching from trying to win to trying to lose. This variation requires completely inverting the strategy. We trained the network on 36 of the 40 basic tasks; all losing variations of poker were held out. We used the learned task representations to train metamappings for each of the three transformations. Two of the metamappings were trained using all five games, but the lose metamapping was trained only on the games other than poker.

After training, the lose metamapping is applied to the task representation of poker, to transform it into a task representation of losing at poker. This representation is then used to play the losing variation of poker. This evaluation exactly matches the evaluation of the human participants.

Since rewards are observed only for the action taken, we must alter the representation of basic task examples. Instead of (input, target) examples we use (state (action, reward)) examples (*SI Appendix*, section A.4).

See Fig. 5 for the results. Human subjects are not optimal at the game (mean performance 64%, bootstrap 95% CI $[0.57, 0.70]$), but are adapting well, at least in the sense that performance is similar in the losing variation on average (losing phase mean performance 64%, bootstrap 95% CI $[0.55, 0.72]$). However, there is substantial intersubject variability in base task performance and adaptation. The evaluation hands were sampled in a stratified way in each phase, so this variability in adaptation is due to randomness either in participants' behavior (e.g., because they are probability matching rather than optimizing bets) or in the way that their behavior changes between

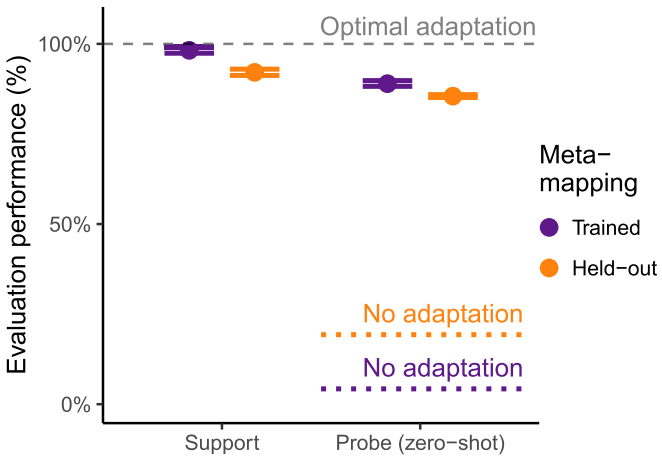


Fig. 3. Metamapping can adapt to a new polynomial zero shot, based on its relationship to prior polynomials. We plot performance (normalized, main text) on transformed polynomials via metamappings. The system performs well on support-set target tasks after adaptation. More importantly, it can perform probe target polynomials that it has never encountered before zero shot and does so substantially better than if it did not adapt (dotted lines). It generalizes well both on trained metamappings (purple) and on held-out metamappings (orange). (Dots show mean and lines show bootstrap 95% CI across five runs.)

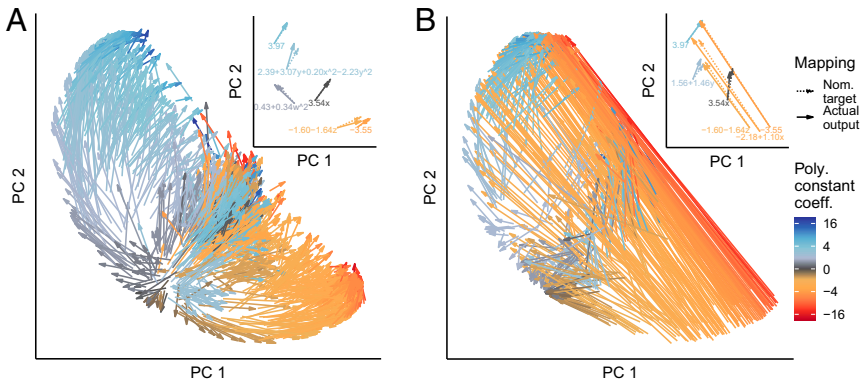


Fig. 4. Visualizing how metamappings systematically transform the model's polynomial representations. *A* and *B* show two metamappings: (*A*) multiplying by 3 and (*B*) squaring. Each arrow shows how a single polynomial's representation transforms under the metamapping. The arrows are colored by the constant term of the polynomial, and the representations are generally organized so that constant polynomials are around the outside, with constant value increasing clockwise, and the polynomials involving more variables are closer to the center. (*A*) Multiplying by 3 pushes polynomials away from the center, with the negative constant polynomials rotating counterclockwise as they become more negative and the positive constant polynomials rotating clockwise as they become more positive. The nonconstant polynomials extend in similar directions, but their trajectories are more complicated. (*B*) Squaring polynomials results in both rotation of the representation space and folding as the negative constants flip to being positive. *Insets* show that polynomial transformations align closely to their nominal targets (that is, the model's representation of the target task). (Plots show the top two principal components (PC 1 and PC 2). Note that only 60 of the 1,200 polynomials shown were used for training each mapping. See *SI Appendix*, section F.2 for further representation analysis.)

winning and losing phases. The metamapping model performs near optimally at the trained task and adapts quite well (mean 85%, 95% CI [79, 90]). In summary, the model performed differently than the human participants, but both the model and humans were able to switch from winning to losing zero shot. See *SI Appendix*, section F.3 for further analyses.

Visual Concepts. We next applied metamapping to visual concepts, a long-standing cognitive paradigm (18). Past work has focused almost entirely on learning a concept from examples. However, adult humans can also understand some novel concepts without any examples at all. If you learn that "blickets" are red triangles and then are told that "zipfs are cyan blickets," you will instantly be able to recognize a zipf without ever having seen an example. This zero-shot performance can be understood as applying a "switch-red-to-cyan" metamapping to the blicket classification function (Fig. 6). To capture this ability, we applied metamapping.

We constructed stimuli by selecting from eight shapes, eight colors, and three sizes. We rendered each item at a random position and rotation within a 50×50 pixel image. We defined the basic concepts (basic tasks) as binary classifications of images (i.e., functions from images to $\{0, 1\}$). We trained the system on all unidimensional concepts (i.e., one-vs.-all classification of each shape, color, and size) as basic tasks, so that it could learn all of the basic attributes. We also constructed composite basic tasks based on conjunctions, disjunctions, and exclusive disjunctions (XOR) of these attributes. For example, one composite concept might be "red and triangle."

For each concept, we chose balanced datasets of examples (that is, there was a 50% chance that each stimulus was a member of the category), during both training and evaluation. We included negative examples that were only one alteration away from being a category member. These careful contrasts can encourage neural networks to extract more general concepts (19).

In this domain we constructed both the basic task and metamapping representations from language rather than examples (Fig. 1 *B* and *E*), to show that metamapping can use this human-relevant cue. That is, there is no example network; instead a language network processes descriptions of tasks and metamappings to construct task and metamapping representations.

We trained the system on metamappings that switched one shape for another or one color for another. We sampled six composite concept transformation pairs that supported each mapping and another six with held-out targets for evaluation. However, our task sampling meant that each held-out example had a closely matched trained example, unlike the other experimental domains. See *SI Appendix*, section B.3 for details of sampling.

We varied the number of metamappings trained and evaluated the system on its ability to apply metamappings to trained source concepts to recognize the held-out target concepts. (Note that we exclude disjunctions from evaluation, because not

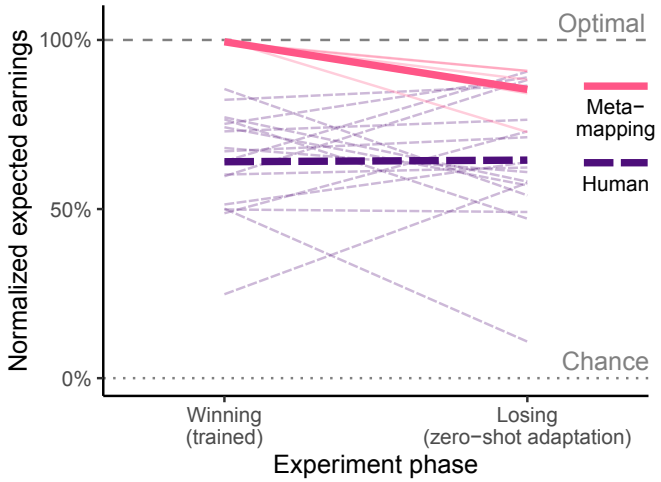


Fig. 5. Comparing metamapping to human adaptation in simple card games. This plot shows performance in the two phases of the experiment: baseline testing on the basic game and after adapting to losing zero shot. Human participants are behaving suboptimally on average, but are achieving similar performance after adaptation, although there is substantial inter- and intrasubject variability. The model performs near optimally at baseline and by metamapping achieves around 90% performance at the new game. (We plot performance as expected earnings from the bets made, as a percentage of the expected earnings of an optimal policy. Thick lines are averages, and thin lines are five runs of the model and 19 individual participants.)

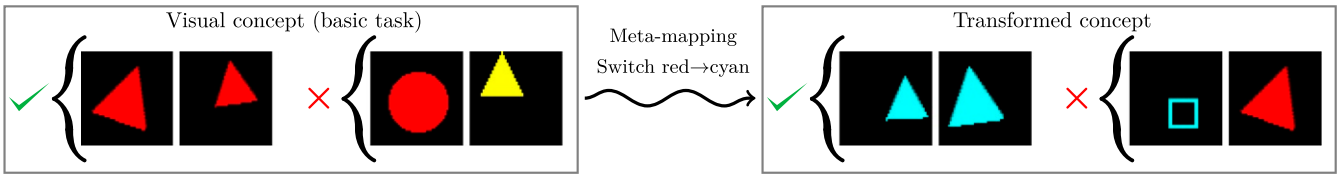


Fig. 6. The visual concepts domain. Concepts consist of mappings from images to binary labels, e.g., 1 for images that are red and triangle, 0 otherwise, for example for a red circle or yellow triangle. These concepts can be transformed by metamappings that alter their attributes, such as switching red to cyan.

adapting works fairly well on them.) Because there are many metamappings available, we were able to hold out one shape metamapping and one color metamapping for evaluation. The same basic concepts instantiating a held-out metamapping were trained as would be for a trained mapping, but the metamapping itself was not. This reduces possible confounds when evaluating metamapping generalization.

The model generalizes well (Fig. 7). On trained metamappings, its performance reaches close to ceiling around 12 training mappings. Furthermore, given enough training metamappings it is able to generalize well to held-out metamappings from a language description of that metamapping. This generalization improves rapidly as the number of metamappings trained increases. Although the average held-out metamappings performance is not perfect even at 32 training metamappings, it is perfect in 40% of the runs.

Reinforcement Learning. We next apply our approach to reinforcement learning (RL). RL-like computations relate to neural activity (20, 21), and RL has driven recent artificial intelligence achievements in complex tasks like Go and StarCraft (22, 23). Furthermore, RL requires sophisticated adaptation, since actions have lasting consequences. Thus, RL is an important testing domain for metamapping.

Our RL tasks consist of simple two-dimensional games (Fig. 8), which take place in a 6×6 room with an additional impassable barrier of one square on each side. This grid is rendered at a resolution of 7 pixels per square to provide visual input to the agent. The agent receives egocentric input; i.e., its view is always centered on its position. This improves generalization (8). The agent can take four actions, corresponding to moving in the four cardinal directions. Invalid actions, such as trying to move onto the edge of the board, do not change the state.

The tasks the agent must perform relate to objects that are placed in the room. The objects can appear in 10 different colors. In any given task, the room only has two colors of objects in it. Each color of objects appears only with one other color, so there are in total five possible color pairs that can appear. In any given task, one of the present colors is "good" and the other is "bad." On some tasks, the good and bad colors in a pair are switched.

There are two types of tasks, a "pick-up" task and a "push-off" task. In the pick-up task, the agent is rewarded for moving to the grid location of each good object, which then disappears, and is negatively rewarded for moving to the location of bad objects. In the push-off task, the agent is able to push an adjacent object by moving toward it, if there is no other object behind it. The agent is rewarded for pushing the good-colored objects off the edges of the board and negatively rewarded for pushing the bad-colored objects off. The two types of tasks (pick up and push off) are visually distinguishable, because the shapes of the objects used for them are different. However, which color is good or bad is not visually discernible and must be inferred from the example (state, (action, reward)) tuples used to construct the task representation.

There are in total (2 task types) $\times$ (5 color pairs) $\times$ (binary switching of good and bad colors) $= 20$ tasks. (See *SI Appendix*, section B.4 for details.) We trained the system on 18 tasks, holding out the switched color combinations of (red, blue) in both task types. That is, during training the agent was always positively rewarded for interacting with red objects and negatively rewarded for interacting with blue objects. We trained the system on the "switch-good-and-bad-colors" metamapping using the remaining four color pairs in both task types and then evaluated its ability to perform the held-out tasks zero shot based on this mapping. This evaluation is a difficult challenge, since the model was always negatively rewarded during training for interacting with the objects that it must interact with in the evaluation tasks.

We evaluate the model for each task by requiring the training accuracy to be above a threshold and selecting an optimal stopping time when the other task is performed well. We also used two minor model modifications to stabilize learning: persistent task representations (discussed in *Training the Model*) and weight normalization. See *SI Appendix*, section A.4 for details. Despite the challenging setting, the model adapts well, achieving 88.0% of optimal rewards (mean, bootstrap 95% CI [75.0–99.0]) on the held-out pick-up task and 71.7% (mean, bootstrap 95% CI [42.0, 94.6]) on the held-out push-off task. The results are plotted in Fig. 9, along with the results from the comparison models from the next section. (The model is also slower to complete generalization episodes [*SI Appendix*, Fig. S26]; perhaps humans, too, might be more hesitant in novel situations.)

In *SI Appendix*, section F.7, we show that metamapping is able to extrapolate metamappings beyond the dimensions it has been trained on, to transform new dimensions. Specifically, when trained with the switch-good-and-bad metamapping applied to colors, it can generalize to switching shapes. This is further evidence for the flexibility and systematicity of metamapping.

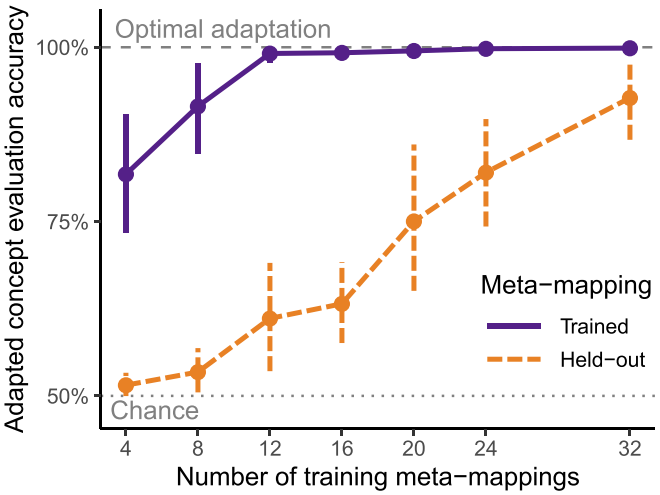


Fig. 7. Applying metamapping to visual concepts, after training the model on varying numbers of metamappings. The model is able to generalize trained metamappings to perform new tasks zero shot. Furthermore, it can generalize to new metamappings once it experiences sufficiently many training metamappings. (Results are from 10 runs with each training set size. Error bars are bootstrap 95% CIs across runs.)

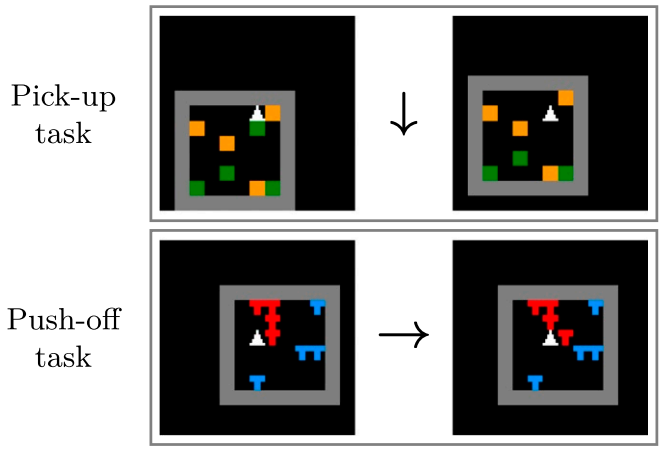


Fig. 8. Illustrative RL task state transitions. In the pick-up example (*Top*), the agent moves down and picks up the green object. In the push-off example (*Bottom*), the agent moves right and pushes the red object. Each image is precisely the visual input the agent would receive. Note that the agent is always centered (egocentric perspective).

Language and Metamapping. Language is often key to human adaptation, and prior work on zero-shot performance has often used a task description as input (6–8). We showed in the visual concepts domain that language provides a suitable cue for basic tasks and met-mappings; in this section we explore the relationship between language, examples, and metamapping further. We compare three approaches to zero-shot task performance in the RL domain: metamapping from examples (shown in the previous section), metamapping from language, and generalization from language alone.

First, we consider metamapping from language. We use language input both to generate task representations (e.g., "pick-up, red, blue, first" to indicate picking up objects, where the first color, red, is good) and as a cue for metamapping ("switch colors"). Applying this approach to the same training and hold-out setup used above for metamapping from examples yields comparable performance: 69.2% (mean, bootstrap 95% CI [49.5, 84.5]) on the pick-up task and 74.9% [60.9, 85.5] on the push-off task (Fig. 9). This shows (as with the visual concepts) that generating task representations from examples is not essential—language can support metamapping.

However, a model that generates task representations from language offers an alternative approach to performing a new task zero shot. If language descriptions systematically relate to tasks, the model should be able to generalize to new tasks from their description alone. If the system learns that "green, yellow, first" means that the objects will be green and yellow, and the first color (green) is good; that "green, yellow, second" means that yellow will be good; and that "red, blue, first" means that red will be good and blue bad; it could in principle generalize appropriately to "red, blue, second." Indeed, this approach to zero-shot task performance has been demonstrated in prior work (7, 8). However, we find that transforming the task representation via a metamapping can provide a stronger basis for adapting, compared to systematic language alone.

To demonstrate this, we compare the example- and language-based metamapping approaches to generalizing from language alone, again using the same basic tasks to train the network to perform tasks from language, but without metamapping training (Fig. 9). Performing the new tasks from language alone results in very poor generalization performance: −92.8% (mean, bootstrap 95% CI [−96.3, −88.4]) on the pick-up task and −79.7% [−92.8, −59.1] on the push-off task. Metamapping provides much better generalization.

The direct comparison between language-based metamapping and language alone shows that metamapping is beneficial, but there are two mechanisms by which it could help. Metamapping at test time could be key to generalization, or metamapping training could simply improve the learning of the basic task representations, such that even language alone would allow good generalization in a metamapping trained model. However, language-alone generalization is not significantly improved even in the language-based metamapping model (*SI Appendix*, section F.6), suggesting that metamapping at test time is key to the benefits we observe.

We also compared metamapping to language alone in the cards and visual concepts domains. We summarize the results here; see *SI Appendix*, section F.6 for details. In the cards domain, the language-based model was not able to generalize well to the losing game, instead degrading to chance-level performance. In the visual concepts domain, by contrast, the language model generalizes comparably to metamapping. This may be due to the concept sampling—each evaluation concept had several closely related training concepts, unlike the other domains. Indeed, metamapping shows a greater advantage when new concepts are less similar to trained ones.

In summary, metamapping (from examples or language) outperforms or equals language alone in all our experiments. Metamapping is especially beneficial when the task space is sparsely sampled or generalization is challenging. We consider the advantage of metamapping further in *Discussion*.

Metamapping as a Starting Point for Later Learning. Zero-shot adaptation by metamapping allows a model to perform a new task without any direct experience. However, as we have seen, zero-shot performance is not always as good as the ultimate performance after training on the task. Here, we show that even if zero-shot performance is not completely optimal, it makes learning much faster than starting from scratch. We also show that this learning can be done in a way that avoids interference with performance on prior tasks.

We return to the polynomials domain to demonstrate this. We reinstate a trained model and consider how it could learn on the held-out tasks once it encounters them. To do so, we optimize the representations of the new tasks to improve performance

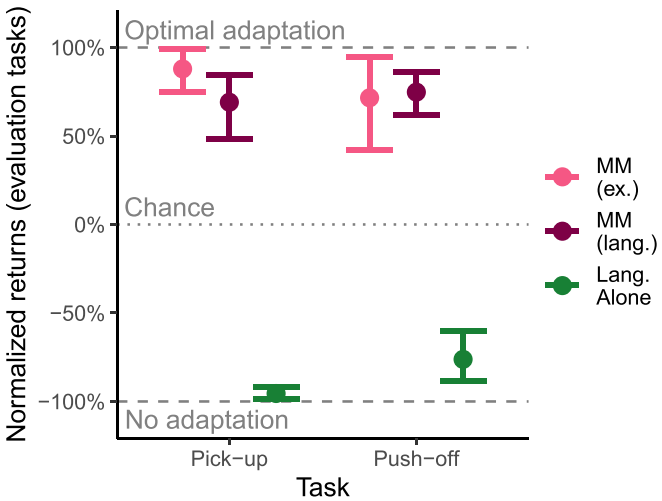


Fig. 9. Comparing RL adaptation performance when metamapping (MM) with task representations constructed from examples, when metamapping with task representations constructed from language, or when generalizing from language alone. Metamapping generalizes well with either type of task representation, while language alone generalizes poorly. (Chance refers to taking random actions.)

on those tasks, without allowing any of the network weights to change (*SI Appendix*, section A.5). This approach improves performance on the new tasks without interfering with prior knowledge (24) (cf. refs. 25 and 26). Thus, it provides a useful approach to learning after zero-shot adaptation, once the system is actually performing the new tasks.

We evaluate a variety of starting points for initializing the new task representations. We compare initializing via metamappings to a variety of reasonable alternatives, such as small random values (the standard in machine learning), the embedding of an arbitrary trained task, and the centroid of all trained task representations. We plot learning curves from these different initializations in Fig. 10. Producing an initial task representation by metamapping results in much lower initial loss and faster learning than any other method.

To quantify this, we consider the cumulative loss over learning, i.e., the integral of the learning curves. This measures how much loss the model had to suffer to reach perfect behavior on the new tasks. Starting from a metamapping results in almost an order of magnitude less cumulative error (mean $= 24.58$, bootstrap 95% CI $[17.71, 32.08]$) than the next best initialization (centroid of trained task representations, mean $= 192.89$, bootstrap 95% CI $[151.98, 234.53]$). Metamapping provides a valuable starting point for future learning. (We also show this in the visual concepts domain in *SI Appendix* and show that a hypernetwork architecture is essential.)

Discussion

We have proposed metamappings as a computational mechanism for performing a novel task zero shot—without any direct experience on the task—based on the relationship between the novel task and prior tasks. We have shown that our approach performs well across a wide range of settings, often achieving 80 to 90% performance on a new task with no data on that task at all. With enough experience, as in the visual classification settings with enough training tasks, it can adapt perfectly. It can also adapt using novel relationships (held-out metamappings) that it has never encountered during training.

As noted in the Introduction, there are computational benefits to adaptivity. Its potential contributions to biological intelligence have been highlighted by Siegelmann (1), who proposes that there is a "hierarchy of computational powers" and that a particular system's location in that hierarchy depends on its "particular level of richness and adaptability." Because our work offers an additional perspective on adaptation, it would be interesting to explore the theoretical computational power of metamapping under different input and representation regimes.

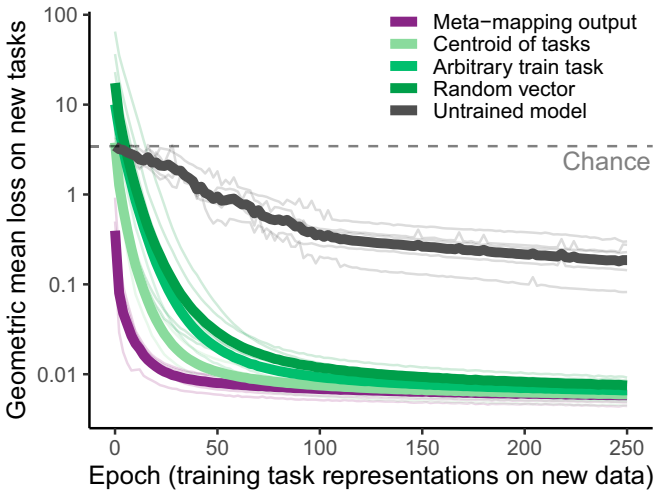


Fig. 10. Metamapping provides a good starting point for later learning. Shown are learning curves (geometric mean of loss on new tasks) while optimizing task representations on new tasks in the polynomials domain. Using metamapping as a starting point offers much lower initial loss and results in faster learning than alternative initializations. (Thin curves are five individual runs, thick curves are averages.)

As Siegelmann notes, for a model to be able to adapt, it must first be capable of performing a variety of related tasks (1). Thus, instead of learning parameters that execute a single task, our model learns to construct task representations from examples or language and to use those representations to perform appropriate behaviors. The key insight of this work is that those task representations are then available for transformation and that transforming task representations by metamappings can allow effective adaptation.

In our experiments, directly exploiting task relationships by metamapping allowed more systematic adaptation than indirectly exploiting them by generalizing through compositional language alone. Even when language alone generalized poorly, as in the RL domain, metamapping with language-based task representations resulted in strong generalization. This illustrates the value of a transformation-oriented perspective.

Why is transforming tasks according to task relationships so effective? We suggest that this is because metamapping constructs and uses an explicit cognitive operation that captures what is systematic in the task relationships. For example, "trying to lose" is systematic precisely insofar as the relationship between winning and losing is similar across different games. The metamapping approach gives primacy to these relationships. It thus directly exploits systematic structure where it exists in the cognitively meaningful relationships between tasks.

We also highlight the results showing that metamapping provides a useful starting point for later learning. While metalearning approaches (27) can construct a good starting point for learning new tasks, they do not use task relationships to offer a uniquely appropriate starting point for each novel task. Our results show that using a task relationship to adapt a prior task can substantially reduce the errors made along the way to mastering the new task. This could make deep learning more efficient. It could also be useful in settings like robotics, where mistakes during learning can be extremely costly (28).

Our results should not be taken as a suggestion that metamapping is the only possible mechanism for adaptation. We see intelligence as multifaceted, and any single model is a simplification. Metamapping may be useful as one tool for building models with greater flexibility.

Metamapping increases the adaptability of our models, although our present work has limitations that we discuss in *Limitations and Future Directions*. Our models can perform tasks from examples, from natural language, and from metamappings, which we have shown are an effective way to adapt zero shot. Thus, our work has many potential applications in machine learning and cognitive science.

Related Work in Machine Learning. To allow zero-shot adaptation, we built on ideas from several areas of machine learning. First, there is a large body of prior work on allowing models to learn to behave more flexibly, for example by metalearning, that is, learning to learn from examples (9, 27, 29). Our approach to inferring tasks from examples draws on recent ideas like aggregating examples in a permutation-invariant way to produce a task representation (12).

Second, a range of work uses the idea of different timescales of weight adaptation—that is, even if some parameters of a network may need to be learned slowly, it may be useful to alter others much more rapidly (30). We have drawn particularly on the idea that the parameters of a network could be specified by another network in a single forward inference (13, 14). This approach has

shown success in metalearning recently (10, 31) and improved our model's adaptation (*SI Appendix*, Fig. S9).

There has been a variety of other work on zero-shot task performance. We compared to the zero-shot task performance from language alone. The idea of performing tasks from descriptions was proposed by Larochelle et al. (6). More recent work has considered zero-shot classification using language (32, 33) or performing tasks from language in RL (7, 8). Some of this latter work has even exploited relationships between tasks as a learning signal (11), but without transforming task representations. As discussed in the beginning of the *Discussion*, transforming task representations with metamappings directly exploits systematic relationships, allowing metamapping to outperform language alone in our experiments. To our knowledge none of the prior work has proposed task transformations to adapt to new tasks.

Other prior work has used similarity between tasks to help generate representations for a new task (34). Again, metamapping may be a stronger approach, since it can specify along which dimensions two tasks are related and the specific ways in which they differ, which a scalar similarity measure cannot.

Aspects of zero-shot adaptation have also been explored in model-based reinforcement learning. Work in model-based RL has partly addressed how to transfer knowledge between different reward functions (35). Metamapping can potentially be applied to this form of transfer as well; indeed, our RL experiments show that metamapping can offer a model-free alternative to model-based adaptation. Metamapping may also offer advantages that could complement model-based methods. Metamapping provides a principled way to infer a new reward estimator by transforming a prior one. It could also transform a transition function used in the planning model in response to environmental changes. Thus, exploring the relationship and synergies between metamapping and model-based RL methods provides an exciting direction for future work.

There has also been other recent interest in task representations. Achille et al. (36) proposed computing embeddings for visual tasks from the Fisher information of a task-tuned model. They show that this captures some interesting properties of the tasks, including some semantic relationships, and can help identify models that can perform well on a task. Other recent work has tried to learn representations for skills (37) or tasks (38) for exploration and representation learning, but without exploring zero-shot transformation of these skills.

Related Work in Cognitive Science. Our work is related to several streams of research in cognitive science. Prior work has suggested that analogical transfer between structurally isomorphic domains may be a key component of "what makes us smart" (39). Analogical transfer is a kind of zero-shot mapping and has been demonstrated across various cognitive domains (18, 40). We hope our work stimulates further exploration of the conditions under which humans can adapt to task transformations zero shot. Different types of task relationships might be made accessible through culture or education—"relational concepts are not simply given in the natural world: they are culturally and linguistically shaped" (ref. 39, pp. 204–206).

Our work also touches on complex issues of compositionality, productivity, and systematicity. Fodor (41, 42) and Lake and Baroni (43) have advocated that cognition must use compositional representations to exhibit systematic and productive generalization. We see our work as part of an alternative approach to this issue, exploring how systematic, structured generalization can instead emerge from the structure of learning experience, without needing to be built in (44, 45). By focusing on task relationships, rather than building in compositional representations of tasks, our model can learn to exploit the shared structure in the concept of "losing" across a few card games to achieve 85% performance in losing a game it has never tried to lose before.

Crucially, the question of whether the model adapts according to compositional task structure is distinct from the question of whether the model's representations exhibit compositional structure. Because the mapping from task representations to behavior is highly nonlinear, it is difficult to craft a definition of compositional representations that is either necessary or sufficient for generalization. For example, if "compositional" is taken to mean that Euclidean vector addition of the representations of two constant polynomials results in the representation of their sum, this is clearly untrue for our model (*SI Appendix*, Fig. S13). However, the nonlinear mapping from representations to behavior can allow for systematic generalization from nonlinear structure. Indeed, it appears that the constant polynomial representations may be approximately systematically arranged in a compressed polar coordinate system. This may support generalization better than a more intuitively compositional representational structure.

Furthermore, there are a number of potential benefits to letting systematic behavior emerge, rather than attempting to build in compositional representations. First, the structure does not need to be hand engineered separately for each domain. Our system required no special knowledge about the domains beyond the basic tasks and the existence of relationships between them. The fact that some of these relationships corresponded to, e.g., permutations of variables in the polynomial domain did not need to be hard coded; instead, the model was able to discover the nature of this transformation from the data (in that it was able to generalize well to held-out permutations). Emergence may also allow for novel decompositions at test time. The ability of our model to perform well on held-out metamappings indicates that it has some promise in this regard. Future work should assess this capability of the model more fully.

We also believe that our approach can capture some of the recursive processing that Fodor (42) and others have emphasized. We have also been influenced by ideas in mathematical cognition about how concepts build upon more basic concepts (15–17). This recursive construction reflects the way that metamappings transform basic tasks—complex transformations are built upon simpler ones. If humans can handle an indefinite number of levels of abstraction, the advantage of using a shared representational space for all levels increases, since it eliminates the need to create a new space for each level. Relatedly, our shared workspace for data points, tasks, and metamappings connects to ideas like the global workspace theory of consciousness (46). The ability to reason about and explain concepts at different levels of abstraction can be explained parsimoniously by assuming a shared representational space. Exploring these connections would be an exciting future direction.

We found particular inspiration in Karmiloff-Smith's (47) and Clark and Karmiloff-Smith's (48) work on rerepresenting knowledge. It would be interesting to explore modeling the phenomena they considered, which they argued required that representations be "objects for further manipulation" (ref. 48, p. 509), as task representations are in metamapping.

Our work also relates to Fodor's (49) ideas about the modularity of the mind. Indeed, our division of the architecture into input and output systems, with the flexible, task-specific computations in the middle, may seem very reminiscent of the modularity that he advocated. However, we chose this implementation for simplicity—we believe that in reality processes such as perception can be influenced by the task, as well as contextual constraints (50).

Reciprocally, we believe that higher-level computations are influenced and constrained by the modalities in which they are supported. This computational feature can emerge in our model; despite the fact that different types of data and tasks are embedded in a shared latent space, the model generally learns to organize distinct types of inputs into somewhat distinct regions

of this space. This means that the task-specific processing can potentially exploit domain-specific features of the input, as for example humans do when they use gestures to think and learn in spatial contexts like mathematical reasoning (51). At the same time, the shared space can allow a graded overlap in the structure that is shared across different entities, insofar as they are related to each other. For example, in the polynomial domain there is more overlap between polynomial representations and metamapping representations than between either type of representations and the representations of numerical inputs. Using a shared space allows the model to discover what should be shared and what should be separated—that is, modularity "may not be built in [but] may result from the relationship among representations" (ref. 52, p. 231).

Finally, our approach relates to work on cognitive control (5). The "default" task-network weights could be used to model more automatic processing. This processing can be overridden by task-specific constraints set by the HyperNetwork, when conditioned on an appropriate task representation. We present a simple demonstration of these ideas in *SI Appendix*, section F.9. Metamapping itself could also be relevant; for example, an imperfect metamapping might capture some failures of control.

Limitations and Future Directions. Although we believe our approach is promising, the present work has limitations. We have explored metamapping within a limited range of settings. While we used one particular model, metamapping could potentially be useful in any metalearning approach that uses task representations (10). Furthermore, we have demonstrated our model only within relatively simple, small domains. The model adapts quite well, but does not always achieve perfect fidelity of adaptation. One factor that may contribute is the relatively limited range of experience of the model—our models lack the rich lifetime of experience that our human participants have. Furthermore, recent work shows that more realistic and embodied environments can improve generalization (8). Thus, evaluating our approach in richer, more realistic settings will be an important future direction.

Another important limitation is that our approach requires the imposition of structured training to provide the network with experience of the relationships between tasks. However, we suggest that identifying task relationships is useful for building more flexible intelligent systems and that exposure to task relationships is an important part of human experience. A long-term goal would be to create a system that learns to identify task relationships for itself from such experience.

Our work suggests many other possibilities. For simplicity we considered using language, examples, and metamapping to infer task representations in this work. However, it would likely be beneficial to use multiple constraints to both infer and adapt task representations. Furthermore, we considered language as input, but producing language as output (in the form of explanations) can improve understanding and generalization in both humans (53) and neural networks (54). Adding language output would likely improve performance and better capture the structure of human behavior.

In addition, we did not thoroughly explore robustness and the effect of noise. We showed that our approach is reasonably robust to sample-size variability (*SI Appendix*, Figs. S5 and S7), but there is room for further exploration. For example, how would input noise affect the computations? How would errors compound if multiple metamappings were applied sequentially?

Our model architecture also has limitations; cognitive tasks often require more complex processing than our model allows. Replacing the feedforward task network with a recurrent or attentional network—or a network with external memory (55)—would increase the flexibility of the model. It will be important to incorporate these ideas in future work.

Conclusions

An intelligent system should be able to adapt to novel tasks zero shot, based on the relationship between the novel task and prior tasks. We have proposed a computational implementation of this ability. Our approach is based on constructing task representations and learning to transform those task representations through metamappings. We have also proposed a homogeneous implementation that reuses the same architectures for both basic tasks and metamappings. We see our proposal as a logical development from the fundamental idea of metalearning—that tasks themselves can be seen as data points in a higher-order task of learning to learn. This insight leads to the idea of transforming task representations just like we transform data.

Metamapping is an extremely general approach—we have shown that it performs well across several domains and computational paradigms, with task representations constructed from either examples or language. Metamapping is able to perform well at new tasks zero shot, even when the new task directly contradicts prior learning. It is generally able to adapt more effectively after experiencing fewer tasks than approaches relying on language alone and sometimes seems to exhibit more systematic behavior. We suggest that this is because task relationships better capture the underlying conceptual structure. Metamapping provides a valuable starting point for later learning, one that can substantially reduce both time to learn a new task and cumulative errors made in learning. Our results thus provide a possible mechanism for an advanced form of cognitive adaptability and illustrate the role it may play in future learning. We hope our work will lead to a better understanding of human cognitive flexibility and to the development of artificial intelligence systems that can learn and adapt more flexibly.

Data Availability. All study data are included in this article and *SI Appendix*.

ACKNOWLEDGMENTS. A.K.L. was supported by a National Science Foundation Graduate Research Fellowship. We appreciate helpful suggestions from Noah Goodman, Surya Ganguli, Felix Hill, Steven Hansen, Erin Bennett, Katherine Hermann, Arianna Yuan, Andrew Nam, Effie Li, and the anonymous reviewers.

1. H. T. Siegelmann, Turing on super-Turing and adaptivity. *Prog. Biophys. Mol. Biol.* **113**, 117–126 (2013).
2. B. M. Lake, T. D. Ullman, J. B. Tenenbaum, S. J. Gershman, Building machines that learn and think like people. *Behav. Brain Sci.* **40**, e253 (2017).
3. G. Marcus, Deep learning: A critical appraisal. arXiv:1801.00631 (2 January 2018).
4. J. Russin, R. C. O'Reilly, Y. Bengio, "Deep learning needs a pre-frontal cortex" in *ICLR Workshop on Bridging AI and Cognitive Science*. https://baicsworkshop.github.io/. Accessed 26 April 2020.
5. K. Dunbar, J. D. Cohen, J. L. McClelland, On the control of automatic processes: A parallel distributed processing account of the Stroop effect. *Psychol. Rev.* **97**, 332–361 (1990).
6. H. Larochelle, D. Erhan, Y. Bengio, "Zero-data learning of new tasks" in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, Eds. D. Fox, C. P. Gomes, Eds. (AAAI Press, Menlo Park, CA, 2008), pp. 645–651.
7. K. M. Hermann *et al.*, Grounded language learning in a simulated 3D world. arXiv:1706.06551 (26 June 2017).
8. F. Hill et al., "Environmental drivers of generalization in a situated agent" in *Proceedings of the 8th International Conference on Learning Representations*. https://openreview.net/pdf?id=SklGryBtwr. Accessed 5 May 2020.
9. O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, D. Wierstra, Matching networks for one shot learning. *Adv. Neural Inf. Process. Syst.* **29**, 3630–3639 (2016).
10. A. A. Rusu *et al.*, "Meta-Learning with latent embedding optimization" in *Proceedings of the 7th International Conference on Learning Representations* (International Conference on Learning Representations, 2019).
11. J. Oh, S. Singh, H. Lee, P. Kohli, "Zero-shot task generalization with multi-task deep reinforcement learning" in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup, Y. W. Teh, Eds. (Journal of Machine Learning Research, 2017), Vol. 70, pp. 2661–2670.

12. M. Garnelo *et al.*, "Conditional neural processes" in *Proceedings of the 35th International Conference on Machine Learning*, J. G. Dy, A. Krause, Eds. (Journal of Machine Learning Research, 2018), Vol. 80, pp. 1704–1713.
13. D. Ha, A. Dai, Q. V. Le, Hyper networks. Proceedings of the 5th International Conference on Learning Representations. https://openreview.net/references/pdf?id=BkXLhl7te. Accessed 15 June 2018.
14. J. L. McClelland, Putting knowledge in its place : A scheme for programming parallel processing structures on the fly. *Cogn. Sci.* **146**, 113–146 (1985).
15. U. Wilensky, "Abstract meditations on the concrete and concrete implications for mathematics education" in *Constructionism*, I. Harel, S. Papert, Eds. (Ablex Publishing, 1991). https://ccl.northwestern.edu/papers/concrete/. Accessed 25 March 2020.
16. O. Hazzan, Reducing abstraction level when learning abstract algebra concepts. *Educ. Stud. Math.* **40**, 71–90 (1999).
17. A. K. Lampinen, J. L. McClelland, Different presentations of a mathematical concept can support learning in complementary ways. *J. Educ. Psychol.* **110**, 664–682 (2018).
18. L. E. Bourne, Knowing and using concepts. *Psychol. Rev.* **77**, 546–556 (1970).
19. F. Hill, A. Santoro, D. Barrett, A. Morcos, T. Lillicrap, "Learning to make analogies by contrasting abstract relational structure" in *Proceedings of the 7th International Conference on Learning Representations* https://openreview.net/pdf?id=SylLYsCcFm. Accessed 14 June 2019.
20. Y. Niv, Reinforcement learning in the brain. *J. Math. Psychol.* **53**, 139–154 (2009).
21. W. Dabney *et al.*, A distributional code for value in dopamine-based reinforcement learning. *Nature* **577**, 671–675 (2020).
22. D. Silver *et al.*, Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
23. O. Vinyals *et al.*, Grandmaster level in Starcraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019).
24. S. Reed, N. de Freitas, "Neural programmer-interpreters" in *Proceedings of the 4th International Conference on Learning Representations* https://arxiv.org/pdf/1511.06279.pdf. Accessed 5 September 2017.
25. T. T. Rogers, J. L. McClelland, *Semantic Cognition: A Parallel Distributed Processing Approach* (MIT Press, 2004).
26. A. K. Lampinen, J. L. McClelland, One-shot and few-shot learning of word embeddings. arXiv:1710.10280 (27 October 2017).
27. C. Finn, P. Abbeel, S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks" in *Proceedings of the 34th Annual Conference on Machine Learning*, D. Precup, Y. W. Teh, Eds. (Journal of Machine Learning Research, 2017), Vol. **70**, pp. 1126–1135.
28. M. Turchetta, F. Berkenkamp, A. Krause, Safe exploration in finite Markov decision processes with Gaussian processes. *Adv. Neural Inf. Process. Syst.* **29**, 4312–4320 (2016).
29. A. Ravichandran, R. Bhotika, S. Soatto, Few-shot learning with embedded class models and shot-free meta training. arXiv:1905.04398 (10 May 2019).
30. G. E. Hinton, D. C. Plaut, "Using fast weights to deblur old memories" in *Proceedings of the 9th Annual Conference of the Cognitive Science Society* (Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1982), pp. 177–186.
31. H. Li *et al.*, "LGM-Net: Learning to generate matching networks for few-shot learning" in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri, R. Salakhutdinov, Eds. (Journal of Machine Learning Research, 2019), pp. 3825–3834.
32. R. Socher, M. Ganjoo, C. D. Manning, A. Y. Ng, "Zero-shot learning through cross-modal transfer" in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Q. Weinberger, Eds. (Neural Information Processing Systems Foundation, 2013), pp. 935–943.
33. Y. Xian, C. H. Lampert, B. Schiele, Z. Akata, Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly. *IEEE Trans. Pattern Anal. Mach. Intell.* **41**, 2251–2265 (2018).
34. A. Pal, V. N. Balasubramanian, " Zero-shot task transfer" in *Proceedings of the 2019 IEEE Conference on Computer Vision and Pattern Recognition*, A. Gupta, D. Hoiem, G. Hua, Z. Tu, Eds. (IEEE, 2019), pp. 2189–2198.
35. R. Laroche, M. Barlier, "Transfer reinforcement learning with shared dynamics" in *Proceedings of the Thirty First AAAI Conference on Artificial Intelligence*, S. Singh and S. Markovitch, Eds. (AAAI Press, Palo Alto, CA, 2017), pp. 2147–2153.
36. A. Achille *et al.*, Task2Vec: Task embedding for meta-learning. arXiv:1902.03545 (10 February 2019).
37. B. Eysenbach, A. Gupta, J. Ibarz, S. Levine, "Diversity is all you need: Learning skills without a reward function" in *Proceedings of the 7th International Conference on Learning Representations*. https://openreview.net/pdf?id=SJx63jRqFm. Accessed 9 May 2019.
38. K. Hsu, S. Levine, C. Finn, "Unsupervised learning via meta-learning" in *Proceedings of the 7th International Conference on Learning Representations* https://openreview.net/pdf?id=r1My6sR9tX. Accessed 9 May 2019.
39. D. Gentner, "Why We're So Smart" *in Language in Mind: Advances in the Study of Language and Thought*, D. Gentner, S. Goldin-Meadow, Eds. (MIT Press, Cambridge, MA, 2003), pp. 195–235.
40. M. L. Gick, K. J. Holyoak, Analogical problem solving. *Cogn. Psychol.* **12**, 306–355 (1980).
41. J. A. Fodor, Language, thought and compositionality. *Mind Lang.* **16**, 1–15 (2001).
42. J. A. Fodor, *LOT 2: The Language of Thought Revisited* (Oxford University Press, 2008).
43. B. M. Lake, M. Baroni, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks" in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri, R. Salakhutdinov, Eds. (Journal of Machine Learning Research, 2018), pp. 2873–2882.
44. J. L. McClelland *et al.*, Letting structure emerge: Connectionist and dynamical systems approaches to cognition. *Trends Cognit. Sci.* **14**, 348–356 (2010).
45. S. S. Hansen, A. Lampinen, G. Suri, J. L. McClelland, Building on prior knowledge without building it in. *Behav. Brain Sci.* **40**, e268 (2017).
46. B. J. Baars, Global workspace theory of consciousness: Toward a cognitive neuroscience of human experience. *Prog. Brain Res.* **150**, 45–53 (2005).
47. A. Karmiloff-Smith, From meta-processes to conscious access: Evidence from children's metalinguistic and repair data. *Cognition* **23**, 95–147 (1986).
48. A. Clark, A. Karmiloff-Smith, The cognizer's innards: A psychological and philosophical perspective on the development of thought. *Mind Lang.* **8**, 487–519 (1993).
49. J. A. Fodor, *The Modularity of Mind* (MIT Press, 1983).
50. J. L. McClelland, D. Mirman, D. J. Bolger, P. Khaitan, Interactive activation and mutual constraint satisfaction in perception and cognition. *Cogn. Sci.* **38**, 1139–1189 (2014).
51. S. Goldin-Meadow, The role of gesture in communication and thinking. *Trends Cogn. Sci.* **3**, 419–429 (1999).
52. M. K. Tanenhaus, M. M. Lucas, Context effects in lexical processing. *Cognition* **25**, 213–234 (1987).
53. M. T. Chi, N. De Leeuw, M. H. Chiu, C. Lavancher, Eliciting self-explanations improves understanding. *Cogn. Sci.* **18**, 439–477 (1994).
54. J. Mu, P. Liang, N. Goodman, "Shaping visual representations with language for few-shot classification" in *Visually Grounded Interaction and Language Workshop, NeurIPS* (2019). https://vigilworkshop.github.io/2019. Accessed 13 December 2019.
55. A. Graves *et al.*, Hybrid computing using a neural network with dynamic external memory. *Nat. Publ. Group* **538**, 471–476 (2016).

PNAS
www.pnas.org

Supplementary Information for

Transforming task representations to allow models to perform novel tasks

Andrew K. Lampinen, James L. McClelland

[1]**To whom correspondence should be addressed. E-mail: andrewlampinen@gmail.com**

This PDF file includes:

Supporting Information Text

The Supporting Information is organized as follows: in Section A, we describe the details of the model, including providing a mathematical formulation, diagram of gradient flow, and architectural and hyperparameters for all experiments. In Section B we describe the different task domains and dataset sizes for our experiments. In Section C we describe the behavioral experiment that we performed on human adaptibity. In Section D we provide links to the repositories containing the code for all experiments and analyses. In Section F we show supplemental analyses, and in Section G we provide a proof that a simpler vector-analogy approach is insufficient for meta-mapping.

A. Model details, training, and methods. This section is organized as follows: in Section A.1 we give a formal (mathematical) description of the model, In Section A.2 we describe the architectural details and hyperparameters, and provide motivation for some of them. In Section A.3 we provide further details of model training and evaluation. In Section A.4 we provide details of the model modifications for the Cards and RL domains. Finally, in Section A.5 we provide details about the optimization of task representations for the **Meta-mapping as a starting point for later learning** experiments.

A.1. Mathematical formulation of the model. In this section we describe each of the networks in the system mathematically and give functional representations of each computation used in the model.

First, in Table S1 we remind the reader of the notation we use, and provide a mathematical characterization of each component, as well as its description. Given this notation, we next describe the computations of the model mathematically, with annotations indicating the meaning of key elements of each equation.

Constructing a basic task representation (from examples): Given a support set of (input, target output) examples tuples $\{(input_0, target_0), (input_1, target_1)...\}$, the representation would be computed as

$$z_{task} = \mathcal{E}(\ \{\ (\ \underbrace{\underbrace{\mathcal{P}(input_0)}_{\text{input embedding}\in Z}\ ,\ \underbrace{\mathcal{O}(target_0)}_{\text{target embedding}\in Z}}_{\text{(input embedding, target embedding) tuple}\in Z^2}\ )\ ,\cdots\}\ )$$

set containing encoded tuple for each support-set example

Constructing a basic task representation (from language): The representation would be computed as

$$z_{task} = \mathcal{L}(\ \underbrace{language}_{\text{description of task}}\ )$$

Performing a task from a representation: Given a task representation denoted by z_{task}, and an input, the output embedding ($z_{out} \in Z$) would be computed as:

$$z_{out} = \mathcal{T}(\ \underbrace{\mathcal{H}(z_{task})}_{\text{parameters}\in\Theta}\ ,\ \underbrace{\mathcal{P}(input)}_{\text{input embedding}\in Z}\ )$$

and the output would be computed as:

$$\text{output} = \mathcal{O}_d(z_{out})$$

Constructing a meta-mapping representation (from examples): Given a support set of (input task, target task) example tuples, the representation would be computed as follows:

$$z_{meta} = \mathcal{E}(\{\ \ (\ \underbrace{\underbrace{z_{inputtask_0}}_{\text{task embedding}\in Z}\ ,\ \underbrace{z_{targettask_0}}_{\text{task embedding}\in Z}}_{\text{(input embedding, target embedding) tuple}\in Z^2}\ )\ \ ,\cdots\})$$

set containing tuple for each mapping example

Constructing a meta-mapping representation (from language): The representation would be computed as

$$z_{meta} = \mathcal{L}(\ \underbrace{language}_{\text{description of meta-mapping}}\ )$$

Performing a meta-mapping from a representation: Given a meta-mapping representation z_{meta} and an input task representation z_{task}, the transformed task representation would be computed as:

$$z_{transformed\ task} = \mathcal{T}(\ \underbrace{\mathcal{H}(z_{meta})}_{\text{parameters}\in\Theta}\ , z_{task})$$

Symbol	Characterization	Description
input	Varies.	The input space for the base tasks, e.g $\mathbb{R}^4$ for polynomials, or RGB images for visual concepts.
output	Varies.	The output space for the base tasks, e.g $\mathbb{R}$ for polynomials, 4 action Q-values for the RL domain.
language	Varies.	A sentence of words from a discrete vocabulary.
Z	$\mathbb{R}^n$	The shared representational space used for representing inputs, tasks, etc.
Θ	$\underbrace{\underbrace{\mathbb{R}^{l_0 \times l_1}}_{\text{weights}} \times \underbrace{\mathbb{R}^{l_1}}_{\text{biases}}}_{\text{one layer's parameters}} \times \cdots$	The parameter space of the task-network $\mathcal{T}$, that is, the set of matrices (and vectors) representing the weights (and biases) of each layer of the MLP.

(a) Representation spaces.

Symbol	Characterization	Description
z_{input}	Representation $\in Z$	The representation of a base-task input (e.g. z_{hand} for a hand of cards), after it is processed by the perception network $\mathcal{P}$.
z_{output}		The representation of a base-task output (e.g. z_{bet} for a bet in the card game). This is processed by the output decoder $\mathcal{O}_d$ to produce the task output.
z_{target}		The representation of a base-task target output (e.g. a ground-truth classification of an image for a visual concept). This is processed by the target output encoder $\mathcal{O}_e$ to produce a target embedding for the input processor. Note that in the case of the cards and RL domains, the "target" is actually an (action, reward) tuple (see below).
z_{task}		Representation of a task. These are used to perform the task, and as inputs and outputs (and targets) of meta-mappings.
z_{meta}		Representation of a meta-mapping, used to perform that meta-mapping.

(b) Different types of representations in Z.

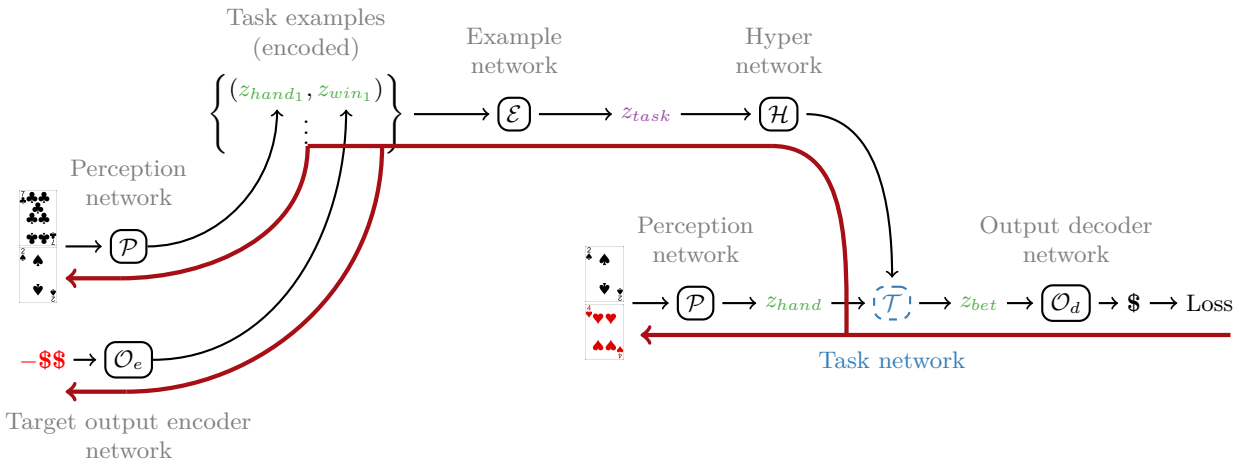
Symbol	Characterization	Description
$\mathcal{P}$	input $\rightarrow Z$	The perception network, an MLP (for polynomial and card tasks) or CNN (for visual concepts and RL tasks), which processes inputs into the shared representational space.
$\mathcal{O}_e$	target output $\rightarrow Z$	The target output encoder network, an MLP, which processes base-task example targets into the shared representational space. Note that in the case of the cards and RL models, these are not in fact outputs, but are rather (action, reward) tuples, see below.
$\mathcal{L}$	language $\rightarrow Z$	The language network, a multi-layer LSTM, which processes langauge into the shared representational space.
$\mathcal{E}$	$\{Z^2\} \rightarrow Z$	The example network, which processes a support set of tuples of (embedding of input, embedding of target output), and outputs a task representation in the shared representation space. This network consists of 1) parallel application of an MLP to each of the (input, output) tuples to produce a representation for each, 2) followed by max-pooling across that set of representations to produce a single representation, 3) followed by another MLP to produce the task representation.
$\mathcal{H}$	$Z \rightarrow \Theta$	The hyper network, an MLP, which maps a task representation to a set of parameters for the network $\mathcal{T}$.
$\mathcal{T}$	$\Theta \times Z \rightarrow Z$	The task network, which is an MLP parameterized by the parameter-space Θ. Once the parameters are specified, it serves as an MLP mapping $Z \rightarrow Z$.
$\mathcal{O}_d$	$Z \rightarrow$ output	The output decoder, an MLP mapping from the representational space Z to the output space for the task (e.g. $\mathbb{R}$ for the polynomials, and Q-values for the actions for the RL tasks).

(c) Networks.

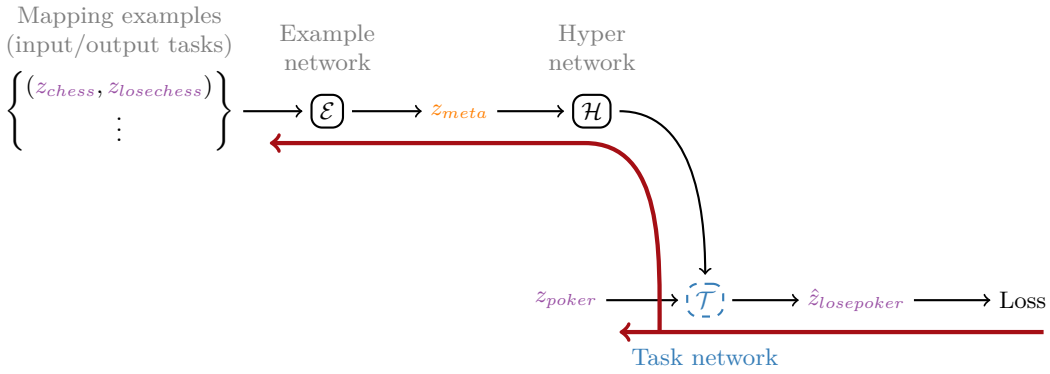
Table S1. Notation used for the (a) representation spaces, (b) types of representations in the shared space Z, (c) and networks in the paper.

	Polynomials	Cards	Visual	RL
Z-dimension	512			
$\mathcal{P}$ num. layers	2			
$\mathcal{P}$ num. hidden units	128			
$\mathcal{P}$ conv. layers. (num filters, size, all strides are 2)	-		(64, 5), (128, 4), (256, 4), (512, 2), max pool	(64, 7), (64, 4), (64, 3)
$\mathcal{L}$ architecture	-	2-layer LSTM + 2 fully-connected		
$\mathcal{L}$ num. hidden units	-	512		
$\mathcal{O}_e$ num. layers	1	3	1	3
$\mathcal{O}_e$ num. hidden units	-	128	-	128
$\mathcal{E}$ architecture	2 layers per-datum, max pool across, 2 layers			
Task, MM representations from	Examples		Language	Examples
$\mathcal{H}$ architecture	4 layers			
$\mathcal{E}$ num. hidden units	512			1024
$\mathcal{H}$ num. hidden units	512			
$\mathcal{T}$ num. layers	3	1	MM: 1, Lang: 3	3
$\mathcal{T}$ num. hidden units	64			128
$\mathcal{H}$ output init. scale	1	1	30	10
$\mathcal{T}$ weight norm. (1)	No			Yes
$\mathcal{O}_d$ num. layers	1		2	1
$\mathcal{O}_d$ num. hidden units	-		128	-
Nonlinearities	Leaky ReLU in most places, except no non-linearity at final layer of networks outputting to the latent space Z, and (where applicable) sigmoid for classification outputs, and softmax over actions.			
Base task loss	ℓ_2	ℓ_2 (masked)	Cross-entropy	ℓ_2 (masked)
Meta-mapping loss	ℓ_2			
Persistent task representations	No			Yes
Persistent embedding match loss weight	-			0.2
Optimizer	Adam	RMSProp		
Learning rate (base)	$3 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$3 \cdot 10^{-5}$	$1 \cdot 10^{-4}$
Learning rate (meta)	$1 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-4}$
L.R. decay rate (base)	$\times 0.85$	$\times 0.85$	$\times 0.8$	$\times 0.8$
L.R. decay rate (meta)	$\times 0.85$	$\times 0.9$	$\times 0.85$	$\times 0.95$
L.R. min (base)	$3 \cdot 10^{-8}$		$1 \cdot 10^{-8}$	$3 \cdot 10^{-8}$
L.R. min (meta)	$1 \cdot 10^{-7}$	$3 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$3 \cdot 10^{-7}$
L.R. decays every	100 epochs	200 epochs	400 epochs	10000
Num. training epochs	5000	100000 (optimally stopped)	10000 for 4 train mappings, 7500 for 8, 5000 for others	300000 (optimally stopped)
Num. runs	5	5	10	5
Base memory buffer size	1024		336	1000
Base memory buffers refreshed	Every 50 epochs		Every 20	Every 1500
Target network updated	-			Every 10000 epochs
RL discount	-			0.85
RL exploration probability (ϵ)	-			Initial: 1., decay: -0.03 when LR decays.
Action softmax inv. temp. (β)	-	8	-	8

Table S2. Detailed hyperparameter specification for different experiments. A "-" indicates a parameter that does not apply to that experiment. Where only one value is given, it applied to all the experiments discussed. See Table S1 for a guide to the notation for the networks.



(a) Basic task inference/training (from examples).



(b) Meta-mapping inference/training (from examples).

Fig. S1. Schematic of architecture, showing inference and gradient flow through the model on a training step. Thin black lines moving rightward represent inference, thick red lines moving leftward represent gradients. (a) Inference and gradients for the basic tasks. (b) Inference and gradients for meta-mappings. The gradients end at the examples of the meta-mapping, rather than propagating through to alter how those representations are constructed, due to GPU memory constraints. In the future, it might be useful to explore whether allowing further propagation would improve results for both basic tasks and meta-mappings. (These figures depict the inference/gradient flow when performing tasks and meta-mappings from examples, performing from language is similar, except that the example inputs and example network are replaced with language inputs and the language processing network.)

A.2. Model architecture & hyperparameters. See Table S2 for detailed architectural description and hyperparameters for each experiment (note that dataset sizes for the different different domains are specified in Table S3). Hyperparameters were generally found by a heuristic search, where mostly only the optimizer, learning rate annealing schedule, and number of training epochs were varied. Architectural parameters were generally chosen based on domain complexity (larger networks for more complex tasks, especially RL), and standard architectural practices.

For example, the convolution sizes and strides were generally chosen to result in reasonably even downsampling of the image, while also maintaining sizes divisible by powers of two (which can increase computational efficiency). The activation function chosen for the hidden layers of the MLPs was Leaky ReLU (leaky Rectified Linear Units), which are piecewise defined as

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.2x & \text{if } x < 0 \end{cases}$$

This function suppresses negative inputs (but does not completely shut them off). It has been shown to be useful for training deep networks (2).

Initialization scales for the HyperNetwork outputs were chosen based on the heuristic that there should be *significant* transmission of signal through the network at initialization to allow for efficient learning (3, 4), i.e. that when different inputs are presented to the untrained network, its output should vary substantially. Learning rate schedules were chosen by search to be slow enough to give fairly stable learning, but fast enough to not harm generalization (c.f. 5).

Many of the remaining parameters take the values they do for somewhat arbitrary reasons, e.g. the polynomial experiments were run earlier, before 1-layer task networks were found to be useful in some settings (although the complex tasks and transformations in the polynomial setting may benefit from the more complex task networks). While it would be ideal to fully search the space of parameters for all models, unfortunately our computational resource limitations prohibited it. Thus the results in the paper should be interpreted as a lower bound on what would be possible.

A.3. Model training details. In all experiments, each epoch of training consisted of a single learning step on each task (both base and meta), in a random order. That is, training of the base tasks and meta-mappings was fully interleaved. However, the greater prevalence of base tasks, the learning rate schedules, and the fact that the loss on the meta-mappings is small when the base-task embeddings are small (near initialization) all mean that the base tasks are effectively prioritized earlier in learning.

Examples & generalizing: Where tasks were performed from examples, in each task training step, the meta-learner received only a subset (the "support set size" in Table S3) of the examples to generate a task representation, and would need to generalize to the remaining probe examples in the batch. In fact, the system was trained to execute the mapping on *both* the support set *and* the probe set. This likely did not substantially alter the learning compared to just training the mapping on the probe set, but may perhaps have made it easier for the model to understand the overall structure of the problem early in learning. Where the task or meta-mapping representations were generated from language, there was no need for a separate support set of examples to generate the task representation. Thus, again, the full batch was used to train the mapping.

The representations of the basic tasks for meta-mappings were computed and cached once per epoch, so as the network learned over the course of the epoch, the task representations became "stale," but this did not seem to be too detrimental to learning. In the case of the RL tasks, where there were persistent task representations (see below), they were used instead.

Gradients: In Fig. S1, we show the flow of inference (forward) and gradients (backward) through our architecture on basic task and meta-mapping training steps. All networks used for performing the base tasks were trained by end-to-end optimization on the appropriate base task loss. That is, the task loss gradients update all networks from the output decoder back through the hyper network, example network, and even the encoding of the task examples and task inputs.

During meta-mapping training, the model was trained to match its transformed task representations to target task representations by an ℓ_2 loss. Gradients were stopped at the example and inference task representations, rather than updating how those representations were constructed. This simplification was due to memory constraints; it was not possible to fit the construction of all task representations used as examples within GPU memory. An implementation that allowed for this (at least for some task representations, e.g. the source task) might improve learning, and could allow meta-mappings to improve basic meta-learning generalization directly, by shaping the construction of the basic task representations to follow the relationship structure of the task space.

Multiple runs & robustness: The results reported in the figures in this paper are averages across multiple runs, with different trained and held-out basic tasks (in the polynomial and visual concepts domains), different trained and held-out meta-mappings (again in the polynomial and visual concepts domains), and different network initializations and training orders each epoch (in all domains), to ensure the robustness of the findings.

Classifying task representations: For classification of task representations, we constructed a representation of the meta-classification, either from examples — i.e. (task representation, binary classification) tuples — or language. We constructed these representations using the same example or language network that was used for the basic tasks and meta-mappings. This meta-classification representation then parameterized the task network (via the same hyper network used for the other tasks). Probe task representations were then fed into the task network, and the model was then trained to output appropriate classifications for them through a *separate classification output network* — it was necessary to have a separate classification output network because in most domains there was not an appropriate classification output. The model was trained on these meta-classifications via a cross-entropy loss.

The idea of this training was that it would help the model identify important features of the task representations that would be relevant for the meta-mappings it needed to perform. However, as we show in Fig. S11, meta-classification did not prove substantially beneficial in our domains. This may be due to the limited set of classifications we provided. See section B for the specific classifications that were used in each domain.

Persistent task representations: In the main approach to performing tasks from examples in our paper, the task representations for basic and meta-mappings were constructed anew on each episode. However, in domains where superficially similar tasks have directly contradicting goals, it can be useful to maintain partly persistent task representations that update more slowly across training steps. Associating each task with a more consistent representation makes it easier for the model to learn the idiosyncrasies of the tasks. We used this approach when performing the RL tasks from examples.

Specifically, the model stored a representation of each task that was updated slowly over learning (persistent), and additionally, on each step constructed a new representation from examples (as in other settings). On each training step, a uniformly random $t \in [0, 1]$ was chosen, and the representation used for actually performing the task was the convex combination

$$t \cdot (\text{persistent representation}) + (1 - t) \cdot (\text{representation from examples})$$

The model also tried to constrain the persistent and example-constructed representations to match, by minimizing an ℓ_2 loss between the two representations. This both updated the persistent task representation to be closer to the representation constructed from examples (thus making the persistent representation essentially a slowly moving average of the example representations), and also updated the representation constructed from examples to be closer to the persistent representation (thus encouraging any useful knowledge contained in the persistent representation to be incorporated in how the example network processed examples). In this way, the knowledge from each representation could support the other.

Note that persistent task representations are not required when performing basic tasks or meta-mappings from language-based representations — because the language input is consistent across training steps (unlike the examples), the language-based task representations already change relatively slowly between training steps.

A.4. Model & training modifications for Cards & RL. Because in both the Cards and RL domains the system can only take one action, and only receives feedback on that action, we needed to modify the architecture and training slightly. As noted in the main text, we thus replaced the (input, target) examples used to infer a supervised task with (state, (action, reward)) example tuples. These tuples are the basic currency of model-free RL algorithms. To use these tuples, we provide both the action and reward to the target output encoding network, so that it can process them together and produce a single representation.

The model is trained to output the expected reward of the actions (in the Cards domain), or the Q-value (in the RL domain), via an ℓ_2 loss. Again, the fact that the network only receives rewards for the action it takes means that, for any given step in memory, the model can only be trained to better predict the reward (or Q-value) of the single action that it took.

Additional model & training modifications for RL: There are a number of additional changes that were necessary for the RL tasks, due to the additional complexity of the temporal structure. These changes generally followed the approach of the original DQN (6). The model received pixel-images as input, and produced Q-values as output. Target Q-values were produced by the Bellman equation (that is, the target was the max Q-value of the subsequent state plus any reward received), but following Mnih and colleagues (6), the target next-state Q-values were produced by a second (identical) network with frozen weights, that had its weights copied from the main network every 10000 epochs. This helps stabilize learning (by allowing estimates to converge somewhat before the targets change).

We made two additional changes to improve the stability of learning. First, the model maintained persistent representations (see previous section) for each trained task and meta-mapping. The persistent representations helped the model overcome conflicting signals from switched-color tasks, and thereby accelerated learning. (Note that in the experiments performing the RL tasks from language-based task representations, persistent task representations were not used, since the language is already consistent across training steps, unlike examples.) We also incorporated weight normalization (1) in the task network, which reparameterizes the weights so that their magnitude and direction are estimated separately. Although learning might have converged without these changes, they seemed to stabilize and accelerate convergence.

The memory buffers of the system were refreshed every 1500 epochs by allowing the system to play each (training) task for as many episodes as were necessary to generate the 1000 (state, action, reward) tuples necessary to fill the memory buffer. The examples used in any particular network training step were sampled uniformly at random from this buffer, without regard to continuity or epsiode boundaries, as is standard in DQN training. During play to fill the memory buffers, we used both ϵ-exploration *and* chose actions from a softmax over Q-value.

As in all other experiments, the base tasks and meta-mappings were trained simultaneously, but with different learning rate schedules (see Table S2).

Evaluation for RL: Evaluation was performed by allowing the system to play each task for a total of 10 randomly generated episodes, with the return assessed as the mean return across this set. While ϵ-exploration was turned off during evaluation, the softmax policy was left on. Without the softmax over actions, the model generalized somewhat worse, presumably because its Q-values are not adapting perfectly and it could easily get stuck in a loop of incorrect actions. The softmax allows some possibilty of breaking out of these loops. Some of the recordings linked in the repository exemplify this, e.g. https://github.com/lampinen/homm_grids/blob/master/recordings/run0_pusher_red_blue_True_False_recording_0.gif, where the agent gets stuck in the corner after pushing the first three blocks, before eventually breaking out and converging on the correct final block.

We decided when to evaluate the model on each task by:

1. Requiring the performance on all trained base tasks to be above 95% (to ensure that the model had learned both tasks, since the "push-off" tasks were slower to learn).

2. Selecting the time when the performance on the **other** evaluation task was highest (i.e. using the other task as a validation set).

This means that the performance on each evaluation task may be evaluated at different times during the run. Selection of the stopping point for each task is independent of selecting the stopping point for the other. Note that this optimal stopping approach is not biased, since the task used to decide when to evaluate is always the task that is **not** being evaluated. To see why this is valid, note that we could have run the model twice for each run, once where we held out one task as a validation set, and the other as the test set, and another run where these were switched. Our evaluation approach is essentially equivalent to this, except applying the two independent evaluations within the same run to save running the entire training process twice as many times.

A.5. Optimizing task representations. To optimize the task representations on new tasks, we perform gradient descent on those embeddings through the model architecture. We use the same optimizer as was used in the main experiments (i.e. Adam for the polynomials results, RMSProp for the visual concepts), but with a fixed learning rate of $1 \cdot 10^{-4}$.

For the random vector initialization, we sampled the values IID from a normal distribution with variance $1/\sqrt{512}$ to give approximately a unit-length vector. The centroid initialization was the centroid of all the trained basic-task representations (i.e. meta-mappings were not included), and the arbitrary trained task representation was likewise an arbitrary trained basic task representation. The untrained model comparison was initialized to exactly the initialization states from which our architectures were trained.

B. Task and dataset details and methods. In this section, we describe the details of basic tasks and meta-mappings in each of our domains. See table S3 for a summary of the training and hold-out sizes (at the level of support sets and probes for both basic tasks and meta-mappings) for each domain. In the remainder of the section, we describe details of how the tasks were sampled, how they were encoded into language (if applicable), etc.

	Polynomials	Cards	Visual	RL
Base input type	$\mathbb{R}^4$	Several-hot vector $\in \{0,1\}^{12}$	50×50 RGB image	91×91 RGB image
Base output type	$\mathbb{R}$	Bet values ($\mathbb{R}^3$)	Label $\in \{0,1\}$	Action Q-values $\in \mathbb{R}^4$
Num. base tasks (training)	2260 ($= 60 + 60 \times 36 + 40$)	36	Varies (∼100-300)	18
Num. base tasks (held out for meta-mapping evaluation)	1440 ($= 40 \times 36$)	4	Varies	2
Num. meta classifications	6	8	8	-
Num. train meta-mappings	20	3	Varies (4-32)	1
Num. held-out meta-mappings	16	0	2	0
Base batch size	1024	1024	336	64
Base support set size	50	768	-	32
Meta batch size (train)	60	36	Varies	18
Meta support set size (train)	Half of train dataset		-	Half of train dataset
Meta support set size (eval)	All of train dataset		-	All of train dataset

Table S3. Dataset compositions and specifications for the different experiments. A "-" indicates a parameter that does not apply to that experiment. Batch sizes refer to the total number of data points used per training step (or the number of (s, a, r) tuples for the RL tasks), including both those used as support set examples provided to the example network, and those used as probe examples for generalization. Support set sizes refer to the number of examples presented to the example network in order to construct a task representation. The difference between the batch size and the support set size provides the number used as probes. Note that for the language-based meta-mapping (performed in the visual concepts domain, as well as in later experiments in the RL domain) all the meta-batch is used as probes, since no support set is needed.

B.1. Polynomials. We randomly sampled 100 train polynomials as follows:

1. Sample the number of relevant variables (k) uniformly at random from 0 (i.e. a constant) to the total number of variables.
2. Sample the subset of k variables that are relevant from all the variables.
3. For each term combining the relevant variables (including the intercept), include the term with probability 0.5. If so give it a random coefficient drawn from $\mathcal{N}(0, 2.5)$.

We then split this set of 100 polynomials into 60 that were used to train the meta-mappings, and 40 for which the targets would be held-out to evaluate each meta-mapping. We thus needed to also train the system on the transformed targets for each meta-mapping applied to the 60 polynomials, so the total number of trained polynomials was $60 + 60 \times 36 + 40 = 2260$. The total number held-out for evaluation was 40 per meta-mapping, i.e. $40 \times 36 = 1440$.

Note that the above means that we trained the system on the transformed polynomials that were in the support set of *even the held-out meta-mappings*. That is, a held-out meta-mapping is held-out in the sense that the meta-mapping itself is not trained, but the supporting polynomials are still in the train set. Of course, in principle the model would be able to perform a meta-mapping supported by polynomials it had never encountered before (using task representations constructed from examples of those polynomials). However, our approach allows more careful evaluation of the meta-mapping generalization of the model, by making the supporting polynomial representations more reliable. This eliminates a confound when comparing held-out meta-mapping generalization to trained meta-mappings, by ensuring base knowledge is matched.

The data points on which these polynomials were evaluated were sampled uniformly from $[-1, 1]$ independently for each variable, and an independent set was sampled for each polynomial. Note that although input domain is restricted, the output range can be quite large under this distribution (often around $[-40, 40]$), because of the wide distribution of coefficients and the summing of multiple terms. The datasets were resampled every 50 epochs of training.

Meta-mappings: We trained on 20 meta-mapping tasks, and held out 16 related meta-mappings.

- Squaring polynomials (where applicable, i.e. where degree was ≤ 1, so that the squared polynomial wouldn't have degree > 2).
- Adding a constant (trained constants: -3, -1, 1, 3, held-out: 2, -2).
- Multiplying by a constant (trained constants: -3, -1, 3, held-out: 2, -2).
- Permuting inputs (trained on 12 permutations, held-out 12, randomly chosen on each run).

Meta-classifications: We also trained the network on 6 task-embedding classification tasks:

- Classifying polynomials as constant/non-constant.
- Classifying polynomials as zero/non-zero intercept.
- For each variable, identifying whether that variable was relevant to the polynomial.

B.2. Card games. Our card games were played with two suits (red and black), and 4 values per suit. In our setup, each hand in a game has a win probability (proportional to how it ranks against all other possible hands). The agent is dealt a hand, and then has to choose to bet 0, 1, or 2 (the three actions it has available). We considered a variety of games which depend on different features of the hand:

- **Straight flush:** Most valuable is adjacent numbers in same suit, i.e. 4 and 3 in most valuable suit (royal flush) wins against every other hand. This is the game on which we tested adaptation in the models and human participants.
- **High card:** Highest card wins.
- **Pairs** Same as high card, except pairs are more valuable, and same suit pairs are even more valuable.
- **Match:** The hand with cards that differ least in value (suit counts as 0.5 pt difference) wins.
- **Blackjack:** The hand's value increases with the sum of the cards until it crosses 5, at which point the player "goes bust," and the value becomes negative.

We also considered three binary attributes that could be altered to produce variants of these games:

- **Losers:** Try to lose instead of winning! Reverses the ranking of hands. This is the mapping we evaluated in the models and human participants.
- **Suits rule:** Instead of suits being less important than values, they are more important (essentially flipping the role of suit and value in most games).
- **Switch suit:** Switches which of the suits is more valuable.

Any combination of these options can be applied to any of the 5 games, yielding 40 possible games. We held out all losing variations of the Straight Flush game for evaluation.

Meta-mappings: We trained the network on meta-mappings that toggled each of the binary attributes, but evaluated primarily on switching to losing the Straight Flush game (since that corresponded to the human experiment).

Meta-classifications: For meta-tasks, we gave the network 8 task-embedding classification tasks (one-vs-all classification of each of the 5 game types, and of each of the 3 attributes)

Language: We encoded the tasks in language by sequences of the form
`[‘‘game’’, <game_type>, ‘‘losers’’, <losers-value>, ‘‘suits rule’’, <suits-rule-value>, ‘‘switch suit’’, <switch-suit-value>]`.

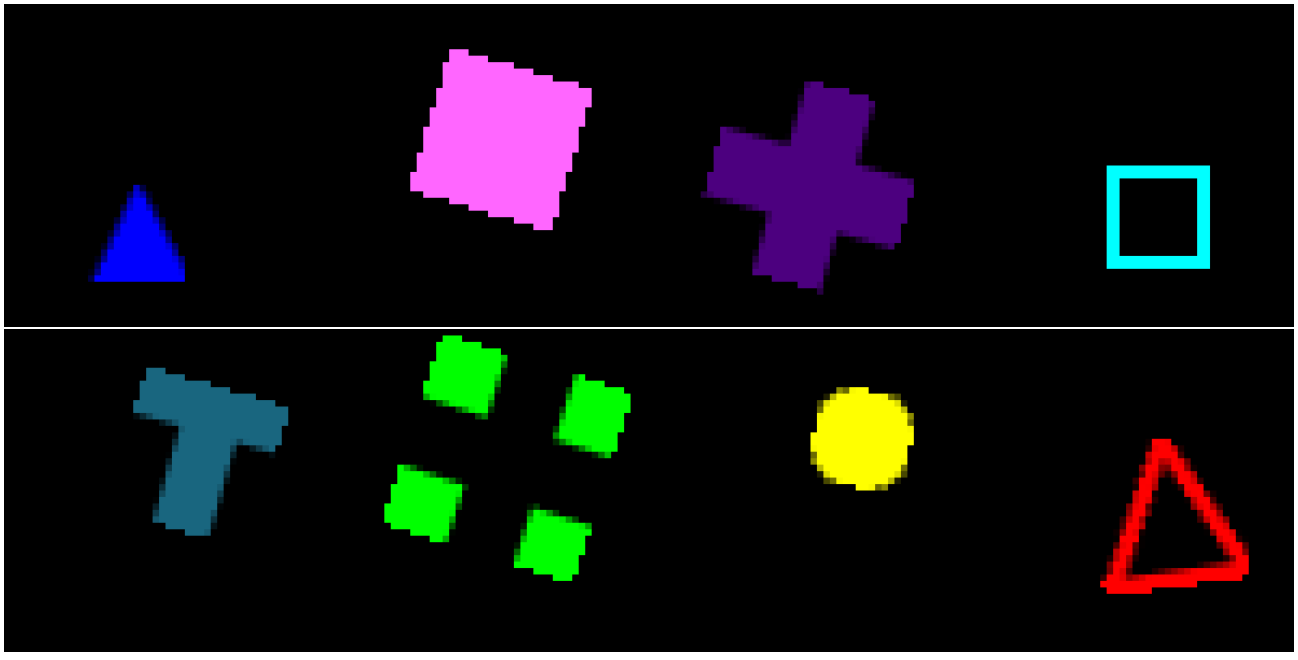


Fig. S2. Sample stimuli for visual concept tasks, showing all shapes, colors, and sizes.

B.3. Visual concepts. In Fig. S2 we show all shapes (triangle, square, plus, circle, tee, inverseplus, emptysquare, emptytriangle), colors (blue, pink, purple, yellow, ocean, green, cyan, red), and sizes (16, 24, and 32 pixels) that we used in our experiments. All stimuli were rendered at random positions within a 50×50 image (constrained so that the full shape remained within the frame), and at random angles within $\pm 20^\circ$ of their canonical orientation.

Sampling of meta-mappings: We sampled an equal number of meta-mappings that switched colors and meta-mappings that switched shape. We held-out one meta-mapping of each type. Within each type, the particular meta-mappings used for training and evaluation on a given run were sampled uniformly at random.

Sampling of basic concepts: We trained the system on all uni-dimensional concepts as training examples (i.e. one-vs.-all classification of each shape, color, and size), so that it could learn all the basic attributes. We included 6 training example pairs of each mapping (one for each combination of rule type and other attribute). We also included 6 other pairs for evaluation, where the source concept was trained, but the target was held-out for evaluation. Note that our selection criteria mean that each held-out example will have a closely matched trained one. That is, the number of basic concepts the system encounters during training is roughly 18 trained per meta-mapping (roughly because it can be reduced if the meta-mappings have overlapping examples), and the number of evaluation concepts is roughly 6 per meta-mapping. For example, the system might be trained on mappings like "switch-red-to-blue," with corresponding examples like AND(red, triangle) $\mapsto$ AND(blue, triangle). It would then be evaluated on closely matched examples like AND(red, circle) $\mapsto$ AND(blue, circle), where the latter is untrained.

In addition to these sampled pairs, we trained the meta-mapping on any other pairs of concepts which were valid examples of the mapping and happened to be sampled as part of support for other meta-mappings. For example, if AND(red, square) was a train target task for some other mapping, and AND(blue, square) was a trained source task for another, the pair AND(red, square) $\mapsto$ AND(blue, square) would be used to train the "switch-red-to-blue" meta-mapping.

For a held-out meta-mapping, e.g. "switch-green-to-blue," the same basic concepts instantiating the meta-mapping were trained as would be for a trained mapping, but the meta-mapping itself was not. As in the polynomials domain, matching the training of the supporting basic tasks between trained and held-out meta-mappings makes the comparison between them more precise.

Meta-classifications: In addition to the meta-mappings mentioned in the main text, we trained the system on 9 meta-classifications: classifying whether the task was a basic-level rule on any of the three basic dimensions, classifying whether each dimension was relevant (regardless of whether the task was basic or composite), and classifying the type of composite (if the task was not basic).

Language: We encoded the tasks in language by sequences from the following grammar:

- **Basic rules:** encoded as `[<attribute-name>, ''='', <attribute-value>]`, for example `[''shape'', ''='', ''triangle'']`
- **Composite rules:** encoded as `[<composite-type>, ''('', ''('', <basic-rule>, '')'', ''&'', ''('', <basic-rule>, '')'', '')'']`, where the `<composite-type>` is one of "AND", "OR", or "XOR", and each `<basic-rule>` is substituted with a sequence as above.
- **Meta-mappings:** encoded as `[''switch'', <attribute-name>, <old-attribute-value>, ''~'', <new-attribute-value>]`.
- **Meta-classifications:** encoded as `[''is'' <composite-type>]` or `[''is'', ''basic'', ''rule'', <attribute-name>]` or `[''is'', ''relevant'', <attribute-name>]`, depending on the type of classification.

B.4. RL. The RL tasks were implemented using the open-source Pycolab library (https://github.com/deepmind/pycolab). The tasks were implemented in a 6×6 room, surrounded by an impassable varrier. The agent could navigate using four actions, corresponding to moving in the four cardinal directions. If it attemtped an invalid action, the state did not change.

Each episode ended after either 150 timesteps elapsed (that is, after the agent took 150 actions, including invalid actions), or after the agent had picked up 4 of the 8 objects (regardless of whether they were good or bad) in the pick-up task, or pushed off 4 of the 8 in the push-off task The agent received a reward of $+1$ for picking up or pushing off the good-colored objects, and -1 for the bad-colored objects. Selected recordings of the agent playing the games after meta-mapping can be found at https://github.com/lampinen/homm_grids/tree/master/recordings, which may help clarify any unclear aspects of the tasks.

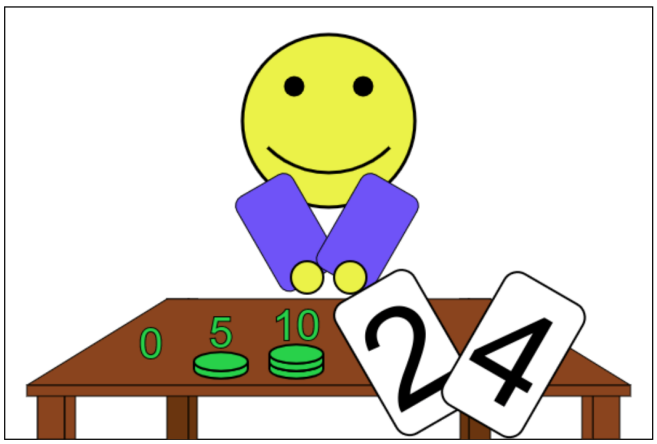
Meta-classifications: We did not train any meta-classifications in this setting.

Language: We encoded the tasks in language by sequences of the following form:

- **Basic task:** encoded as `[<game-type>, <color1>, <color2>, <good-color-position>]`, where `<game-type>` was either "pusher" or "pickup", colors were names of a color pair, and `<good-color-position>` was "first" or "second" depending on whether the first color was good, or the second color (after switching).
- **Meta-mapping:** encoded as `[''switch'', ''colors'']`.

C. Cards behavioral experiment. Here we provide the details of the human experiment for the cards tasks. The human experiment was conducted on Amazon Mechanical Turk. Our study protocol was approved by the Stanford University Institutional Review Board panel on Human Subjects in Non-Medical Research. At the beginning of the experiment, subjects were given overview of the general topic of the experiment (game playing) and the compensation scheme, and then were informed that they could opt out of the experiment at any time if they did not wish to participate further.

We tried to design the game that participants played to make it easy for them to learn, without relying on their prior knowledge of card games. The game was a simplified variation of poker, which we denoted "Straight Flush" in the card game descriptions above. The participants were dealt hands which consisted of two cards, each with a number (rank) between 1 and 4, and a color (suit) of red or black. The participants played against a computer opponent that was dealt a similar hand. The hands were ranked such that straight flushes (adjacent cards in the same suit) beat adjacent cards in a different suit, which beat non-adjacent cards (including pairs). Ties were broken by the highest card, or by suit if both cards were tied.



(a) Before betting.

(b) Feedback.

Fig. S3. The card game experiment trials, as seen by participants. (a) The beginning of the trial, in which participants can see their hand, and choose an amount to bet by clicking on it. (b) The feedback phase, where participants saw their opponents hand and the result. In the evaluation trials, where participants did not receive feedback, this phase was replaced with a semi-transparent gray overlay before the next trial.

On each trial, participants were dealt a hand and asked to make a bet of 0, 5, or 10 cents (see Fig. S3). If their hand beat the opponent's hand, they won the bet amount. If their hand lost, they lost it. If the hands were tied, they neither won nor lost money.

The experiment had several phases. First, participants were instructed in the rules and payment scheme for the experiment. Next, they were instructed on the rules of the game. After this, they were tested with four hand-comparison trials intended to probe their understanding of each of the rules of the game. If they failed more than one of these trials, they were not allowed to continue with the experiment.

Following this understanding check, participants played a block of 32 hands (sampled to have a diversity of expected values), where they saw the results of their play (as in Fig. S3b). After this block, they played a similar block of 24 trials where they did not see the results of their play. The results were replaced with a brief grayed-out screen, and participants were payed the net expected value of their actions over the block (rounded to the nearest 10). The evaluation phase without feedback provides an evaluation with relatively less potential for learning, in order to get a precise estimate of their performance.

Finally, participants were told that we wanted them to try to lose for the remaining trials, and that "for the remainder of the experiment, if you bet and lose, you'll gain the amount you bet, and if you bet and win, you'll lose the amount you bet." They were then given an attention check to evaluate whether they had understood this instruction. Subjects who failed this attention check were excluded from the analysis. They then played another block of 24 trials where they were rewarded for losing instead of winning (i.e. the relationship between actions and expected returns was reversed relative to the first phase of the experiment). As in the previous block, they did not see the results of their actions, they were only shown their total earnings at the end of the block. By not providing feedback on each trial, we were able to get many trials of "zero-shot" data, to more carefully evaluate their performance. They were finally asked a few demographic questions.

Our target comparison was performance in the two blocks without feedback – were participants able to switch their behavior to lose at the game as well as they won at it? Rather than evaluating on stochastic rewards based on sampled opponent hands, we evaluated them by the expected value of their performance across the hands they played. This is exactly analogous to the experiment performed for the model (except that the performance of the model was evaluated on all possible hands in each condition, which was infeasible for the human participants).

Participants were paid $1 for starting the experiment and completing the instruction section. If they failed the first understanding check, the experiment ended. Otherwise, they were paid an additional $1.50 to complete the performance phase, and then were bonused based on their winnings to incentivize performance. We recruited 40 participants for the experiment, but only 19 successfully passed the first understanding trials. Of those 19, only 17 passed the try-to-lose attention check, so our analyses were restricted to 17 subjects.

Further details of the experiment, including the text of all instructions, can be found in the first author's dissertation (7, pp. 112-117, accessible at https://stacks.stanford.edu/file/druid:xj689nb3522/dissertation-augmented.pdf).

D. Source repositories. The full code for the experiments and analyses can be found on github:

- Meta-mapping library: https://github.com/lampinen/HoMM
- Polynomials: https://github.com/lampinen/HoMM_polynomial_analysis
- Cards (models): https://github.com/lampinen/HoMM_cards
- Cards (human experiment): https://github.com/lampinen/cards_for_humans

- Concepts: https://github.com/lampinen/categorization_HoMM
- RL: https://github.com/lampinen/HoMM_grids
- Stroop results (below): https://github.com/lampinen/stroop

E. Other acknowledgements. The color palettes used in the figures are adapted from ColorBrewer (8). The playing card images used in the main text are based on the images at https://commons.wikimedia.org/wiki/Category:Playing_cards_set_by_Byron_Knoll on WikiCommons, which the creator kindly released for use.

F. Supplemental analyses & figures. The analyses are organized as follows. In F.1, we show additional analyses in the polynomial domain, including evaluation of sample efficiency and several architectural lesions. In F.2, we analyze the representations of the models in the polynomial domain, showing that they are systematically organized across runs, and presenting further details on the representation transformation results presented in the main text. We also show evidence that the meta-mapping and basic task representations are sharing representational subspaces, and show significant overlap with isomorphisms we know exist between them in the polynomials domain. In F.3, we show further analyses of the results of the card game experiments, both from the behavioral and modeling perspective. In F.4 we show more detailed results in the visual concepts domain. In F.5 we show further analyses of the RL experiments. In F.6 we provide details and analyses for the comparisons to generalizing from language alone, and in F.7 we provide additional experiments on generalizing from switching colors to switching shapes in the RL context. In F.8, we provide further analyses of meta-mapping as a starting point for later learning. In F.9, we demonstrate our model on a simple Stroop-like task, common in cognitive control.

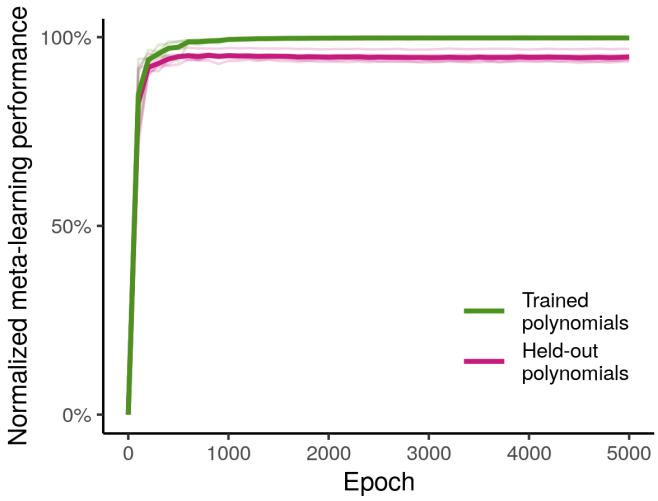


Fig. S4. Basic task (meta-learning) performance in the polynomials domain over learning. The system is generalizing at the meta-learning level. That is, this graph shows that, after the example network receives a set of (input, output) example tuples, it is generating a sufficiently good representation to regress held-out points from that polynomial. This is true both for polynomials it was trained with (green), and for polynomials that are held-out and never encountered during training (pink). Performance is plotted normalized as 100% $\times(1 - \text{loss}/c)$, where c is the loss for a system outputting all zeros, as in the meta-mapping results. In this case, this measure corresponds exactly to the percentage of variance explained. (Thick dark curves are averages over 5 runs, shown as light curves.)

Evaluation types		MSE loss			Normalized performance	
Meta-mapping trained?	Polynomial is trained example?	Meta-mapping	Zeros	No adaptation	Meta-mapping	No adaptation
Trained	Support (trained)	0.317	18.8	18.1	98.3%	4.18%
Trained	Probe (held-out)	1.85	16.7	16	89%	4.26%
New	Support (trained)	0.97	12.4	9.89	92.1%	20.6%
New	Probe (held-out)	1.56	10.8	8.71	85.5%	19.3%

Table S4. The raw mean-squared-error (MSE) losses and the normalized performance measures after meta-mapping in the polynomials domain. The first column indicates whether the meta-mapping is trained or held-out, and the second indicates whether the polynomial is provided as an example of the mapping (and is therefore used for training the mapping, if the mapping is trained) or if the polynomial is held out for evaluation. The meta-mapping columns provide the MSE/performance for the model after meta-mapping, the zeros column provides the loss for a model that outputs all zeros, and the "No adaptation" columns provide the MSE/performance for a model using the unadapted source task representations. The normalized performance measure is calculated as 100% $\times(1 -$ Model MSE$/$Zeros MSE$)$.

F.1. Polynomials. **Basic meta-learning:** In Fig. S4, we show that the basic meta-learning is working well in the polynomials domain. That is, we show that after the example network is presented with a set of example input, output pairs from a

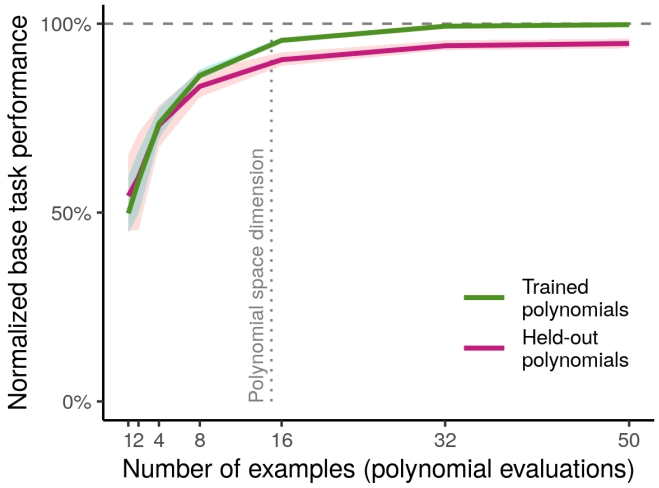


Fig. S5. The effect of number of examples on basic task performance in the polynomials domain. The system is relatively sample efficient. Its performance is quite high by the time it has received the minimum number of samples that an optimal least-squares solver with knowledge of the ground-truth task space would need (that is, the dimensionality of the polynomial vectors space, indicated by the vertical dotted line), although the performance continues to improve slowly beyond that point. (Averages across 4 runs, with bootstrap 95%-CIs across runs.)

polynomial, the system is generalizing well to other points from that polynomial. At the end of training, the mean performance on trained polynomials is 99.78% (bootstrap 95%-CI [99.74, 99.84]), and for held-out polynomials it is 94.8 (bootstrap 95%-CI [93.8, 95.9]).

Relationship of raw meta-mapping performance to normalized performance: In Table S4, we show the relationship between the mean-squared-error (MSE) losses of the model on meta-mapped tasks, and the normalized performance measure we report in the main text. Note that this table includes results for training examples, while the main text only reports the evaluation results. For context, the MSE of the model when performing a trained polynomial from examples is 0.025, and the performance is 99.8% (see above), so the model is not performing quite as well after meta-mapping even a trained example polynomial as it performs a trained polynomial from examples. This is not particularly surprising, since there are more sources of noise in meta-mapping a task — first, the representation of the source task, next the representation of the meta-mapping, and finally the transformation itself.

Sample efficiency (base tasks): In Fig. S5, we explore the sample efficiency of the basic meta-learning system by evaluating how the performance of the system changes depending on the number of examples it is given. Note that because the models were trained with 50 examples per polynomial, performance at smaller sizes would likely improve somewhat beyond these results if it were trained initially with smaller numbers of examples.

Meta-mapping results by mapping type: In Fig. S6 we show the meta-mapping results in the polynomials domain, broken down by the type of mapping. The system performs well across all mapping types.

Sample efficiency (meta-mappings): In Fig. S7, we show how the meta-mapping performance depends on the number of examples — that is, (input task, output task) tuples — that the system is given. Performance is unsurprisingly quite low with 1 example, but increases rapidly with a few examples. In Fig. S8 we show performance by number of examples for each meta-mapping type. The square meta-mapping in particular is difficult, and performance is actually negative with only a few examples of it, unlike the other mappings. However, once the system receives enough examples, it is able to recognize the square mapping and perform well at it.

Nonhomiconic architectures: We next consider some architecture lesions. In Fig. S9, we compare our homoiconic architecture to a nonhomoiconic architecture – i.e. one in which there are separate example networks ($\mathcal{E}_{base}, \mathcal{E}_{meta}$) and hyper networks ($\mathcal{H}_{base}, \mathcal{H}_{meta}$) for the base tasks and meta-mappings. The nonhomoiconic approach performs substantially worse. Specifically, on trained meta-mappings the homoiconic model is achieving a normalized performance of 88.99% (bootstrap 95%-CI [88.20, 89.98]), while the non-homoiconic achieving a normalized performance of 83.2% (bootstrap 95%-CI [81.9, 84.9]). On new meta-mappings the homoiconic model is achieving a normalized performance of 85.54% (bootstrap 95%-CI [85.14, 85.94]), while the non-homoiconic model is achieving a normalized performance of 81.3% (bootstrap 95%-CI [80.3, 82.2]). (See also Sec. F.2, in which we show that there is intriguing overlap between the representations of meta-mappings and base tasks in a homoiconic architecture.)

A simpler task architecture: In Fig. S10a we show that a simpler task network, which just takes a task representation as another input to feed-forward processing, performs perhaps slightly worse than the HyperNetwork-based approach. Specifically, in the simpler architecture, there is a fixed feed-forward task network, and rather than using the task representation to alter the weights of this network, the task-representation is simply concatenated to the input representation and then propagated through the fixed network. Note that the task-concatenated architecture does not perform worse at meta-learning (normalized performance on evaluation tasks 95.7%, bootstrap 95%-CI [95.0, 96.6] vs. 94.8% [93.8, 95.9]), it is adapting via meta-mappings

that proves challenging for it.

Meta-classification task lesion: In Fig. S11a we show that the meta-classification training is not beneficial in the polynomials domain. Specifically, on trained meta-mappings the model is achieving a normalized performance of 88.99% (bootstrap 95%-CI [88.20, 89.98]), while without meta-classification it is achieving a normalized performance of 89.7% (bootstrap 95%-CI [88.87, 90.61]). On new meta-mappings the model is achieving a normalized performance of 85.54% (bootstrap 95%-CI [85.14, 85.94]), while without meta-classification it is achieving a normalized performance of 86.29% (bootstrap 95%-CI [85.54, 86.79]). However, the effect is small, and in Fig. S11b we show that meta-classification may be helpful in the cards domain, where there are fewer training tasks.

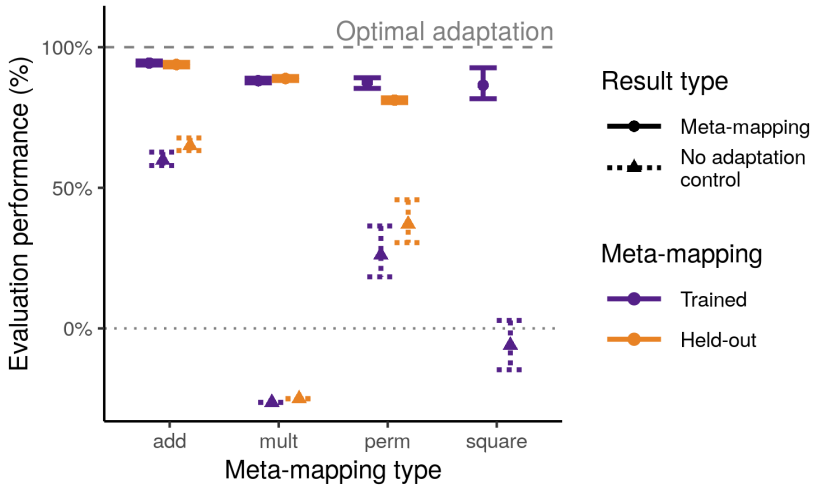


Fig. S6. Meta-mapping performance in the polynomials domain, broken down by meta-mapping type. We plot a normalized performance measure, as in the main text. The system is performing well across all meta-mapping types, although there is some variability. Triangles show performance of a baseline model that does not adapt — note that some meta-mappings are relatively easier for such a model, while in other cases such a model results in worse performance than outputting all zeros.

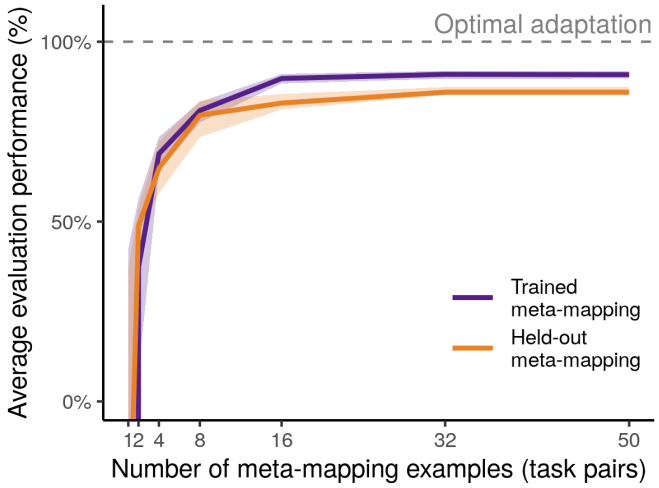


Fig. S7. The effect of number of examples on meta-mapping performance (for add, multiply, and permute) in the polynomials domain. The system is relatively sample efficient. Although the system was trained with 30 examples of each meta-mapping, performance is relatively stable above 16 examples. (Averages across 4 runs, with bootstrap 95%-CIs across runs. The square meta-mapping is omitted from the data in this plot because of its unique trajectory, see Fig. S8.)

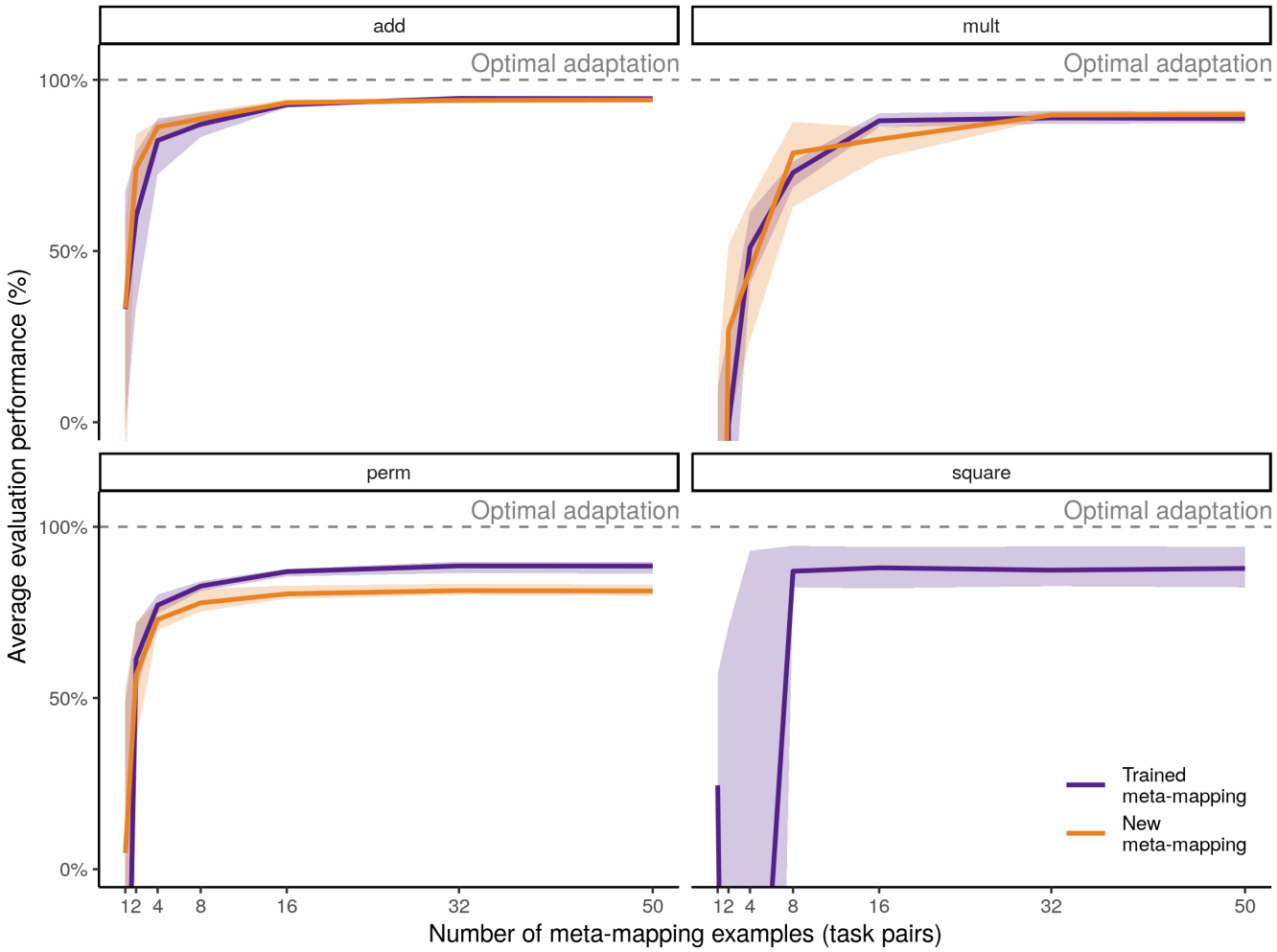
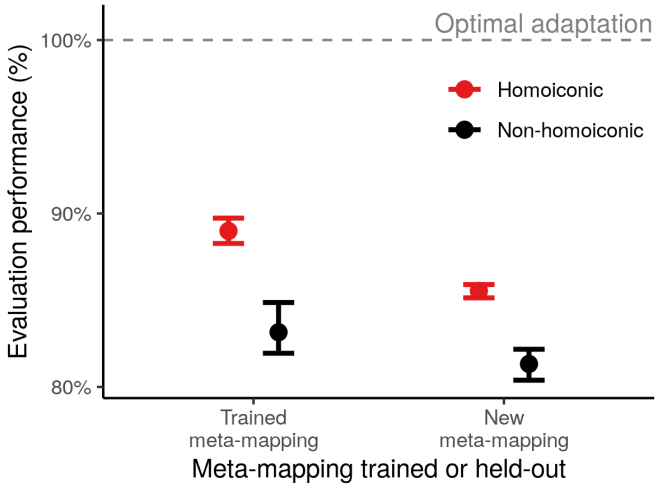
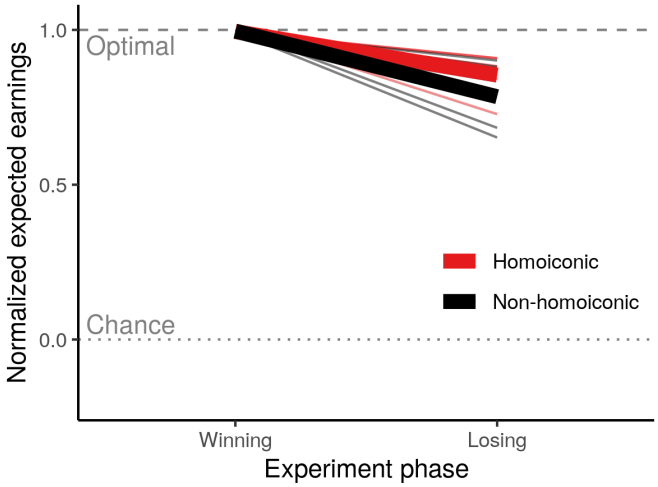


Fig. S8. The effect of number of examples on meta-mapping performance in the polynomials domain, broken down by meta-mapping type. The sample efficiency of the system depends on the meta-mapping. In particular, the square meta-mapping is difficult to estimate from few examples, and performance on that mapping is quite low with small numbers of examples. (Averages across 4 runs, with bootstrap 95%-CIs across runs.)

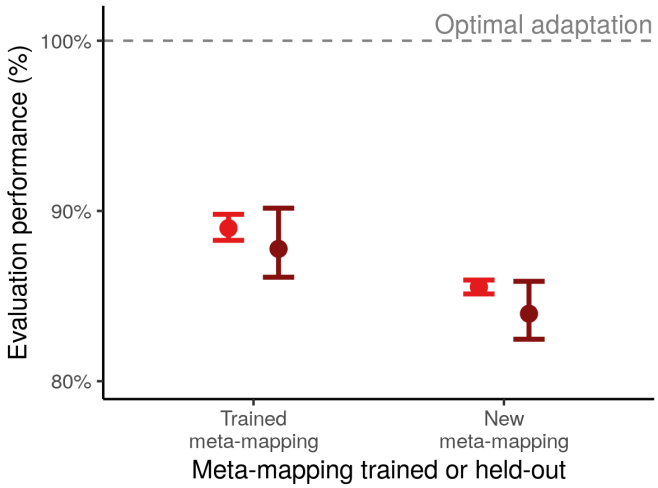


(a) The polynomial domain, compare to Fig. 3.

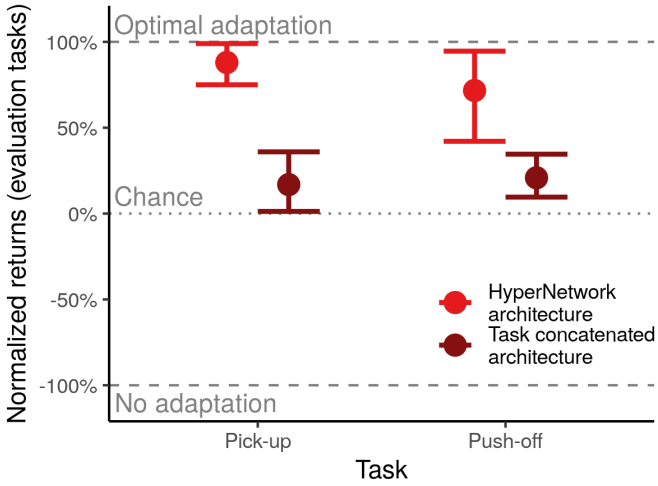


(b) The cards domain, compare to Fig. 5.

Fig. S9. Our homoiconic model outperforms or equals a non-homoiconic baseline in the polynomials and cards domains. This figure compares the meta-mapping performance of our architecture with that of a nonhomoiconic model that instantiates separate copies of the example network ($\mathcal{E}_{base}, \mathcal{E}_{meta}$) and hyper network ($\mathcal{H}_{base}, \mathcal{H}_{meta}$) for the basic tasks and the meta-mappings. In the polynomials domain (a), the homoiconic architecture significantly outperforms the nonhomoiconic one, while in the cards domain (b), the difference is not significant. These results suggest that there is sufficient shared structure between the basic tasks and the meta-mappings for the homoiconic approach to improve generalization, at least in the polynomials case, and supports our use of homoiconic architectures.

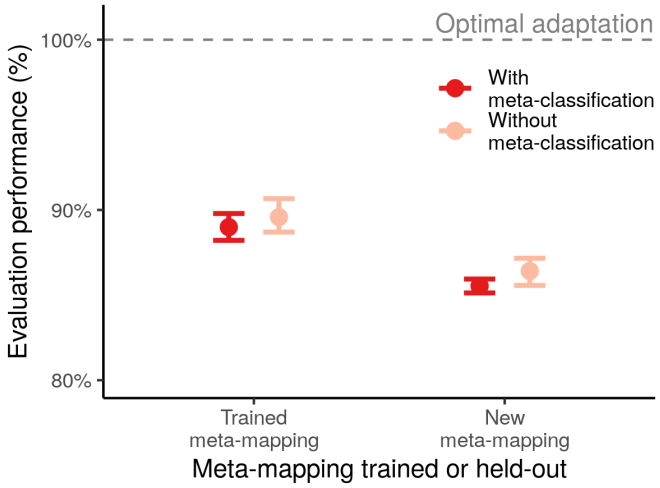


(a) The polynomial domain, compare to Fig. 3.

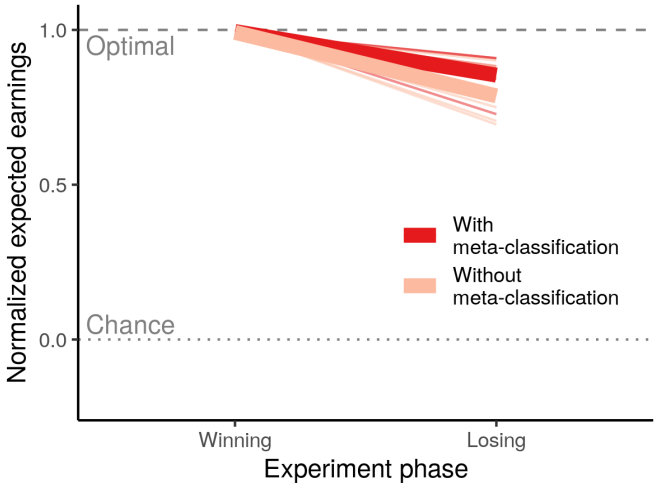


(b) The RL domain, compare to Fig. 9.

Fig. S10. The HyperNetwork-based architecture we propose in the main text performs as well or better on meta-mappings than an architecture that simply concatenates a task representation to the input before passing it through a fixed MLP, at least on the subset of our domains on which we ran a comparison. (See Fig. S29 for a similar comparison for the language generalization baseline.)



(a) The polynomial domain, compare to Fig. 3.



(b) The cards domain, compare to Fig. 5.

Fig. S11. The meta-classifications we trained the model with do not appear to be substantially beneficial — a model trained without them performs slightly better in the polynomials domain, while the model trained with them performs marginally better in the cards domain. This difference may be due to the fact that the model is trained on many more basic tasks in the polynomials domain, perhaps obviating the need for meta-classification to shape the representations.

F.2. Polynomial representations. In order to understand the model better, we analyzed its task representations.

However, we note that there are challenges to interpreting representation analyses, particularly in an architecture like ours. Some recent work (9) shows two key challenges of representation analysis. Although that work explored different types of analyses in simpler models, the findings may also apply to this work. First, the representations of a model that performs multiple tasks may be biased towards the simpler tasks, because of the learning dynamics. This may relate to some phenomena we observe below, such as the first principal components of the task representations being driven in large part by the polynomial constant terms, since constant polynomials are the simplest tasks. Second, when the task is non-linear, linear representation analyses can be misleading. The task representations in our model are related to the behavior in a highly non-linear way. Thus, it is not necessary for the representations to be linearly organized for the model to generalize well — indeed, we show below that the model representations may be organized in a more polar structure. Furthermore, our model's mapping of task representations to tasks may be many-to-one; just as we can write either $(x+1)^2$ or x^2+2x+1 to denote the same function, the model may be able to represent the same task with multiple distinct representations. These issues make finding certainty in the meaning of the model's representations difficult.

Nevertheless, the representations do show interesting structure that gives some intuitions for how the model may be performing the tasks. This structure is also relatively consistent across runs, suggesting that the underlying dynamics driving the emergence of these representations are fundamental to the interaction of the task space and the architecture — this merits future investigation. We first examine how the representations of the polynomials are organized, then provide some further details on how they transform under additional meta-mappings, and finally show some relationships between the representations of meta-mappings and basic polynomials.

PCA: First, we performed principal components analysis on the task and meta-mapping representations in the model after training (Fig. S12). This analysis reveals strikingly similar organization of the representation space across different training runs, with constant polynomials pushed to the outside in a semi-circle, and more complex polynomials stretching toward the center, where meta-mappings and meta-classifications are located. This may be due to the learning dynamics — the distance of the task representations from the center appears to be roughly inversely proportional to the complexity of the task, which might imply that the constant polynomials have the largest-magnitude representations because they are easiest to learn, and so their representations receive more consistent updates starting from earlier in the learning process.

To analyze this further, in Fig. S13 we plot the representations for only the constant polynomials, colored by their value (square-root compressed for clarity). This shows that the representations of the constant polynomials are consistently arrayed angularly from lowest to highest value.

Finally, we examined the meta-mapping representations more closely (Fig. S14). This analysis shows that the mappings have a consistent organization across runs, with permutations and addition grouping tightly, but multiplication and squaring, which more drastically alter the polynomials, more dispersed. In particular, multiplying by negative numbers and squaring, which can change polynomials signs and therefore cause a more drastic adaptation, are more separated from the remaining meta-mappings. It is also interesting to note that the addition meta-mappings appear to be organized more by absolute value than sign in at least some runs. There is some interesting structure in higher principal components as well, for example the addition mappings appear to be organized linearly by absolute value in principal components 3 and 4. The organization of the permutation mappings is more chaotic — while mappings that have similar representations appear more likely to differ by only a transposition, because the relationships among the permutations have a much higher-dimensional group structure, they do

not project cleanly into two-dimensional plots.

How meta-mapping transforms the representations: Next, we analyzed how meta-mapping transforms the task representations (Fig. 4). We conducted these analyses (and some of the subsequent ones on homoiconicity and representations) at the suggestion of a reviewer; because of this, there were conducted on a new set of runs, as we had not retained the model parameters for the prior runs. Here, we show some more detailed results. First, in Fig. S15, we show higher-resolution versions of the inset figures from Fig. 4, showing the alignment between the meta-mapping outputs and the nominal targets. Second, in Fig. S16 we show the transformations induced by two additional meta-mappings, adding 3 and an input permutation.

Homoiconicity and overlap between different representations of different data types in the shared space: We then explored how homoiconicity contributes to the success of the model, by analyzing the relationship between the representations of basic tasks and meta-mappings. This is motivated by an observation by a reviewer that one possible explanation for our observation (above) that homoiconic architectures yield better performance is that the result is purely due to regularization, and that the basic tasks and meta-mapping representations reside in orthogonal subspaces of the representation space. While it is difficult to completely rule out the possibility that regularization is playing a role, in this section we show at least that there is more overlap between the meta-mapping and base-task subspaces than would be expected by chance, and that at least some sensible isomorphisms between the basic tasks and meta-mappings may be shaping the representations.

First, in Fig. S17, we explore the cosine similarity between base-task and meta-task representations. We observe non-trivial overlap, which we explore in greater detail in Fig. S18, showing that there is strong and sparse alignment between the top principal components of the polynomials and the meta-mappings, and Fig. S19, showing that the variance of the meta-mapping representations is mostly contained within lower (more important) principal components of the base task representations. Finally, in Fig. S20, we show intriguinging patterns of alignment of the multiplication meta-mappings and constant polynomials depending on whether the signs match, which suggests that the model may be at least partly uncovering the isomorphic numerical structure between these different levels of abstraction. Exploring the alignment between base tasks and meta-mappings further will be an interesting direction for future work.

We also explored the relationship between the representations of basic data inputs to the model (that is, (w, x, y, z) tuples at which to evaluate a polynomial), and the representations of tasks and meta-mappings. The magnitude of the similarities was overall quite small (see Fig. S21), suggesting that, unlike in the case of meta-mappings and base tasks, the model is not substantially exploiting relationships between tasks and data points. This result is not particularly surprising for several reasons. First, there is more structure in common between basic tasks and meta-mappings than between either category and data points, because both basic tasks and meta-mappings are functions. Second, there are more constraints that encourage basic task and meta-mapping representations to be similar in the homoiconic architecture — both are output by the same example network, and both are processed by the same hyper network. By contrast, data points and basic tasks only have a one-sided constraint, viz. that they are both processed by the same example network.

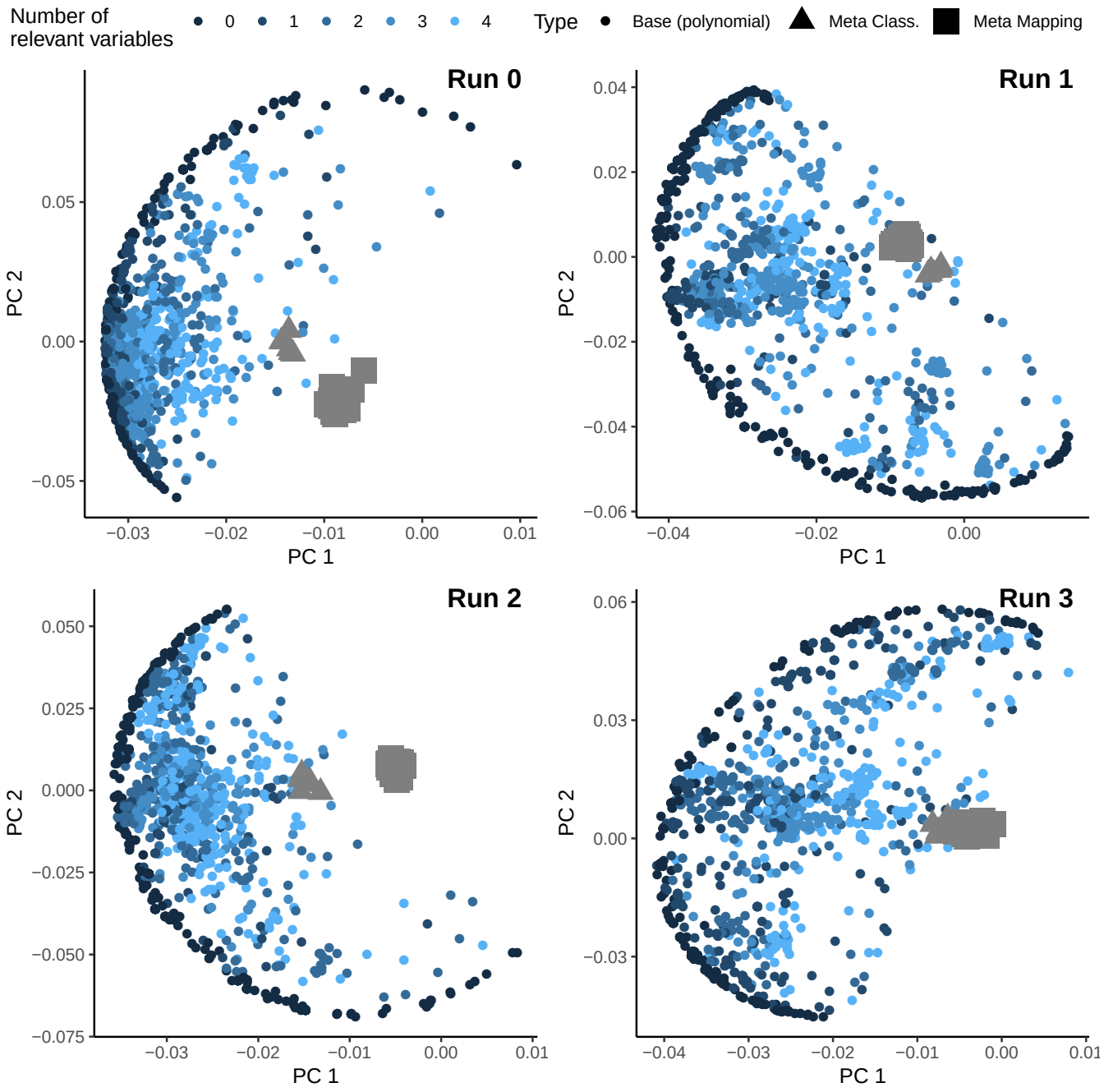


Fig. S12. Principal components of task and meta-mapping representations of our model after training on the polynomials domain. The representation space is organized relatively consistently across runs, with constant polynomials pushed to the outside, and meta-mappings and meta-classifications more centrally located.

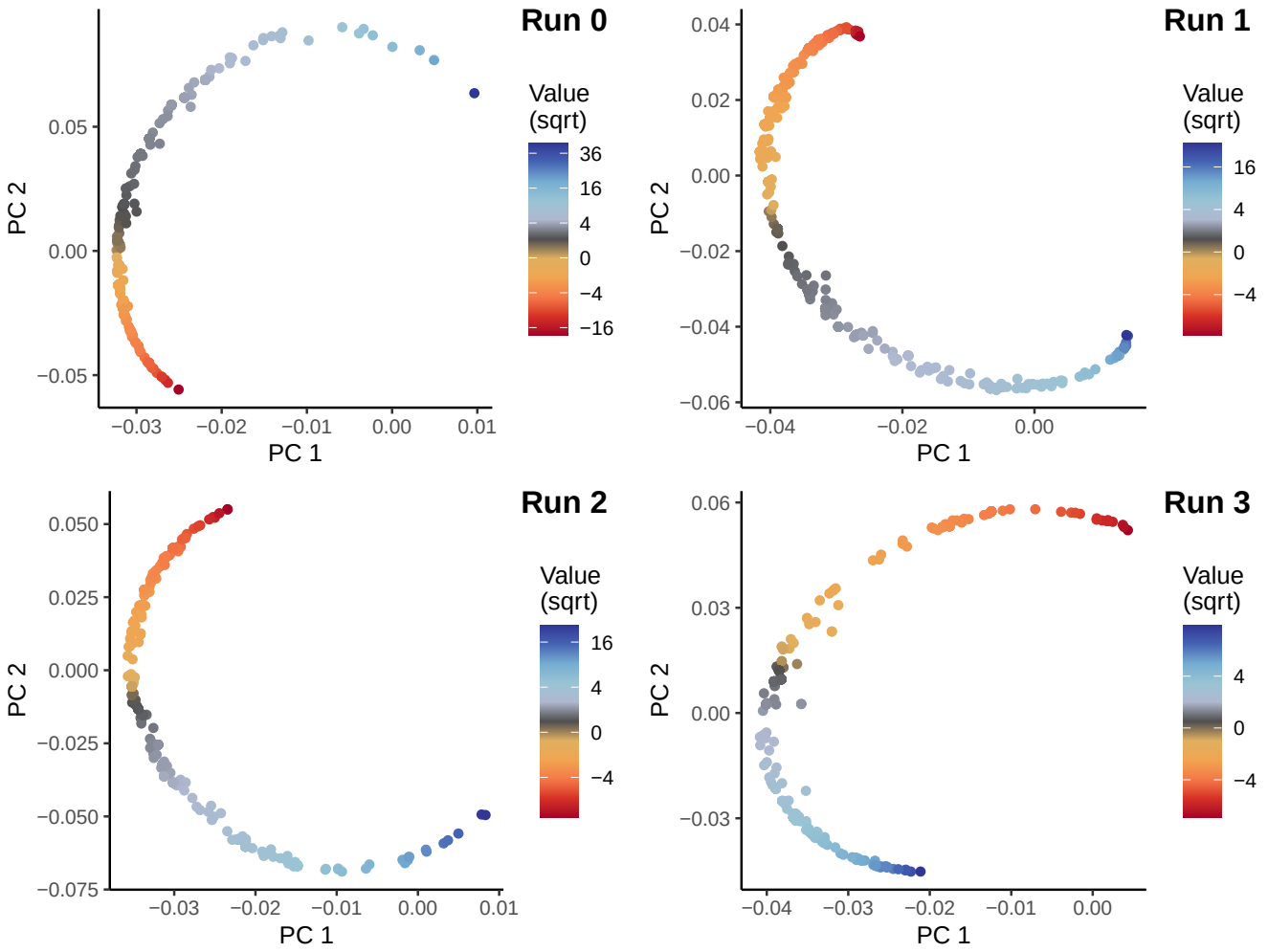


Fig. S13. Principal components of constant polynomial representations, showing systematic organization by value. Intriguingly, this relationship appears to be systematically non-linear across runs. (PCs computed across all task representations, color scale of values is compressed with a square-root transformation.)

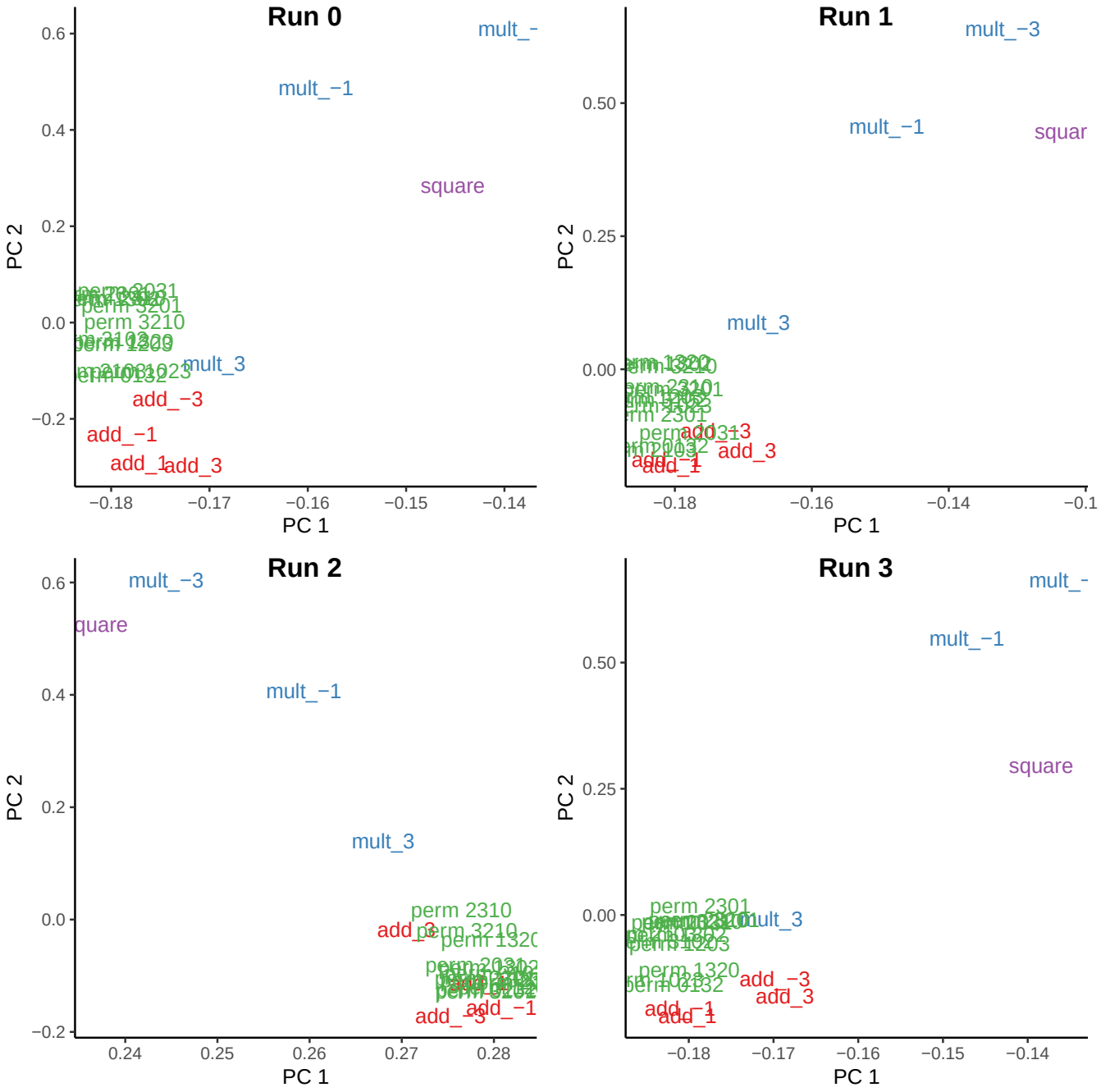
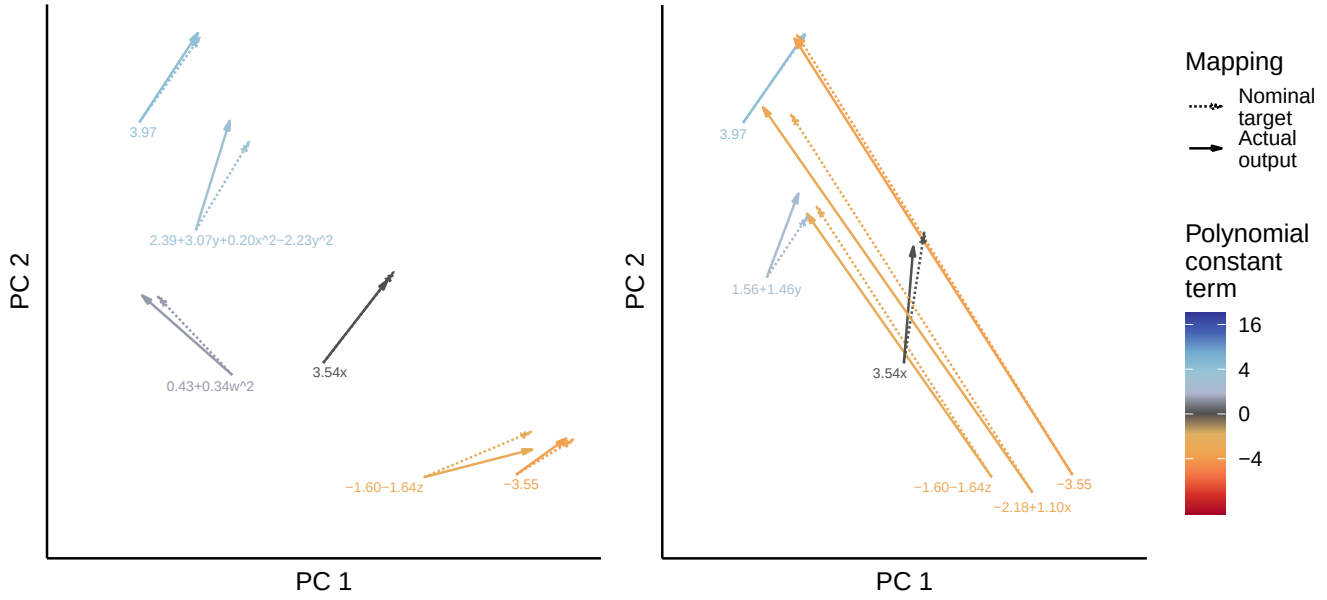


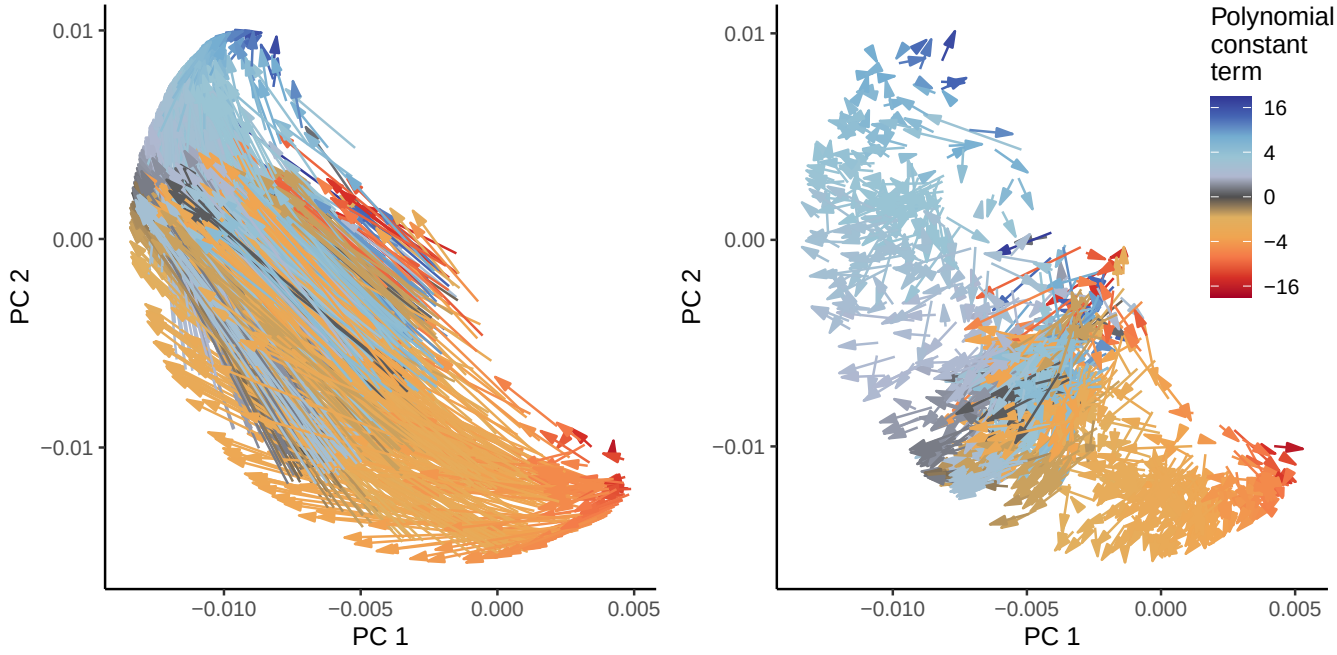
Fig. S14. Principal components of meta-mapping representations in the polynomial domain, showing systematic organization by type. Permutation mappings cluster tightly, as do addition, while multiplication and squaring are more dispersed. The addition and multiplication mappings are partially organized by absolute value.



(a) Meta-mapping: multiply by 3.

(b) Meta-mapping: square.

Fig. S15. The match between the meta-mapping outputs and the nominal targets (higher-resolution versions of the inset figures from Fig. 4). (a) The multiply by 3 meta-mapping. (b) The square meta-mapping. The meta-mapping outputs are generally close to the nominal targets (and note that mismatch does not necessarily indicate a mistake, see main text).



(a) Meta-mapping: add 3.

(b) Meta-mapping: permute (z, y, x, w).

Fig. S16. Visualizing how other meta-mappings transform the polynomial model representations (compare to Fig. 4). (a) The add 3 meta-mapping. Adding a constant results in rotation of the polynomial representations, and a slight outward expansion (as the polynomials become relatively more dominated by their constant terms). (b) A permutation meta-mapping which affects all variables (note: only non-constant polynomials are included in this panel). The reorganization of the space under the permutation is difficult to interpret, likely because the structure of the polynomial variables is higher dimensional, and involves many more principal components.

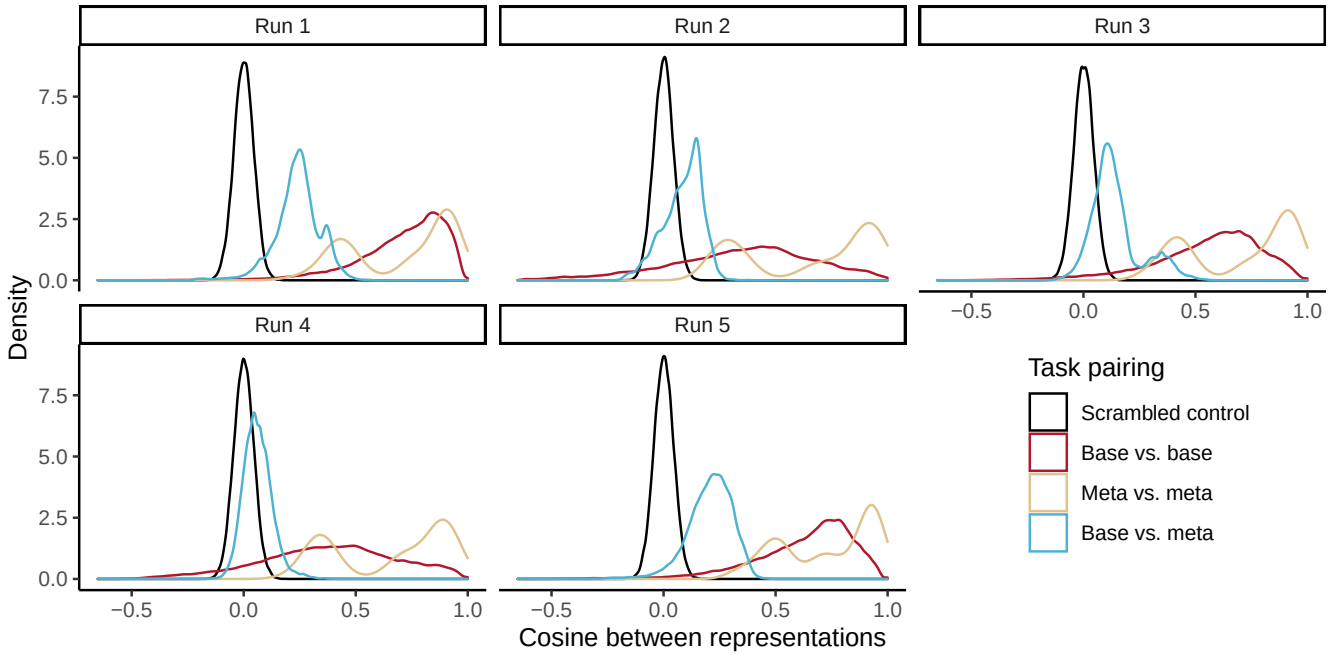


Fig. S17. There is non-trivial overlap between the representations of meta-mappings and base tasks in the polynomials domain. This figure plots the cosine similarity between different groups of representations, base tasks vs. base tasks. meta tasks vs. meta tasks, base tasks vs. meta tasks, and a control similarity distribution from a scrambled representation matrix. Although base tasks are more similar to other base tasks than to meta tasks, there is more similarity between the base and meta representations than would be expected by chance, though the absolute amount varies from run to run.

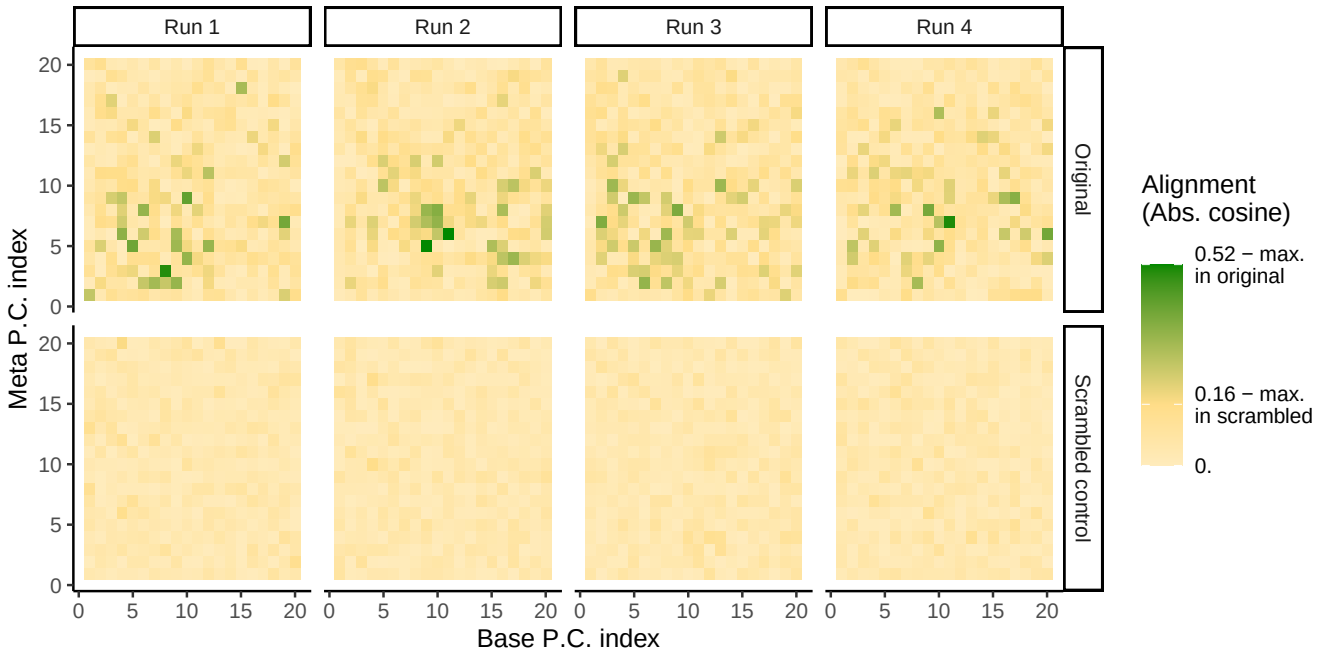


Fig. S18. There is nontrivial overlap between the top 20 principal components of the base task representations, and the top 20 principal components of the meta task representations, in the polynomials domain. For each run (columns), the top panel shows the alignment (abs. cosine similarity) between base task PCs (x-axis) and meta task PCs (y-axis). The bottom panels show the same results for a matched control (a scrambled representation matrix). The color scale is set so that cells are colored green only if the alignment is larger than any alignment observed in any control matrix. There are strong and relatively sparse alignments between the principal components of the basic- and meta-tasks, showing that the representations are not residing in orthogonal subspaces, and suggesting that homoiconicity is contributing non-trivially to the representation structure.

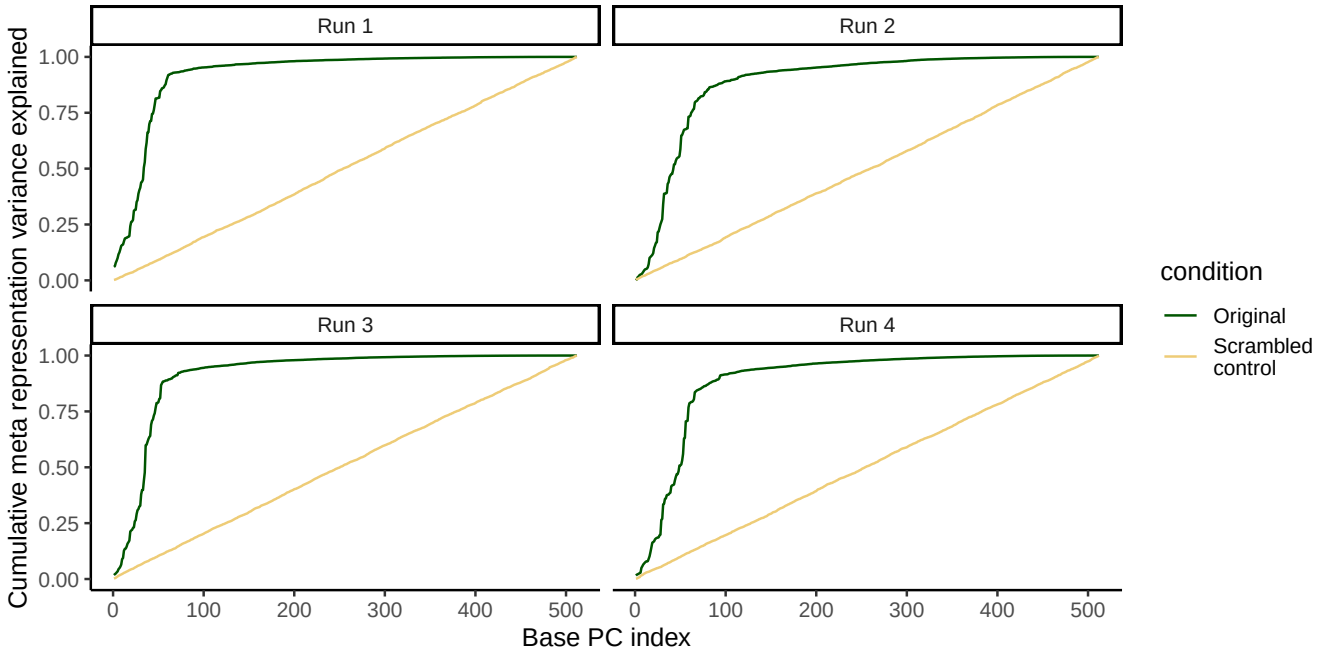


Fig. S19. The meta-task representation variance is preferentially distributed in the top principal components of the base task representations. For each run (panels), this plot shows the cumulative meta-task representation variance (vertical axis) explained by the base task representation principal components (horizontal axis). The dark green line shows the actual results, while the yellow line shows the results for a matched control (scrambled representation matrix). The meta-task representation variance is mostly contained in the earlier (more important) base principal components, again suggesting that homoiconicity is contributing non-trivially to the representation structure.

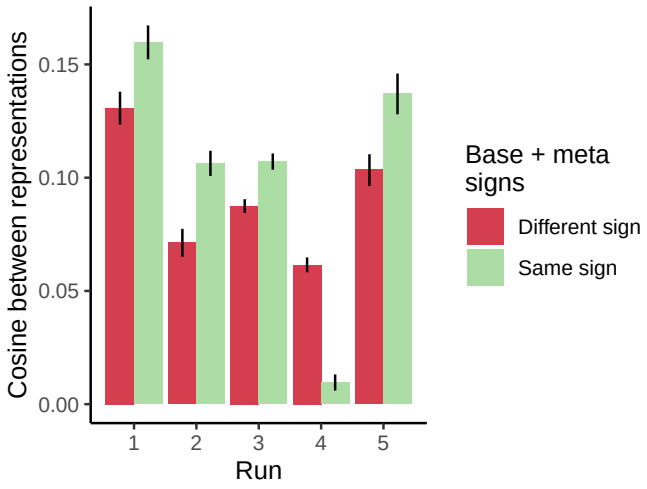


Fig. S20. There is significant organization of the multiplication meta-mappings by sign, in alignment (or anti-alignment) with the signs of the constant polynomials. This plot shows cosine similarity between representations of meta-mappings for the trained multiplication tasks (multiply by -3, -1, and 3) and the constant polynomials, depending on whether the multiplication value (for the meta-mappings) and the constant value (for the basic tasks) have the same sign or different signs. The difference is significant in each run (all ts > 5.3, all ps $< 1 \cdot 10^{-6}$), with greater similarity when the signs are aligned in all runs except run 4, where the effect goes in the opposite direction. These results suggest that the homoiconic model may be exploiting homomorphisms between scalar values that appear in a constant polynomial, and scalar values that appear in a meta-mapping (note that the non-canonical sign-switching alignment in run 4 may nevertheless capture useful structure).

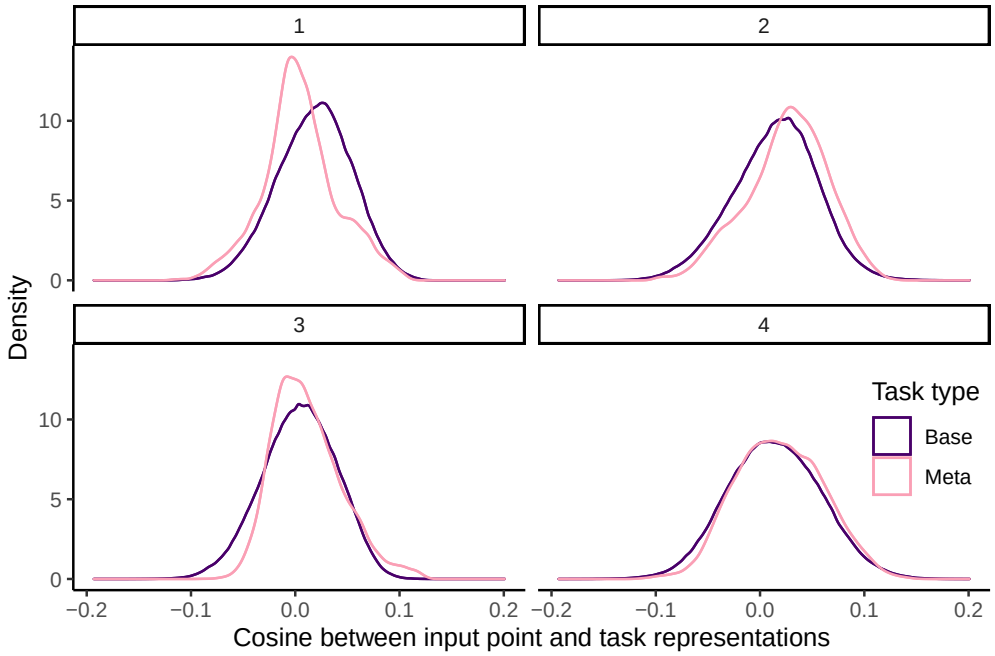


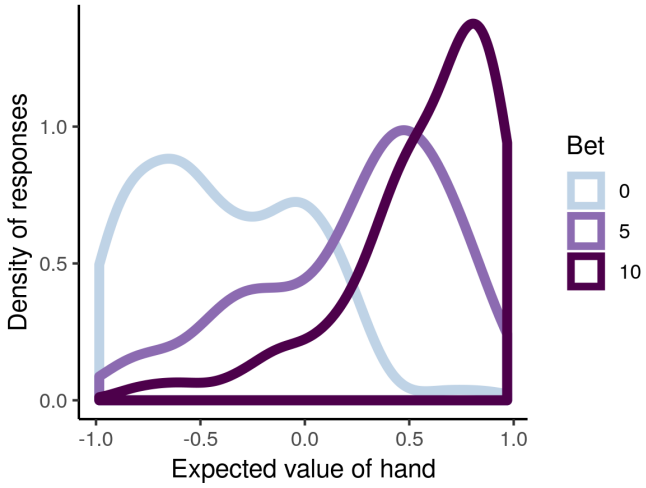
Fig. S21. The cosine similarity between basic input representations and basic task or meta-mapping representations is fairly small, likely reflecting the smaller amount of shared structure between these different entities, and the weaker constraints on alignment (see text for further discussion).

F.3. Cards. **Further analyses of human performance:** In Fig. S22 we show details of human participants performance on the card game tasks, including bet densities and subject-level fits of betting probability by hand value. As noted in the main text, the human subjects are performing far from optimally even in the trained task, and these figures show details on why this is true: subjects are both sub-optimal in finding the threshold at which to switch from betting to not betting, and are betting intermediate values, which an optimal better would not.

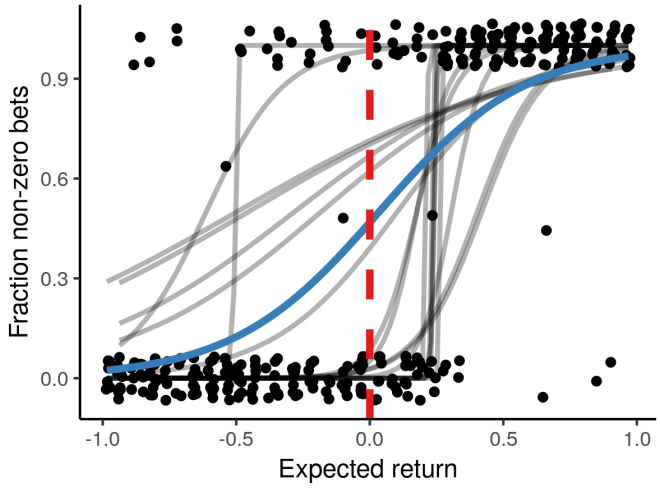
Basic meta-learning: In Fig. S23, we show that the basic meta-learning is working well in the cards domain. That is, we show that after the example network is presented with a set of example (hand, bet, reward) tuples, the system is generalizing well to other hands of that game. At the end of training, the mean reward on trained games is 99.20% of optimal (bootstrap 95%-CI [98.90, 99.40]), and for held-out games it is 83.82% (bootstrap 95%-CI [80.50, 86.00]).

Architectural comparisons: In Figure S9b we show that non-homoiconic architectures may perform slightly worse in the cards domain, but the difference is not significant. Specifically, the homiconic model is achieving an average expected reward of 85.38% (bootstrap 95%-CI [79.49, 90.32]), while the non-homoiconic model is achieving an average expected reward of 79.49% (bootstrap 95%-CI [69.50, 87.34]).

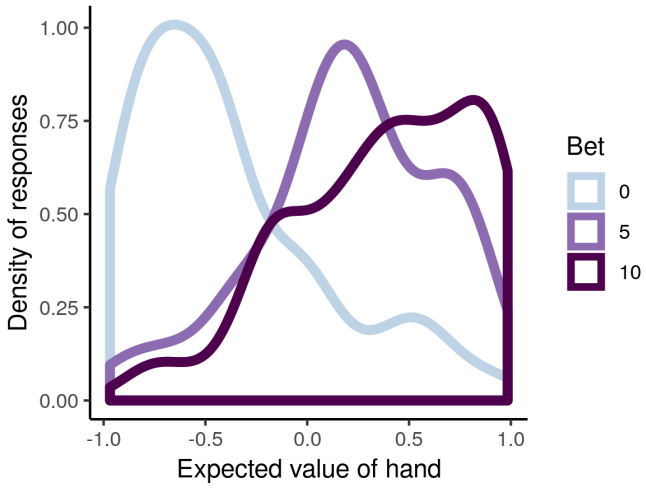
Meta-classification task lesion: In Figure S11b we show that meta-classification may be slightly beneficial in the cards domain, but the difference is small. Specifically, the model is achieving an average expected reward of 85.38% (bootstrap 95%-CI [79.49, 90.32]), while without meta-classification it is achieving an average expected reward of 78.68% (bootstrap 95%-CI [71.01, 85.97]). Because the meta-classifications appear to be more useful in this domain than in the polynomials domain, it is possible that they are particularly useful for understanding the structure of the task distribution when there are fewer basic training tasks. However, further work would be needed to verify this.



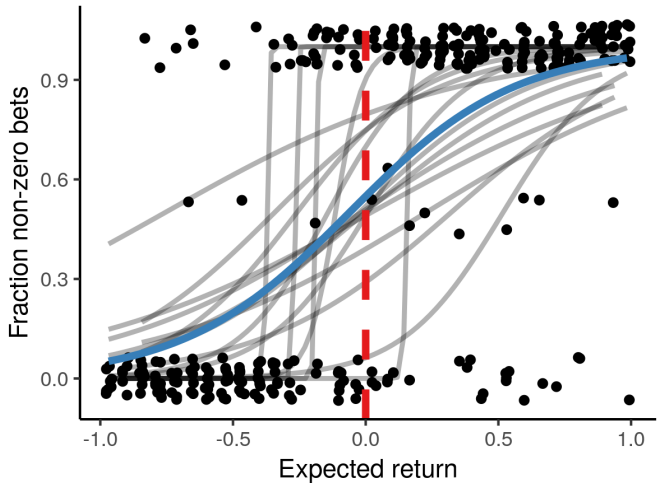
(a) Basic game: Bet density by expected value.



(b) Basic game: Probability of non-zero bet by expected value. The red dashed line is the optimal threshold, the grey curves are the individual subject fits.



(c) Losing variation: Bet density by expected value.



(d) Losing variation: Probability of non-zero bet by expected value. The red dashed line is the optimal threshold, the grey curves are the individual subject fits.

Fig. S22. Human performance on the card game task. Top row is basic game evaluation (before being told to lose), bottom is after being told to lose. While participants are performing well above chance, they are far from optimal. They make intermediate value bets, and do not switch optimally between betting and not betting depending on hand value. There is also substantial inter-subject variability.

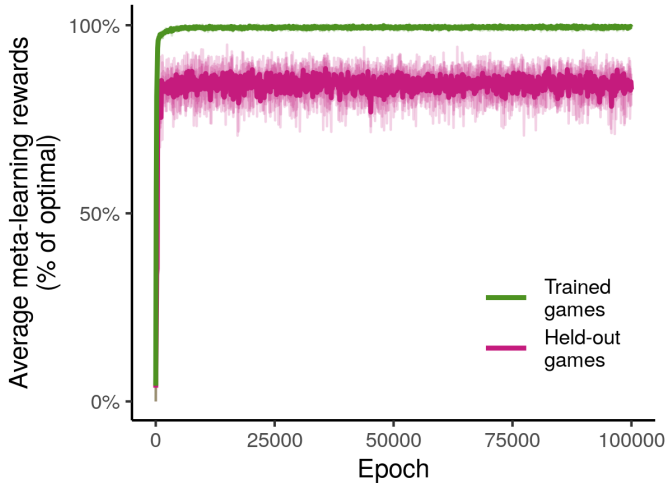


Fig. S23. Basic meta-learning performance in the cards domain over learning. The system is generalizing at the meta-learning level. That is, this graph shows that, after the example network receives a set of (hand, bet, reward) example tuples from a game, it is generating a sufficiently good representation of that game to play held-out hands. This is true both for gamess it was trained with (green), and for games that are held-out and never encountered during training (pink). (Thick dark curves are averages over 5 runs, shown as light curves.)

F.4. Visual concepts. In Fig. S24 we show the proportion of runs in which the model achieved $> 99\%$ performance; systematic generalization is increasingly likely as the number of training meta-mappings increases. In Fig. S25 we show learning curves for all runs of the meta-mapping model on these tasks.

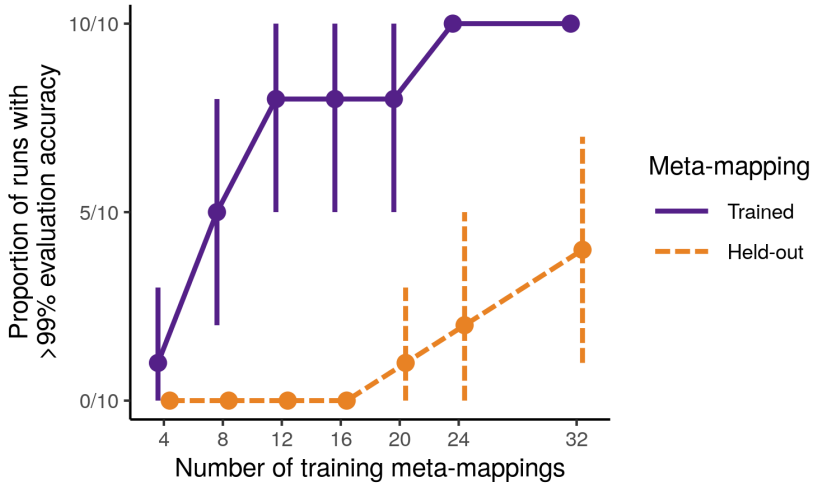
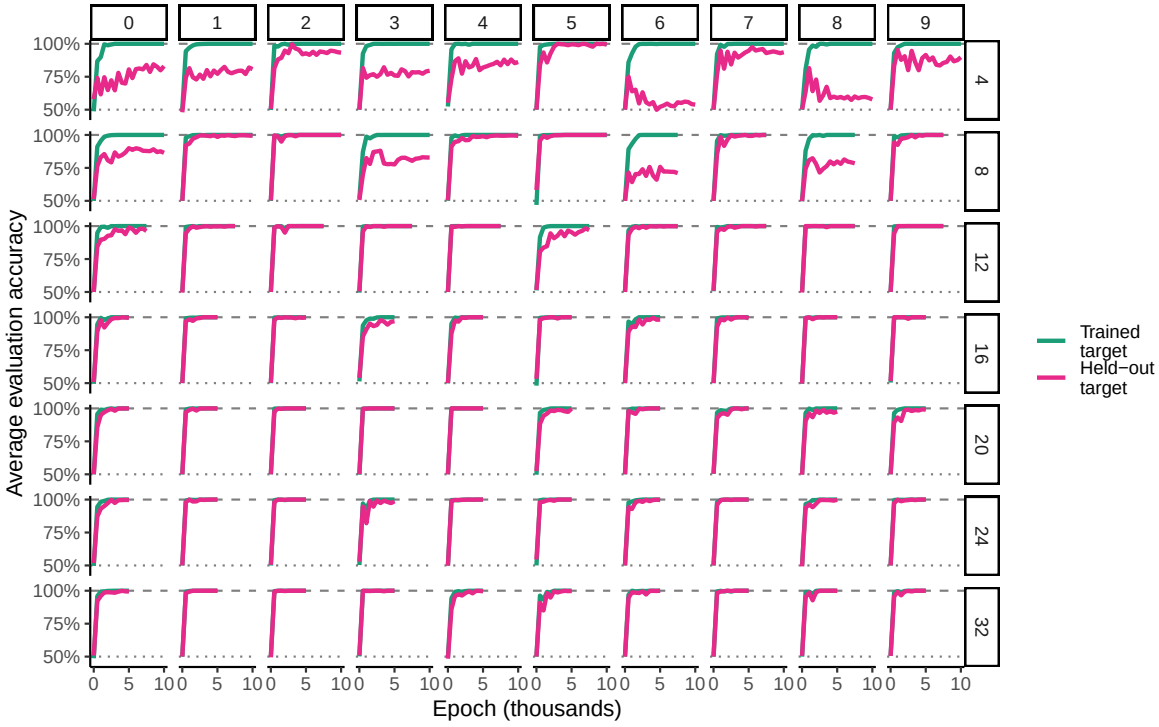
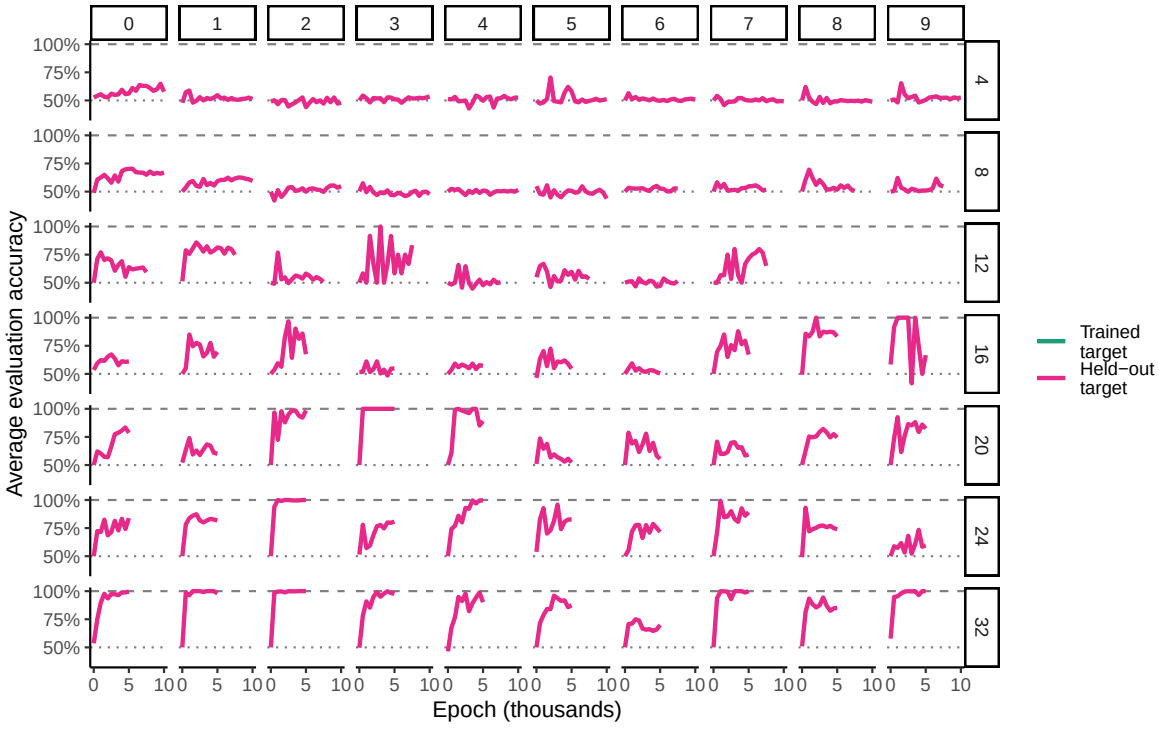


Fig. S24. In the visual concepts domain, the proportion of runs in which the model attained $> 99\%$ accuracy across all transformed concepts. The model shows extremely systematic generalization on trained meta-mappings at moderate sample sizes. At the largest sample sizes we considered, the model is able to adapt near-perfectly to new meta-mappings on many runs. Note that even at this largest sample size, the system is generalizing from only 32 trained meta-mappings.



(a) Trained meta-mappings.

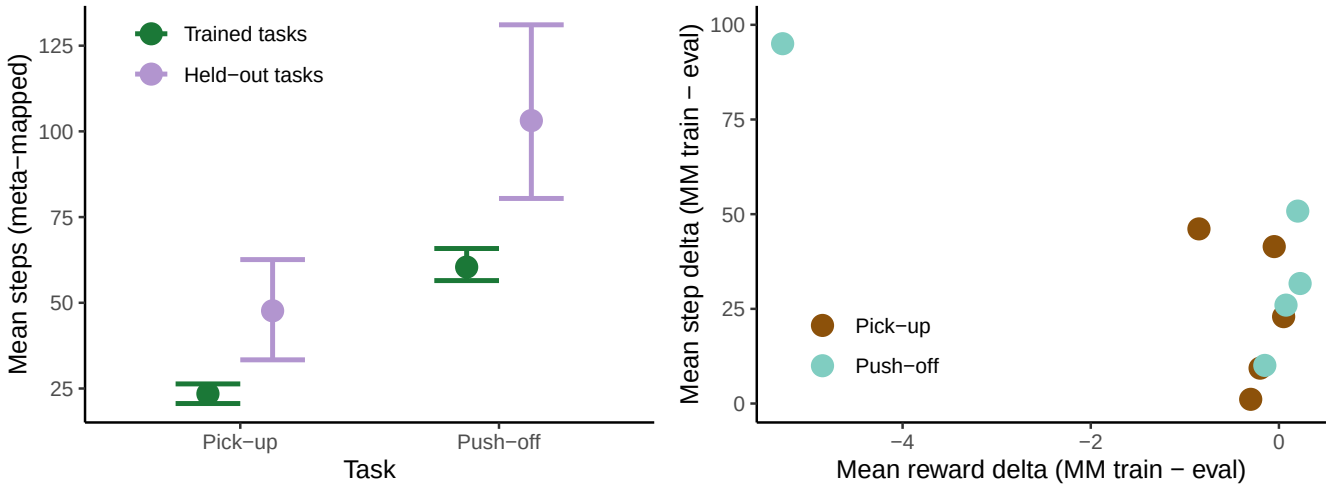


(b) Held-out meta-mappings.

Fig. S25. Meta-mapping performance (evaluated as average accuracy on the transformed task) in the visual concepts domain broken down by number of training meta-mappings (rows), and by run (columns). The green lines are performance when the transformed task was encountered during training, the pink lines are performance on transformed tasks that were never encountered during training. Panel (a) shows the results for trained meta-mappings, and panel (b) shows the results for held-out meta-mappings. With more training meta-mappings, generalization is better both when applying the trained meta-mappings to held-out examples (a), and when applying held-out meta-mappings (b). However, even with smaller sample sizes, the model is achieving perfect generalization on the trained meta-mappings on many runs. (The dotted line denotes chance performance, the dashed line optimal.)

F.5. RL. In Fig. S10b we also show that the HyperNetwork-based architecture performs better in this domain.

Behavioral uncertainty in generalization: In Fig. S26 we show intriguing behavioral uncertainty in generalization, where the model exhibits more uncertainty (takes longer to solve the task) even when it performs well. Selected recordings of behavior can be found at: https://github.com/lampinen/homm_grids/tree/master/recordings.



(a) Mean step counts.

(b) Differences in steps vs. differences in rewards.

Fig. S26. The model exhibits behavioral uncertainty in meta-mapping generalization on the RL tasks, measured by the steps taken to complete each episode. (a) The model takes more steps to complete episodes from the held-out tasks via a meta-mapping than to complete episodes from tasks used as training targets for the meta-mapping. That is, it appears to be more uncertain about its behavior on the generalization tasks. (b) The behavioral uncertainty effect is not solely driven by the model performing more poorly overall; even on the runs where it performs well, it is almost always taking longer to complete the episodes from the tasks it has never seen before. To show this, we plot the difference in average steps vs. difference in average rewards between train and eval. Note that the step difference is almost always positive (evaluation tasks are slower), even where rewards are comparable. (Panel a: means and bootstrap 95%-CIs across 5 runs. Panel b: each point is one game type within one run.)

F.6. Meta-mapping and language. In this section we show further figures and statistics corresponding for the language comparisons mentioned in the main text, and some supplemental analyses and discussion of the performance of these models.

RL: The language-alone model performs the trained tasks well, but adapts poorly, with generalization performance of -92.8% (mean, bootstrap 95%-CI [-96.3, -88.4]) on the pick-up task and -79.7% (mean, bootstrap 95%-CI [-92.8, -59.1]) on the pusher task. The difference between the models is significant ($t(20.6) = -19.515$, $p < 1 \cdot 10^{-14}$) in a mixed linear regression controlling for task type and a random effect of run.* Intriguingly, the language model does transiently exhibit slightly positive generalization very early in learning (see Fig. S27), but decays to below chance as the model masters the training tasks. This early generalization is not included in the main results since the train accuracy at this time is below the threshold of having adequately learned the tasks.

By contrast, meta-mapping with task representations constructed from language performs well, with generalization performance of 69.2% (mean, bootstrap 95%-CI [49.5, 84.5]) on the pick-up task and 74.9% (mean, bootstrap 95%-CI [60.9, 85.5]) on the push-off task. These models were trained separately from the language models whose results are reported below, but the language-alone generalization performance of even the models trained with meta-mapping is poor (respectively -79.6% [-95.0, -53.8] and -61.0% [-89.0, -0.195] on the two tasks). That is, meta-mapping at test time is key to generalization. Meta-mapping is not restructuring the basic task representations to allow better generalization from language alone. This is likely due in part to a memory limitation of the models, noted above — due to GPU memory constraints, meta-mapping training was not able to alter the construction of the basic task representations. If a future implementation of the model allowed this, meta-mapping training might be able to more directly improve basic-task generalization.

Cards: The language-alone model performed near-optimally at the trained tasks, but was not able to generalize well to the losing variation from the given dataset (mean performance on losing variation 2%, bootstrap 95%-CI $[-12, 16]$), see Fig. S28. Intriguingly, this corresponds to behaving approximately randomly; performance would be worse if the model did not adapt at all. In Fig. S29 we show that the poor language generalization is not simply due to the HyperNetwork architecture, by comparing to a task-concatenated architecture, as we did for meta-mapping in Fig. S10.

Visual concepts: In this setting the meta-mapping model and the language-alone model perform comparably (Fig. S30). In Fig. S31 we show that the language generalization is better with a more complex architecture (deeper & nonlinear) than we used for the meta-mapping approach. The comparisons in Fig.S30 use the better-performing architecture for each model.

The comparable performance in this domain may be due in part to the fact that our task sampling guaranteed a training task close to each evaluation task in this setting. This may be because of the structure of the task spaces; there are many more training visual concepts than training tasks in the other domains. Thus, while language-based generalization can be effective,

*Degrees of freedom calculated by the Satterthwaite approximation.

meta-mapping may be especially useful when there are relatively few training tasks — that is, it may be more sample efficient. However, another factor may be even more critical. The RL and Cards training tasks more directly contradict the evaluation tasks. By contrast, in the visual concepts domain our task sampling guarantees that each held-out concept will have a "nearby" training concept, one with the same relation type and same other attribute (see above). With less structured visual concept sampling, meta-mapping's advantage is slightly more clear (Fig. S32), even though the meta-mappings have less extensive support sets in that case.

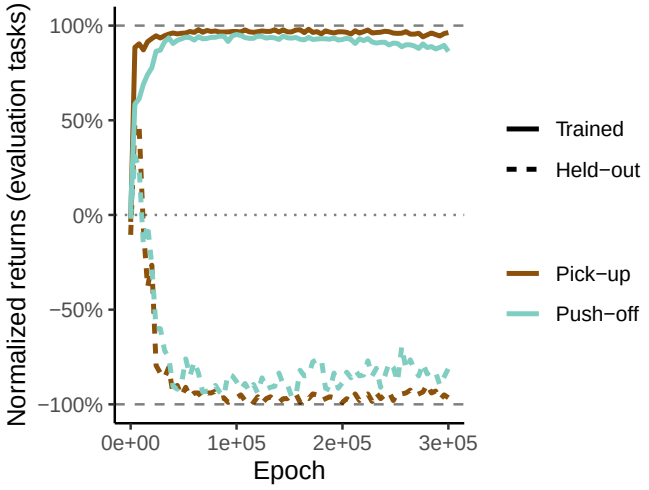


Fig. S27. Average performance of the language generalization model over training on the RL tasks. The model exhibits intriguing but transient generalization early in learning, before it has understood the full structure of the tasks (especially the more difficult and sequential push-off task), but delays to below-chance generalization as it masters the training tasks.

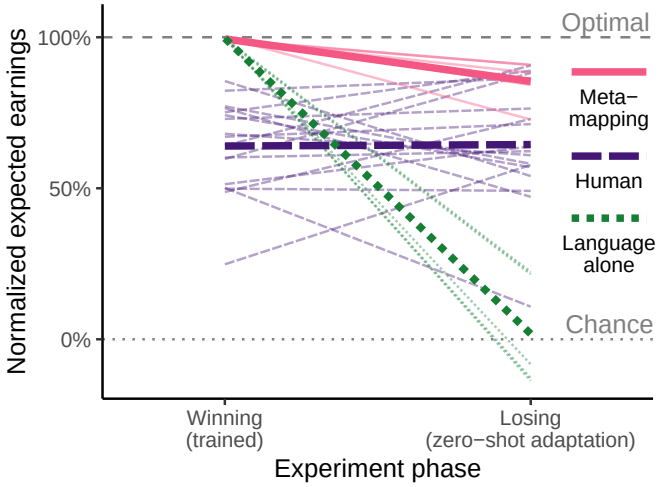


Fig. S28. Comparing language generalization to meta-mapping and human adaptation in the card games domain. The language-based model performs the trained tasks optimally, but degrades to chance performance on the losing variation. (We plot performance as expected earnings of the actions taken, as a percentage of the earnings of an optimal policy. Thick lines are averages, thin lines are 5 runs of each model, and 19 individual participants who passed attention checks.)

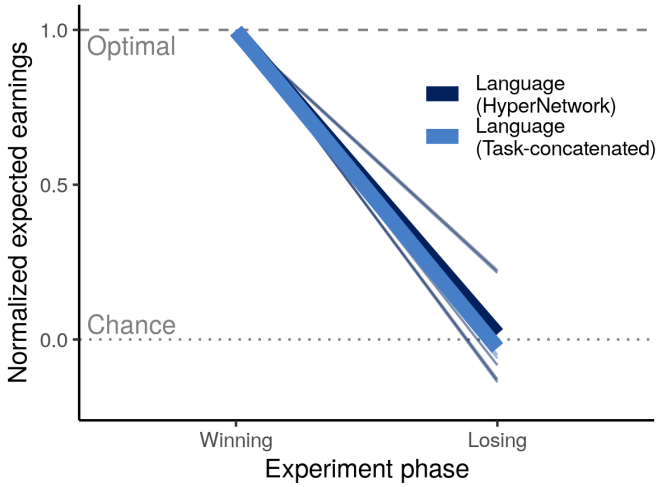


Fig. S29. Language generalization is similar in the cards domain with either the HyperNetwork architecture used by the meta-mapping model, or a simpler task-concatenated architecture. See Fig. S10 above for a similar comparison for meta-mapping itself.

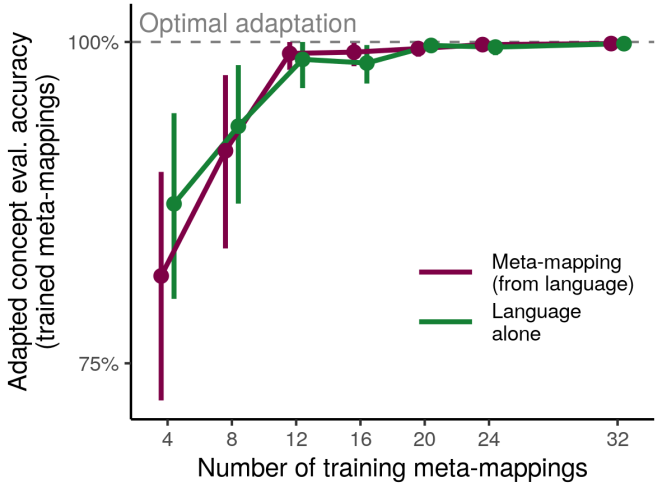


Fig. S30. Language generalization performs comparably to meta-mapping in the visual concepts domain, across training set sizes. (Results are from 10 runs for each model with each training set size. Errorbars are bootstrap 95%-CIs across runs.)

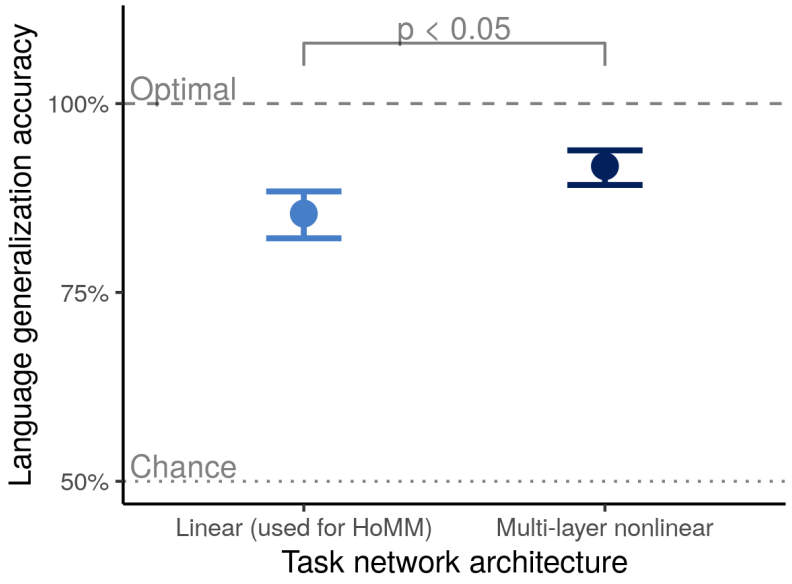


Fig. S31. Comparing language generalization on the visual concepts tasks between a linear task network architecture and a deep, nonlinear one. The nonlinear task network generalized better to new language instructions (comparisons shown are from the better version).

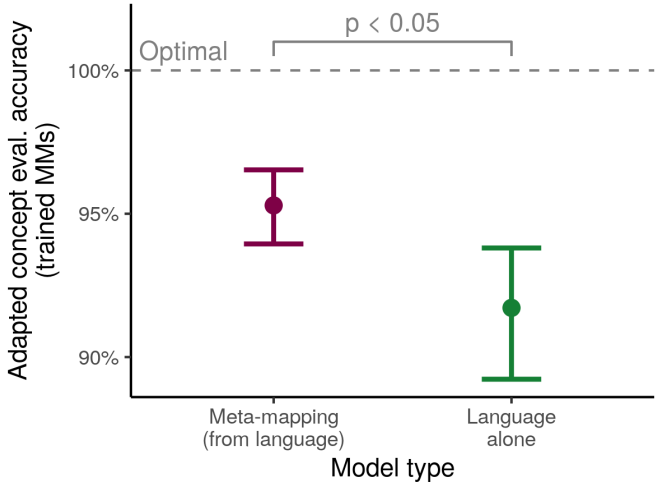


Fig. S32. Trained meta-mapping results in the visual concepts domain with 150 randomly sampled training concepts, rather than the structured sampling used in the main text. This task sampling scheme means that some evaluation tasks will be farther from the trained tasks. Meta-mapping has a correspondingly larger advantage here. However, the tasks are still likely to be closer to a trained task than in e.g. the RL setting where the evaluation tasks directly contradict the trained ones, and the language model is performing correspondingly better here than on the RL tasks.

F.7. Generalizing from color to shape in RL. We next evaluated the generalization capabilities of meta-mapping in a more challenging RL experiment. In this experiment, we trained the model on tasks similar to those in the main text experiments, but where the good and bad objects could be discriminated by either color (with shape matched) **or** shape (with color matched). We trained good-and-bad-switched variations of all color tasks, but did not train any switched variations of the shape-discrimination tasks. Specifically, we used 8 colors, of which we used 4 for the pick-up tasks and 4 for the push-off tasks (so the task type would still be superficially distinguishable. We trained color-discrimination between two pairs of colors in each type, when presented with either both colors appearing on square shapes, or both appearing on diamond shapes. We also trained switched-good-and-bad variations of all those color discrimination tasks. We then trained four shape discrimination tasks for each game type, one in each of that game type's four associated colors. In the shape discrimination tasks, the tee-shaped objects were always good, and triangular objects were always bad. (This results in a total of 24 training tasks, a larger number than were included in the main text experiments.)

We trained the "switch-good-and-bad" meta-mapping on the color discrimination tasks, and evaluated whether meta-mapping was able to correctly generalize this meta-mapping from switching colors to switching shapes, in order to infer that the triangular

objects, which had always been negatively rewarded before, were now beneficial. We found it was useful to increase the initial meta-mapping learning rate to $3 \cdot 10^{-4}$, but otherwise used the same hyperparameters as the main text experiments. See Fig. S33 for the results. We found that meta-mapping indeed allowed generalization well above chance. As in the main-text experiments, this is true whether meta-mapping is performed using task and meta-mapping representations constructed from examples (average returns across pick-up and pusher 64.3% percent of optimal, 95%-CI [55.1, 72.8]), or task and meta-mapping representations constructed from language (average returns across pick-up and pusher 68.3% percent of optimal, 95%-CI [56.6, 78.3]). These experiments show that meta-mapping is able to successfully extrapolate well beyond the training examples of the mapping, to transform behavior along new dimensions.

Intriguingly, the language-alone baseline model performed less poorly at these experiments than at the main text experiments, although its generalization was not statistically different from chance (average returns 17.8% of optimal, 95%-CI [-4.0, 37.4]). Note, however, that there are also 25% more training tasks in this setting than in the main text experiments. Furthermore, the performance of language alone was still substantially worse than either meta-mapping approach. In a mixed model controlling for game type and its interaction with model and the random effect of run, the difference in performance between meta-mapping from either examples or language and the language-alone performance were both significant (from examples $t(119.01) = 4.64$, $p = 8.9 \cdot 10^{-6}$, from language $t(119.04) = 3.79$, $p = 2.4 \cdot 10^{-4}$). The effect of game type on generalization in the language model was not significant ($t(119.02) = 1.18$, $p = 0.24$), nor were the interactions of game-type with either model type (respectively, the interaction of meta-mapping from example by game-type $t(119.01) = -1.522$, $p = 0.13$ and from language by game-type $t(119.03) = -0.08$, $p = 0.94$).

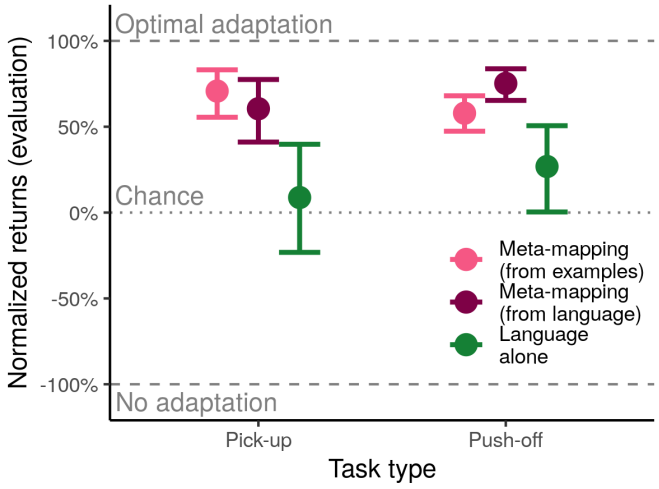


Fig. S33. Meta-mapping can generalize switching good and bad objects from the color dimension to the shape dimension. In this experiment, we trained meta-mapping on tasks similar to those in the main text experiments, but where the good and bad objects could be discriminated by either color (with shape matched) **or** shape (with color) matched. We trained good-and-bad-switched variations of all color tasks, but did not train any switched variations of the shape-discrimination tasks, to evaluate whether meta-mapping was able to infer how to transfer a mapping from switching colors to switching shapes. Indeed, meta-mapping performs well above chance at this task, though not quite as well as on the simpler generalization in the main text. Intriguingly, the language model also appears to be perfoming somewhat better in this setting, though it is not statistically above chance. (Results from 5 runs, see the text for further details of the experimental setup.)

F.8. Meta-mapping as a starting point. **Visual concepts:** In Fig. S34 we show that meta-mapping provides a good starting-point for learning in the visual concepts domain as well. In this setting the small random initialization is more competitive, but meta-mapping still yields lower cumulative error over learning than random initialization, and much lower than the centroid (which was better in the polynomials domain). Specifically, initializing with a meta-mapping output results in a mean cumulative error of 0.33 (bootstrap 95%-CI $[0.10, 0.57]$), while a small random initalization results in a mean cumulative error of 9.62 (bootstrap 95%-CI $[6.63, 13.59]$). This difference is significant in a mixed linear model ($t(4) = 4.628$, $p = 0.01$).

The non-hyper-network architecture makes optimization more difficult: We have compared our hyper-network-based meta-mapping architecture to the simpler alternative of concatenating a task representation to an input embedding before passing it through a fixed network, in various supplemental analyses (Figs. S10 and S29). The hyper network approach generally performs at least as well as, and sometimes substantially better than, the simpler approach. Hyper networks may also be particularly beneficial for continual learning (10). Furthermore, they may also make it easier to optimize the task representation, by giving it more direct control over the computations of the network. Thus, it seems useful to compare these two architectures in this setting.

We therefore performed the polynomial domain experiments, reported in the main text in the meta-mapping as a starting point section, with the simpler task-network architecture as well. In Fig. S35, we show the learning curves for both architectures for the two best initializations (meta-mapping output, and centroid of the trained task representations). The hyper-network architecture learns much more rapidly than the simpler architecture. The initial meta-mapping outputs do not differ so substantially — most of this effect is due to the slower improvement of the loss when optimizing the task representation in the

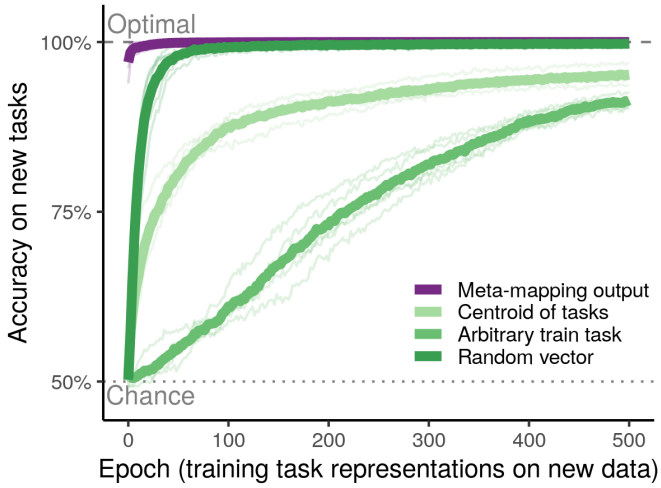


Fig. S34. Meta-mapping provides a good starting point for later learning in the visual concepts domain. This figure is the visual concepts analog of Fig. 10 in the main text, with 16 training meta-mappings. Using meta-mapping as a starting point offers much lower initial loss, and faster learning than other initializations. (Thick curves are averages over 5 individual runs, shown as light curves.)

non-hyper architecture. Indeed, optimization in the non-hyper network architecture appears to be plateauing at a much higher loss value than in the hyper-network architecture.

As before, we quantify this by plotting the cumulative loss on the novel tasks in Fig. S36. The simpler non-hyper architecture resulted in about five times greater cumulative loss than the hyper network architecture when starting from the meta-mapping output (mean $= 133.81$, bootstrap 95%-CI $[102.65, 171.10]$), and similarly from the centroid of the trained task representations (mean $= 1139.35$, bootstrap 95%-CI $[943.60, 1344.52]$). We therefore conclude that hyper-network-based architectures may be particularly conducive to this perspective on continual learning.

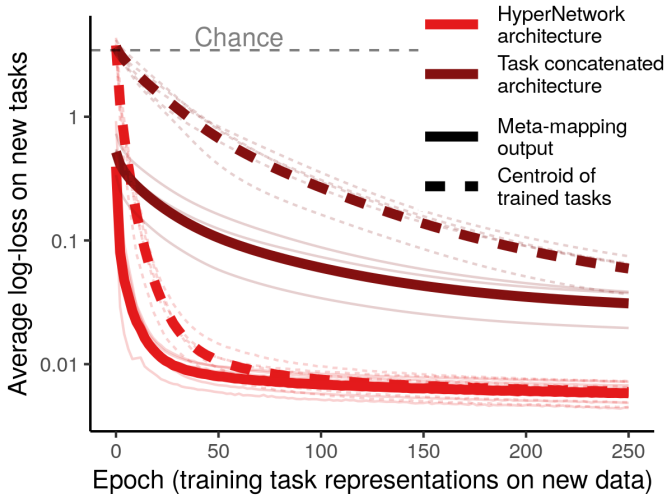


Fig. S35. Comparing the learning curves of the hyper network architecture and a simpler architecture when optimizing the task representations for new polynomials. The simpler architecture improves much more slowly, and appears to plateau at a higher loss. (Note that the y-axis is log-scale. Results are from 5 runs, individual runs are shown as light curves.)

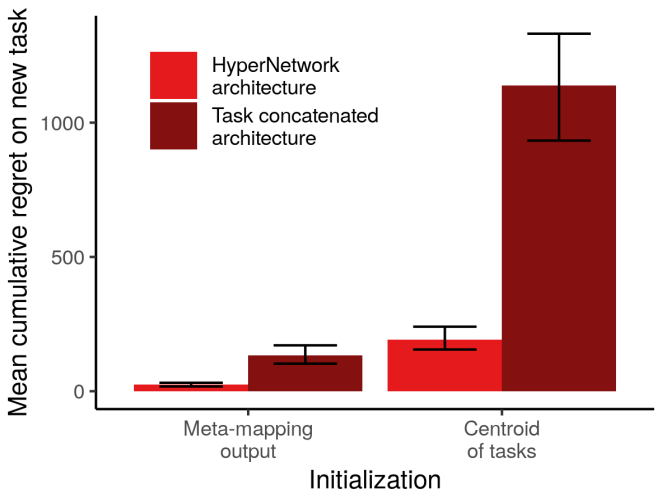


Fig. S36. Comparing the cumulative losses of the hyper-network architecture and a simpler architecture when optimizing the task representations for new polynomials, starting from either the result of a meta-mapping or the centroid of the trained tasks. The simpler architecture results in substantially more cumulative loss. (Results from 5 runs, errorbars are bootstrap 95%-CIs.)

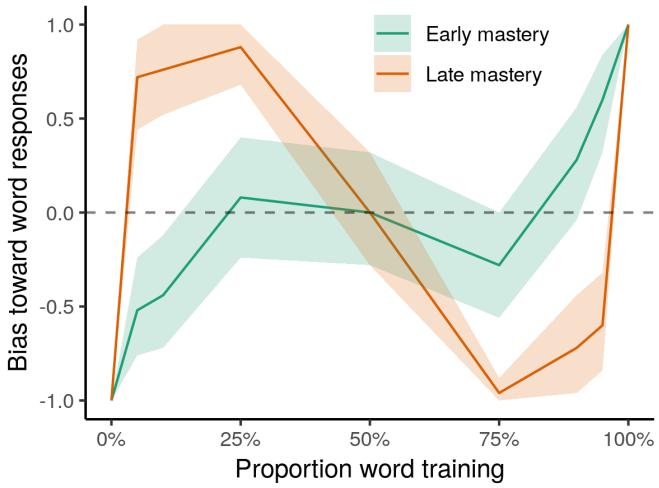


Fig. S37. Measuring the default behavior of our architecture on a Stroop-like task. We plot the bias of the model towards word or color responses, when given an all-zeros task representation, at different proportions of training on words or colors, and different stages of training. When the model has just mastered the less frequent task, it exhibits a default bias towards the more frequent task. However, later in training, when it has mastered both tasks, it exhibits a paradoxical bias towards the *less* frequent task.

F.9. Default processing & cognitive control. Our architecture could be of interest to researchers in cognitive control, even beyond the idea of meta-mapping as adaptation. The system can perform different tasks based on task examples or language inputs, which is fundamentally the same problems human face when we must adapt our behavior. There are a number of features of the model that offer the opportunity for intriguing investigations based on this idea. For example, the task network in our architecture has a default set of bias weights that are modulated by the HyperNetwork. These can be thought of as the "automatic" or "default" processing habits of the system, whereas the weight alterations the HyperNetwork imposes can be thought of as the exertion of cognitive control to modulate behavior.

To explore this, we trained our architecture on a very simple stroop task taken from Cohen et al. (11). The model receives two sets of two inputs, that can be thought of as corresponding to "word" and "color" domains. One input in each domain is turned on, representing a color word written in a color. The model's task is to report either the color or the word, depending on context.

The context we give the model is in the form of examples of the task as (input, output) pairs. These are used to construct a task representation, which is then used to modulate the parameters in the task network, via the HyperNetwork. We trained the model repeatedly with different proportions of training on the word task vs. the color task, in order to investigate the default vs. controlled behavior in different training regimes. Specifically, we compared training the model to the point that it barely mastered the less frequent task (when it first achieves 100% performance and cross-entropy loss < 0.3 on both tasks) to the point that it had mastered both tasks (100% performance and cross-entropy loss < 0.01 on both). We then tested the

model's default behavior by giving it an all-zeros task representation, and seeing whether its performance was more aligned with the "word" or "color" task.

In Fig. S37, we show the results. We plot the bias as $2 \times$ (word accuracy $-$ color accuracy), which is -1 if the model is responding only to color, 1 if the model is responding perfectly to word, and 0 if it is responding equally to each (or otherwise responding randomly). When the model has just barely mastered the less-frequent task, it exhibits a default bias towards the more frequent task. However, once we train it to full master of both tasks, it exhibits a surprising paradoxical bias towards the task that was mastered more recently. This may relate to observations that switching from a less-practiced task back to a more practiced one is difficult (12), possibly because performing the less-practiced task requires strong suppression of the default behavior. It's possible that in the course of achieving full mastery on the less-practiced task, the more practiced task must be so suppressed that it fades away from being the default. These phenomena provide possible inspiration for future investigations in cognitive control.

For this experiment, we used similar hyperparameters to the polynomials experiments, except we used a much smaller model — a single-layer task network, a Z-dimensionality of 8, and $\mathcal{H}, \mathcal{E}$ had 64 hidden units per layer. We optimized the model via stochastic gradient descent with a learning rate of 0.01 to follow more closely the approach taken by Cohen et al., although results are similar with other optimizers.

G. Proofs.

G.1. Inadequacy of vector analogies for meta-mapping polynomials. One possible implementation of meta-mapping would be to just construct an analogy vector and use that for the mapping. This is motivated by work showing that word vector representations often support vector analogical reasoning, for example if we denote the vector for the word king as $\vec{v}_{king}$, relationships like $\vec{v}_{queen} \approx \vec{v}_{king} + (\vec{v}_{man} - \vec{v}_{woman})$ often hold (13). Thus, a plausible approach to meta-mapping would be to take a similar approach, for example in the polynomials domain, the meta-mapping "Permute (w, z, x, y)" could be estimated by taking the vector differences between the representations of inputs and targets, computing an average difference vector, and adding that to the held-out examples to produce an output for each one. In this section, we prove that such an approach cannot accurately represent all the meta-mappings in the polynomials domain. Furthermore, we sketch a proof by construction that the linear task network (i.e. an affine transformation, matrix multiplication plus a bias vector) we used in this domain suffices, if it is parameterized separately for each meta-mapping.

Proof that vector analogies are inadequate: In essence, the proof is simply that many of our meta-mappings are non-commutative, while vector addition is commutative. Consider the mappings for adding 1 to a polynomial, and multiplying by 2. Assume there were vector representations for these mappings, respectively $\vec{m}_{+1}$ and $\vec{m}_{\times 2}$. Let $\vec{f}_x$ be the representation for the polynomial $f(w, x, y, z) = x$. Then $\vec{f}_x + \vec{m}_{+1} = \vec{f}_{x+1}$, $\vec{f}_x + \vec{m}_{\times 2} = \vec{f}_{2x}$. But then:

$$\vec{f}_{2(x+1)} = \left(\vec{f}_x + \vec{m}_{+1}\right) + \vec{m}_{\times 2} = \vec{f}_x + \vec{m}_{+1} + \vec{m}_{\times 2} = \left(\vec{f}_x + \vec{m}_{\times 2}\right) + \vec{m}_{+1} = \vec{f}_{2x+1}$$

Thus such a representation would result in contradictions, such as $2x + 1 = 2x + 2$. Similar issues occur for input permutation and other non-commutative mappings.

Proof sketch that affine transformations in an appropriate vector space suffice: Suppose that we have a vector representation for the polynomials, where there is a basis dimension corresponding to each monomial, so that the polynomial can be represented as a vector of its coefficients. (This is the standard vector-space representation for polynomials.) Then permutation corresponds to permuting these monomials, i.e. a permutation of the basis dimensions, which is a linear transformation. Adding a constant corresponds to adding to one dimension, which requires only the vector addition part of the affine transformation. Multiplying by a constant requires multiplying each dimension, i.e. a block-diagonal linear transformation.

Squaring polynomials is slightly more complex, and requires augmenting the vector space with components whose values are the product of the coefficients of each pair of monomials. In this case, squaring corresponds to a simple linear transformation. However, this augmentation makes the other meta-mappings more complex. Surprisingly, the most complex case in this representational scheme is adding a constant, which requires shifting each pair term containing a constant by the product of the constant and the coefficient of the other monomial, but this again reduces to simply an appropriately parameterized affine transformation — each pair term containing a constant term simply needs the added constant (from the meta-mapping) as a weight times the component for the other monomial. Thus affine transformations suffice in this setting.

Of course, with a sufficiently complex, deep, recurrent, and non-linear task network, any meta-mapping could be computed in principle, since a sufficiently large such network is Turing-complete (14). Thus, our approach to meta-mapping is fully general, conditioned on a sufficiently complex task network, while simpler approaches may not be.

References

1. T Salimans, DP Kingma, Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. *Adv. Neural Inf. Process. Syst.* (2016).
2. B Xu, N Wang, T Chen, Empirical evaluation of rectified activations in convolution network. *arXiv preprint arXiv:1505.00853* (2015).
3. X Glorot, Y Bengio, Understanding the difficulty of training deep feedforward neural networks. *Proc. 13th Int. Conf. on Artif. Intell. Stat. (AISTATS)* **9**, 249–256 (2010).

4. AM Saxe, JL McClelland, S Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *Adv. Neural Inf. Process. Syst.*, 1–9 (2013).
5. Y Li, C Wei, T Ma, Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks. *Adv. Neural Inf. Process. Syst.*, 1–49 (2019).
6. V Mnih, et al., Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
7. AK Lampinen, *A Computational Framework for Learning and Transforming Task Representations*. (PhD Dissertation, Stanford University, https://stacks.stanford.edu/file/druid:xj689nb3522/dissertation-augmented.pdf), (2020).
8. M Harrower, CA Brewer, ColorBrewer. org: an online tool for selecting colour schemes for maps. *The Cartogr. J.* **40**, 27–37 (2003).
9. KL Hermann, AK Lampinen, What shapes feature representations? Exploring datasets, architectures, and training. *arXiv preprint* (2020).
10. JV Oswald, C Henning, J Sacramento, BF Grewe, Continual learning with hypernetworks. *Int. Conf. on Learn. Represent.*, 1–25 (2020).
11. JD Cohen, K Dunbar, JL McClelland, On the control of automatic processes: A parallel distributed processing account of the stroop effect. *Psychol. Rev.* **97**, 332–361 (1990).
12. S Monsell, Task switching. *Trends Cogn. Sci.* **7**, 134–140 (2003).
13. T Mikolov, Wt Yih, G Zweig, Linguistic regularities in continuous space word representations. *Proc. NAACL-HLT*, 746–751 (2013).
14. HT Siegelman, ED Sontag, On the computational power of neural nets. *Proc. fifth annual workshop on computational learning theory* (1992).