Using transfer learning to identify a neural system's algorithm

John Morrison[1,2,3], Nikolaus Kriegeskorte[3,4,5], Benjamin Peters[6]

[1]Department of Philosophy, Barnard College, Columbia University
[2]Program in Cognitive Science, Barnard College, Columbia University
[3]Zuckerman Mind Brain Behavior Institute, Columbia University
[4]Department of Psychology, Columbia University
[5]Department of Neuroscience, Columbia University
[6]School of Informatics, University of Edinburgh

Keywords: algorithms; transfer learning; cognitive neuroscience; systems neuroscience; artificial neural networks; philosophy of cognitive neuroscience

Corresponding Author:
John Morrison
Email: jmorrison@barnard.edu
Department of Philosophy
Barnard College
3009 Broadway, New York, NY 10027

Code: All of the code used in this study is available at https://anonymous.4open.science/r/transfer-learning-for-algorithms-D372/.

Abstract

Cognitive psychologists use algorithms to explain cognition. For example, they use tree-search algorithms to explain planning, reinforcement learning algorithms to explain exploration, and Bayesian algorithms to explain categorization. Some neuroscientists and machine learning researchers hope to use these algorithms to better understand biological and artificial neural systems. A standard method is to look for parts in the neural system corresponding to the parts of the algorithm, thereby treating the algorithm as a causal model of the neural system. However, this method has struggled to find the algorithms used in cognitive psychology, leading others to view these algorithms as merely useful fictions – useful for predicting a neural system's outputs, but fictional as causal models of its operations. As an alternative method, we suggest identifying a neural system's algorithm by assessing how quickly it learns new input-output mappings, that is, its transfer learning profile. The basic idea is that different algorithms make some input-output mappings easier to learn than others, so we can infer a system's algorithm from how easily it can learn different input-output mappings. We use artificial neural networks to demonstrate that this method productively applies to a variety of neural systems and tasks. We conclude that transfer learning is a promising approach for integrating cognitive psychology algorithms with neural systems and thus for integrating cognitive psychology with systems neuroscience and machine learning.

Significance

Cognitive psychology algorithms have the potential to provide simple explanations of complex neural systems. The standard methods for identifying these algorithms look for parts of algorithms within the system. However, these methods often fail. We use networks' transfer learning speeds to reveal algorithms that standard methods miss. In particular, we consider networks that appear similar according to standard parts-based methods yet learn new tasks at different speeds, indicating different underlying algorithms. These results raise the question of what ultimately grounds the attribution of algorithms to neural systems, with consequences for how cognitive psychology relates to systems neuroscience and machine learning.

Introduction

Algorithms transform inputs into outputs through a series of operations on intermediate representations. They were originally invented to supplement cognition. For example, column addition was introduced to add large numbers, predicate logic to deduce theorems [1], merge sort to organize data [2], tree-search to play chess [3], recursive grammars to analyze sentences [4], signal detection theory to identify planes [5], Bayesian inference (e.g., conditionalization) to update scientific beliefs [6], and reinforcement learning (e.g., Q-learning) to choose actions that maximize cumulative reward [7].

Cognitive psychology's bold conjecture is that we can also use algorithms to *explain* cognition. This conjecture has been tremendously productive; there are now algorithmic explanations of a wide range of different kinds of cognition, including perception, memory, language, and decision-making [8]. The appeal of these explanations is that they typically abstract away from the details of their implementation in neural systems, thereby providing a simple explanation of a complex phenomenon. But this abstraction comes at a cost: identifying these algorithms in neural systems can be challenging.

The challenge is conceptual as well as empirical. Consider an algorithm that merely describes a network's activity neuron-by-neuron or node-by-node. Identifying that algorithm poses only an empirical challenge – it is clear what to look for, even if it is often hard to measure the relevant activity. Cognitive algorithms, by contrast, typically abstract away from these details. As a result, there is an added conceptual challenge: What should we look for when trying to identify these algorithms in neural systems?

A standard method is to search for parts in a neural system that correspond to parts of an algorithm. Here, a neural system's "parts" include its basic components such as weights and activations as well as any mathematically definable construct of these basic elements, such as points, regions, dimensions, and vectors within the system's space of possible activities. These parts can be identified using a wide range of analyses, including decoding, representational similarity analyses, and an analysis of learned weights [9–13]. The assumption underlying this method is that for an algorithm to causally explain a neural system, rather than merely describe its input-output behavior, the algorithm's parts must correspond to causally relevant parts in the neural system.

This parts-based method has had a number of successes. In neuroscience, they include path integration in the ring ellipsoid body of flies [14], sound localization in the nucleus laminaris of chickens and owls [15], central pattern generation in the spinal cord of rodents and other vertebrates [16], neural integration theory in the brainstem of rhesus monkeys and other primates [17], sequential sampling in the lateral intraparietal region of macaques [18], temporal difference learning in the ventral tegmental area in macaques and other primates [19], and feature matching in the fusiform gyrus of humans and other primates [20]. In machine learning, they include modular addition [21] and monotonic natural language inference [22], both in transformer models. There have also been partial successes, where parts of algorithms have been identified but a complete algorithm has not. For instance, estimates of subjective value, a variable in many decision-making algorithms, have been found in the human ventromedial prefrontal cortex [23]. Similarly, induction heads, a copying mechanism in many text prediction algorithms, have been found in

transformer models [24]. Despite these successes, it has been challenging to identify cognitive algorithms through parts-based methods. Several explanations are possible: current techniques for identifying parts may be inadequate, the parts may be too entangled for any technique to ever disentangle, or there may not be any parts corresponding to cognitive algorithms. Whatever the underlying reason, identifying cognitive algorithms through parts-based methods remains difficult.

A second, behavioral method attributes algorithms based solely on input-output mappings, bypassing the need to identify intermediate parts. This method has been standard in cognitive psychology for decades — researchers often attribute algorithms without neural recordings. Even in cognitive neuroscience, where you might expect neural recordings to be the starting point, some advocate for the primacy of behavioral methods [25, 26]: first use behavior to identify what algorithm a system implements, then look inside to understand the causes of that behavior. It is also widely used in machine learning to study transitive-inference algorithms [27, 28], compositional algorithms [29, 30], and Bayesian algorithms [31]. Across all these domains, its main attraction is that we do not need to search for intermediate parts.

The main challenge for this second method is that the same input-output mapping is almost always consistent with many different algorithms. Researchers address this by taking into account how the system generalizes to new inputs, its pattern of errors, and its reaction times. Nonetheless, a system's input-output mapping is still almost always consistent with a great many different algorithms, and those algorithms can differ in fundamental ways. As a result, attributing algorithms using this method is not always as informative as we would like; algorithms with fundamental differences (e.g., Bayesian inference and heuristics) would be empirically indistinguishable [32], and thus would not inform us about fundamental differences in the networks they describe.

The challenges for parts-based and behavioral methods have led some to regard cognitive algorithms as merely useful fictions about neural systems [33–35]. Their view is that, like Ptolemaic earth-centered models of the universe, these algorithms allow us to reliably predict behavior even though they misdescribe the causes of that behavior. Others conclude that some cognitive algorithms, such as Bayesian algorithms [36], are merely normative, indicating the ideal input-output mapping without attributing any particular operation to neural systems. These views have motivated the use of alternative frameworks, such as dynamical systems theory, to describe processes in neural systems [37, 38]. This skepticism might ultimately be proven correct. But that would be a disappointing endpoint because cognitive algorithms have the potential to provide unique and valuable insights into neural systems. By abstracting away from implementation details, they enable meaningful comparisons across diverse architectures, provide normative benchmarks for evaluation, and provide a framework that makes complex systems comprehensible. Before giving up, we should investigate alternative methods for attributing algorithms to neural systems.

We propose an alternative behavioral method. The intuition underlying our proposal is that a network is faster at learning tasks that require small adjustments to its algorithm, allowing us to look for a match between its transfer learning profile and an algorithm's distances to other algorithms. More precisely, we look for a match between distances in two spaces (in an informal sense). On the one hand, algorithms have distances to other algorithms, defined by how many changes are required to transform one into another. This induces an algorithmic space. On the other hand, for a given network, the distance between its current input-output mapping and other mappings is the amount of training needed to learn them. This induces a transfer learning space. If a network's distances to new mappings in transfer learning space correspond to an algorithm's distances to other algorithms in algorithmic space, we attribute that algorithm to the network.

We assess the viability of this method in a series of experiments that use simple artificial neural networks and tasks [12]. This provides a strong test: if standard parts-based methods can detect algorithms, they should succeed for these systems. We show that transfer learning attributes algorithms when parts-based methods fail, preserves fundamental differences that input-output approaches miss, reliably predicts performance on new tasks, and provides a framework for investigating underlying mechanisms. We conclude that transfer learning offers a promising method for attributing algorithms to neural systems.

Transfer learning has been a standard method in cognitive psychology for decades. For example, Tolman [39] observed that, after mice aimlessly explored a maze, they were faster at learning to navigate between two locations. He concluded that they had learned a map-like representation of the maze during exploration. The insight underlying Tolman's argument is that it takes less time for a subject to learn a new task when some of the representations and algorithms learned on a previous task can be reused, and more time when they cannot. More recently, this insight underlies work on learning to learn in cognitive psychology [40] and transfer learning in machine learning [41, 42]. For example, learning tennis facilitates learning badminton [43], learning one language – whether Spanish or Python – facilitates learning a related language [44–47], and learning individual motor transformations facilitates learning their combination [48]. Similarly, for artificial networks, learning to categorize natural images facilitates learning to categorize medical images [49], learning to sort cards by shape facilitates learning to sort cards by color [50], learning multiple classifications of the same inputs facilitates learning novel classifications [51], and learning to play Space Invaders facilitates learning Frostbite and other Atari games [52].

Previous researchers have suggested connections between transfer learning and algorithms. Davies [53] briefly illustrates how transfer learning might, in principle, reveal the linguistic algorithms underlying semantic processing. More substantially, Maloney and Mamassian [54] propose using transfer learning to identify constituent components of

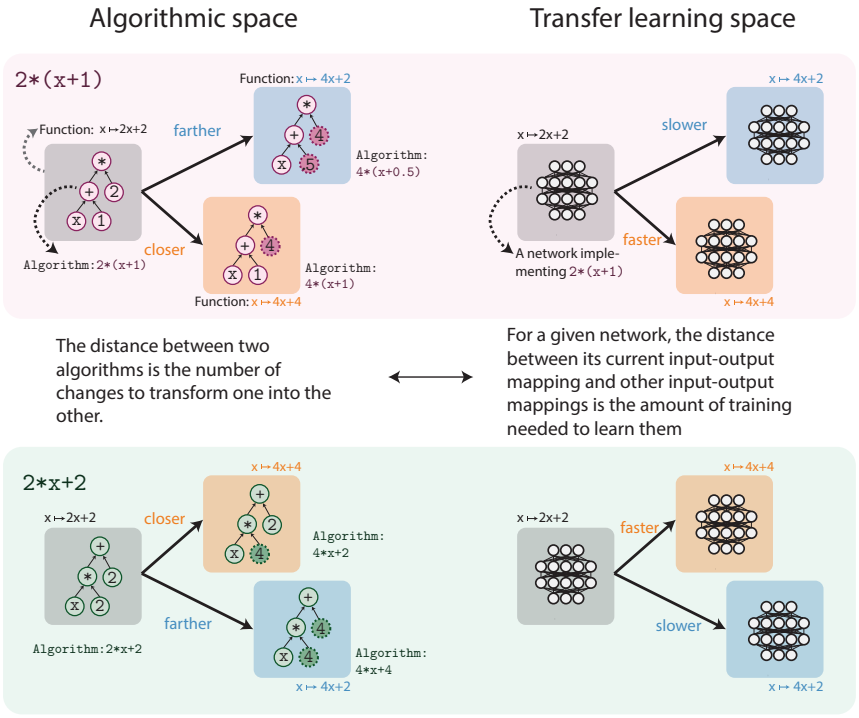

Figure 1: We propose attributing algorithms to neural systems by comparing distances in algorithmic space to distances in transfer learning space. Algorithms (depicted as graphs) and networks (depicted as neural network icons) both implement input-output mappings denoted by gray ($x \mapsto 2x+2$), yellow ($x \mapsto 4x+4$), or blue ($x \mapsto 4x+2$) colored boxes. Top left: The algorithm `2 · (x + 1)` requires one change to reach `4 · (x + 1)` and two changes to reach `4 · (x + .5)`. Changes are highlighted in the graph by darker nodes and broken outlines around the nodes. `2 · (x + 1)` is therefore closer to `4 · (x + 1)` in algorithmic space. This places `2 · (x + 1)` closer to an algorithm generating $x \mapsto 4x+4$ than an algorithm generating $x \mapsto 4x+2$. Turning to transfer learning space (top right): some networks trained on $x \mapsto 2x+2$ are faster at learning $x \mapsto 4x+4$ than $x \mapsto 4x+2$. We attribute `2 · (x + 1)` to those networks because their distances in transfer learning space correspond to the distances of `2 · (x + 1)` in algorithmic space. The algorithm `2 · x + 2` has the opposite pattern (bottom row): it is closer in algorithmic space to an algorithm generating $x \mapsto 4x+2$. We therefore attribute `2 · x + 2` to networks that are faster at learning $x \mapsto 4x+2$.

Bayesian inference in biological and artificial systems (see also [55]). Papadimitriou et al. [56] use transfer learning to detect whether languages share grammatical structure, though without specifying these structures. But so far, no one has used transfer learning to distinguish between competing algorithmic hypotheses about neural systems or compared this approach to standard behavioral and parts-based methods. We do both. Across six experiments in two domains, we demonstrate that transfer learning attributes algorithms even when parts-based and input-output methods do not.

Our method attributes algorithms without identifying parts. This raises questions about how transfer learning and parts-based methods relate to each other, and about what ultimately grounds the attribution of algorithms to neural systems. These questions are as much philosophical as empirical, and we return to them in the Discussion.

In what follows, we apply the transfer learning method to networks trained on three algebraic tasks and three image classification tasks. For each task, the candidate algorithms differ in the number and magnitude of changes required to transition to other algorithms. We attribute algorithms by measuring how quickly networks learn the input-output mappings of other algorithms. We thereby match a network's location in transfer-learning space with an algorithm's location in algorithmic space.

Algebra

First Experiment

Our first experiment involved a linear algebraic task with one variable. We created 100 simple networks with one input node, one output node, and three small hidden layers (number of nodes for each layer: 1-8-4-8-1). The 100 networks were feedforward, fully connected, ReLU, and had different random initial weights.

We used gradient descent (Adam) to minimize their mean squared error on the initial input-output mapping $x \mapsto 2x+2$ for domain $[-50, 50]$. If one of our original networks was unable to learn the task (loss $>$ .05), we reinitialized and retrained it. This guaranteed that all 100 networks learned the initial task.

The initial mapping $x \mapsto 2x+2$ could be generated by two different algorithms. The first algorithm, `2 · x + 2`, doubles the input x and then adds 2. The second algorithm, `2 · (x + 1)`, adds 1 to the input and then doubles the sum. There are other possible algorithms. But we wanted to know: Is the network using an algorithm more like `2 · x + 2` or `2 · (x + 1)` (see Fig. 2A, panel i)?

These algorithms differ in the number and magnitude of changes required to transition to other algorithms. For example, the transition from the algorithm `2 · x + 2` to `4 · x + 2` would require one change while the transition to `4 · x + 4` would require two changes. Thus, we would expect a network using `2 · x + 2` to learn the mapping $x \mapsto 4x+2$ faster than the mapping $x \mapsto 4x+4$. In contrast, the transition from the algorithm `2 · (x + 1)` to `4 · x + 1` would require one change while the transition to `4 · x + .5` would require two changes.

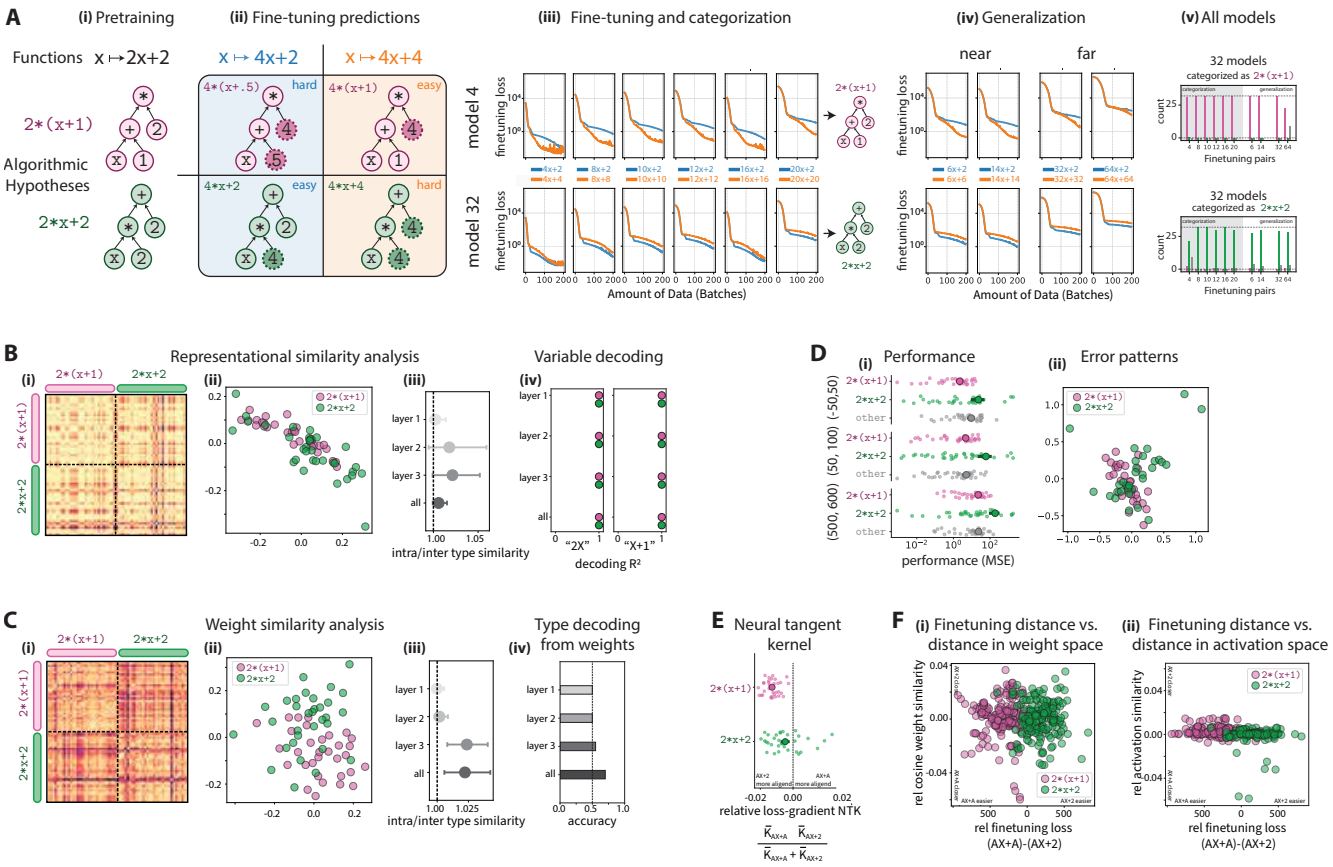

Figure 2: (A) (i) We pretrained simple artificial neural networks on $x \mapsto 2x+2$. We wanted to choose between two algorithmic hypotheses, `2 · x + 2` and `2 · (x + 1)`, here represented as directed graphs. (ii) These algorithms differ in the number and magnitude of changes required to transition to other algorithms. These differences define distances in algorithmic space. The ease with which a network learns a new task should correspond to these distances. (iii) We used the transfer learning speeds of the networks on six pairs of functions to categorize them as `2 · x + 2` or `2 · (x + 1)`. (iv&v) These categorizations predicted transfer learning speed on nearby pairs of functions as well as more distant pairs. (B) These algorithmic categories were not evident from standard analyses of the network's activations. (i) We started by computing representational distances for each network, and then comparing representational distance matrices (RDMs) between networks using representational similarity analysis (RSA). Networks with the same algorithmic category were not more similar to each other than networks from the other category, as shown in (ii) with multidimensional scaling (MDS) and (iii) the similarities ratio of models within vs. across algorithmic categories. (iv) These algorithmic categories were also not apparent from decoding variables of the algorithm from the network's activations (e.g., "2X" or "X+1"). (C) These algorithmic categories were also not evident from analyses of the network's weights. (i) We computed permutation-invariant weight similarity between networks. Networks with the same algorithmic category were not significantly more like each other than networks from the other category, as shown with (ii) MDS and (iii) the similarity ratio within vs across algorithmic categories. (iv) These algorithmic categories were also not apparent from a decoder trained to predict the network's category from its weights. (D) (i) Algorithmic categories were not apparent from standard behavioral analyses, such as performance within domain and to near and far domains. (ii) They were also not apparent from clustering error patterns, as shown in an MDS plot. (E) We could not predict which mapping in each pair a network would learn faster from Neural Tangent Kernel analysis. (F) Nor could we predict it from the magnitude of a network's change during transfer learning, as measured by (i) representational similarity or (ii) weight similarity analyses.

Thus, we would expect a network using `2 · (x + 1)` to learn the mapping $x \mapsto 4x+4$ faster than the mapping $x \mapsto 4x+2$ (see Fig. 2A, panel ii). In light of this contrast, we used a network's learning speed to determine whether it was using an algorithm more like `2 · x + 2` or `2 · (x + 1)`.

After training on the initial mapping, we trained our networks on six pairs of new input-output mappings that, like $x \mapsto 4x+2$ and $4x+4$, allow us to distinguish the two algorithms. In particular, we trained them on the mappings $x \mapsto 4x+2$ and $4x+4$, $8x+2$ and $8x+8$, $10x+2$ and $10x+10$, $12x+2$ and $12x+12$, $16x+2$ and $16x+16$, and $20x+2$ and $20x+20$. For each network, we determined which mapping in each pair it learned faster by comparing its losses (mean squared error) after each batch. If its loss for one mapping was lower than its loss for the other mapping on more than 60% of batches (excluding batches after both losses stabilized), we categorized it as faster at learning that mapping. If a network was faster at learning the mapping predicted by the algorithm `2 · x + 2` for at least five of the six pairs, we categorized it as that algorithm. In contrast, if it was faster at learning the input-output mapping predicted by `2 · (x + 1)` for at least five of the six pairs, we categorized it as that algorithm (see Fig. 2A, panel iii). We consider alternative thresholds and metrics for categorizing networks into algorithms in the appendix.

Our method attributed `2 · x + 2` to 10 of them and `2 ·`

`(x + 1)` to 54 of them. Thus, it attributed algorithms to 64% of the networks. To be clear, attributing `2 · x + 2` to a network does not mean it is using *exactly* that algorithm. It just means that it is using an algorithm more like `2 · x + 2` than `2 · (x + 1)`. To more precisely locate the network in algorithmic space, we would need to consider more algorithms.

The odds of getting these results from random fluctuations in the networks are close to zero. Consider that if each pair of new input-output mappings was a coin flip, the probability of attributing an algorithm to a network would be $2 \times \left(\binom{6}{5} \left(\frac{1}{2}\right)^6 + \binom{6}{6} \left(\frac{1}{2}\right)^6 \right) \approx 21.9\%$ and the probability of attributing an algorithm to at least 64 out of 100 networks would be $\sum_{k=64}^{100} \binom{100}{k} (0.219)^k (0.781)^{100-k}$ and thus $< 10^{-18}$. Note that this calculation assumes that each input-output mapping pair always favors one algorithm or the other. Dropping that assumption would push the probability even lower.

We validated our method by also training our networks on two other held-out pairs of input-output mappings within the range of the other input-output mappings: $6x+2$ and $6x+6$, and $14x+2$ and $14x+14$ (see Fig. 2A, panel iv). Our categorizations predicted performance on these other input-output mappings 73% and 89%, respectively, roughly the same as the input-output mappings used by our categorization. We also trained our networks on two more pairs of input-output mappings outside the range of the other input-output mappings: $32x+2$ and $32x+32$, and $64x+2$ and $64x+64$. Our categorizations predicted performance on these other input-output mappings 68% and 57%, respectively. Thus, our method attributed algorithms that were predictive of learning on further mappings (see Fig. 2A, panel v).

How do the categorizations derived from transfer learning compare to common behavioral approaches, such as within- and out-of-distribution generalization and clustering by error patterns, as well as parts-based approaches like representational similarity analysis, decoding analyses, and weight similarity? To address these questions, we compared models of `2 · x + 2`, `2 · (x + 1)`, and those characterized as 'other' against each other. We subsampled the models to ensure equal representation across categories (n=32 per category).

Let's start with the behavioral approaches. We considered the networks' performance on the original domain, $[-50, 50]$, as well as on two out-of-distribution domains, $[50, 150]$ and $[500, 600]$. We found that, for each domain, performance did not significantly correlate with category. The median performance values were nearly identical, with substantial overlap in distributions (see Fig. 2D, panel i). Moreover, classifiers trained with logistic regression to predict a network's category were barely above chance (52%, 38%, and 40%, respectively, chance level: 33%). Finally, clustering by performance (k-means, k=3) failed to recover our algorithmic categories, yielding adjusted rand indices near zero (.04, .003, and .007). Our inability to distinguish the algorithms by their performance is perhaps unsurprising given that both algorithms make identical predictions for all possible inputs, and thus we wouldn't expect performance to allow us to predict a network's algorithm.

Next, we clustered networks by their error patterns. We were most successful when dividing the inputs into ten bins, averaging the error within each bin, and clustering networks by these ten-dimensional error profiles. However, this produced clusters that bore little relationship to our algorithmic categories (adjusted rand index = .019) (see Fig. 2D, panel ii). We also attempted to decode a network's category from its error patterns using a neural network classifier, but accuracy was not significantly above chance for raw errors (53%, chance: 33%) and only marginally above chance for normalized errors (56%). Even when restricting to the most discriminative input locations (top 5–50, selected using k-NN single-feature classification with k=5), accuracy remained near chance (46–49%, none significantly above chance).

We then turned to parts-based approaches, starting with representational similarity analysis (RSA). For each network and hidden layer, we generated a representational dissimilarity matrix (RDM) using Euclidean distance. We then measured the RDM similarity between networks of the same type (`2 · x + 2` and `2 · (x + 1)`) and between networks of different types (see Fig. 2B, panels i-iii). We considered all of the standard metrics: Euclidean distance, correlational distance, cosine distance, procrustes distance, and centered kernel alignment. Regardless, the similarities within category did not significantly differ from the similarities across categories. Moreover, even if there had been greater similarity within groups than between groups, this approach would not tell us which group was using which algorithm. Clustering networks according to their RDM similarities is a data-driven approach in that it looks for patterns in the data without generating hypotheses about which pattern is associated with which algorithm. It is thus not by itself enough to attribute algorithms.

Another parts-based approach is weight similarity. In a fully connected network, the ordering of weights at a given layer is arbitrary. Therefore, for each pair of networks and each layer, we needed to search for the ordering of weights that maximized their weight similarity at that layer. We did so using the Hungarian combinatorial optimization algorithm. We then compared the similarities within `2 · x + 2` networks, within `2 · (x + 1)` networks, and between `2 · x + 2` and `2 · (x + 1)` networks (see Fig. 2C, panels i-ii). The similarities within each category differed only slightly from the similarities across categories (see Fig. 2C, panel iii), and that difference was too small to reliably predict whether a network was categorized `2 · x + 2` or `2 · (x + 1)`. Moreover, just as with similarities in activity, even if there had been a correlation, that would not have been enough to attribute an algorithm because we would have been left with two uninterpreted groupings of networks. Clustering networks according to their weight similarity is also a data-driven approach, and thus not by itself enough to attribute algorithms.

We also tried to decode whether a network was `2 · x + 2` or `2 · (x + 1)` from its weights. We used logistic regression decoders with inputs from individual layers as well as across all

layers. The best individual-layer decoder (layer 3) achieved 59% test accuracy (baseline: 33%). The best all-layer decoder achieved only 34% test accuracy, suggesting that the larger number of parameters led to overfitting (see Fig. 2C, panel iv). It's possible that decoder performance would improve with more networks to train on. But, regardless, this establishes that the difference in learning speeds is not easily discoverable through an inspection of the networks' weights.

A different decoding approach is to search for the algorithm's variables. Unlike data-driven approaches, this approach is hypothesis-driven: each candidate algorithm specifies which variables to look for. We found that both of the relevant variables ($x+1$, $2x$) could be perfectly decoded from all three layers using a linear decoder (see Fig. 2B, panel iv). This shouldn't be surprising because these variables are linear transformations of both the input and output. This illustrates why decoding by itself is not enough to attribute algorithms, at least on linear tasks. We will later consider decoding on non-linear tasks.

These results demonstrate that our transfer learning approach discovers differences between networks that are not easily discoverable through an investigation of their performance, activations, and weights.

Does a network's learning speed simply reflect how much it changes during transfer learning? We measured change in two ways: cumulatively (summing changes across all training steps) and overall (comparing the network before and after transfer learning). Neither predicted which function in each pair a network would learn faster. Cumulative weight change correctly predicted the slower function in only 56.4% of pairs (chance: 50%). For overall change, our best results were 65% accuracy using representational similarity (RSA metric: correlation, RDM metric: correlation, layer three activations) and 60% using weight similarity (weight metric: correlation, output weights). Even with more conservative algorithm attributions, these accuracies improved only marginally (to 66% and 62%, respectively). Learning speed does not simply reflect how much a network changes during transfer learning, suggesting that gradient descent does not take a direct path in weight or representational space (see Fig. 2F, panels i-ii).

Finally, we tested whether transfer learning speed directly reflects the alignment of the trained network's gradient structure with the new input-output mapping. If the network's gradients better align with one input-output mapping rather than another, gradient descent may learn the former function much faster. We use Neural Tangent Kernel (NTK) to measure gradient alignment. We computed the NTK of a trained network for each transfer function (e.g., 8x+2 or 8x+8) and averaged over the input space to quantify how well the neural network's gradients align with that function under linearized training dynamics. We found that in all but one case this analysis predicted that a network would learn $a \cdot b + a$ faster than $a \cdot b + 2$, and thus its performance at identifying the algorithm was barely above baseline (see Fig. 2E).

Combined, these results establish that transfer learning reveals algorithmic differences beyond the reach of standard parts-based and behavioral methods.

Second Experiment

To test whether our results generalize beyond single-variable tasks, our second experiment involved two variables. We trained 100 simple networks with the same architecture on the mapping $x, y \mapsto -x - y$ for domain $[-50, 50] \times [-50, 50]$. This mapping could be generated by two different algorithms: `-x - y` and `-(x + y)`. Our method attributed `-x - y` to 32 networks and `-(x + y)` to 23 networks. Once again, transfer learning revealed algorithmic differences beyond the reach of standard methods (see the appendix for results).

Third Experiment

To test whether our results extend to nonlinear functions, our third experiment involved a nonlinear mapping. We trained 150 simple networks with the same architecture on the mapping $x, y \mapsto (-x - y)\tanh xy$ for domain $[-50, 50] \times [-50, 50]$. $\tanh(xy)$ is positive in the first and third quadrants, and negative in the second and fourth quadrants, creating a saddle-like surface, essentially a continuous version of XOR. Our method attributed `(-x - y)tanh(xy)` to 101 networks and `-(x + y)tanh(xy)` to 26 networks. Standard methods again failed to detect these distinctions (see the appendix for results).

Categorization

Fourth Experiment

To test whether these results extend beyond algebraic tasks, our fourth experiment involved shape categorization. Identifying which kind of algorithm a neural system is using is crucial for understanding how it categorizes images, yet standard methods struggle to do so. Unlike our previous experiments, which distinguished algorithms by their order of operations, this experiment distinguished algorithms by how they detect features. One kind of algorithm jointly detects features without distinguishing them. Another kind of algorithm separately detects each feature.

We created 100,000 64×64 monochromatic images using five latent variables: `shape` (square, circle, ellipse, capsule), `color` (object's color, from white to black), `background` (background color, from white to black), `area` (from 160 to 3217 pixels), and `orientation` (0 to 180 degrees). Networks were trained to classify images based on whether

$$\texttt{shape} \cdot \Big[\texttt{color} + \texttt{background} + \texttt{area}\Big] > 0 \qquad (1)$$

where two shapes were assigned 1, the other two shapes were assigned -1, and all other variables were scaled between -1 and 1. To learn this task, networks need to recover the values of `shape`, `color`, `background`, and `area` from each image (see Fig. 3A, panel i).

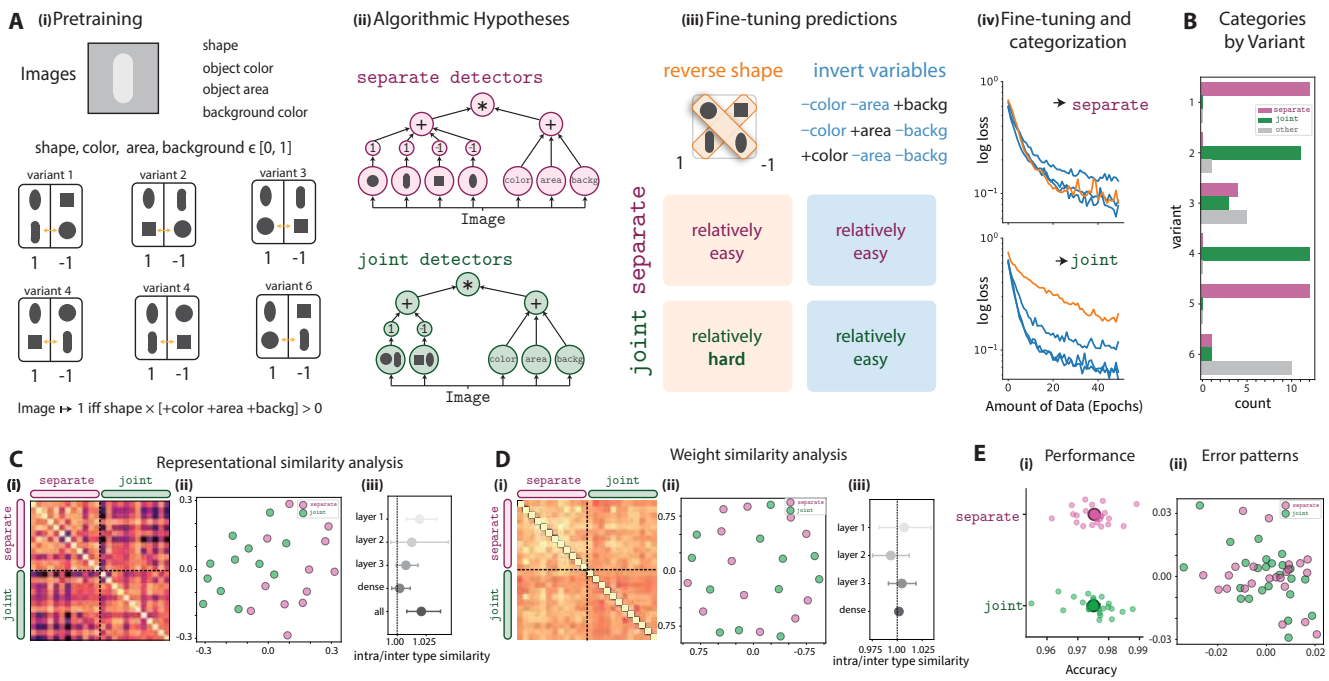

Figure 3: (A) (i) We pretrained CNNs on images generated from five latent variables: shape (square, circle, ellipse, capsule), object color, background color, object area, and orientation (which wasn't used in the classification rule). Latent variables were assigned values between -1 and 1. For shape, two shapes were assigned +1 and two were assigned -1. There were six variants corresponding to the six different ways of grouping shapes for training (the groupings shown) and transfer learning (regroupings indicated by orange arrows). Images were classified as 1 or 0 based on the rule shown at the bottom of the panel. (ii) We considered two algorithmic hypotheses for assigning values to shapes: separately detect all four shapes, or jointly detect shapes assigned the same value. (iii) These algorithms make different predictions about transfer learning difficulty. Joint detectors should find shape recombination (the orange arrows) relatively hard, requiring disentanglement of previously grouped shapes, but variable inversion relatively easy. Separate detectors should find both tasks similarly easy. (iv) We used these asymmetries to categorize networks: joint detector networks were slower at learning new shape groupings than variable inversions, while separate detector networks showed similar speeds on both. (B) The algorithm a network learned correlated with its training variant: similar shapes grouped together (e.g., ellipse and circle) predominantly yielded joint detectors, while dissimilar shapes yielded separate detectors. (C) Algorithmic categories were not evident from representational similarity analysis. (i) RDM similarities between networks, (ii) visualized with MDS, the algorithm types appear linearly separable, but which cluster corresponds to which algorithm is only apparent after categorization using transfer learning. (iii) Intra-category similarity did not significantly exceed inter-category similarity across layers. (D) Algorithmic categories were also not evident from weight similarity analysis, as shown by (i) the similarity matrix, (ii) MDS visualization, and (iii) intra/inter similarity ratios. (E) Standard behavioral analyses also failed to reveal algorithmic differences: (i) performance accuracy and (ii) error pattern clustering did not distinguish joint from separate detector networks.

For a given variant, there are at least two algorithms that a network could use to estimate the value of `shape` for each image. Applying the distinction above, a network could jointly detect two shapes without distinguishing between them, or it could separately detect each shape (see Fig. 3A, panel ii). To identify which algorithm each network used, we trained them on two new types of labels: labels that recombined shapes into different groups, and labels that inverted two of the other latent variables. For networks using joint detectors, shape recombination would require a large algorithmic change because they would need to disentangle previously entangled shapes. Variable inversions would require a smaller algorithmic change because the initial task already forced them to distinguish `color`, `background`, and `area`. For networks using separate detectors, both shape recombination and variable inversions would require similar algorithmic changes – all of the shapes and variables were already distinguished. For this reason, joint-detector networks should be significantly slower at learning labels that recombine shapes than labels that invert variables, while separate-detector networks should show similar learning speeds (see Fig. 3A, panel iii).

To induce networks to learn different algorithms, we trained CNNs on six different groupings of shapes. We created them by varying which shapes were grouped. For example, in one variant, ellipses and circles were assigned 1, with squares and capsules therefore assigned -1, while in another variant, ellipses and squares were assigned 1, with circles and capsules therefore assigned -1.

We trained 10 randomly initialized CNNs on all six versions of the experiment, generating a total of 60 trained networks. Each network had three convolutional layers (16, 32, and 64 filters, all 3×3 with 2×2 max pooling), followed by a 128-node dense layer and sigmoid output. Networks were then trained on the following new tasks, where (2) uses an alternative shape grouping (see Fig. 3) and (3)-(5) invert two of the other variables:

$$\texttt{altshape} \cdot \Big[\texttt{color} + \texttt{background} + \texttt{area}\Big] > 0 \quad (2)$$

$$\texttt{shape} \cdot \Big[-\texttt{color} - \texttt{background} + \texttt{area}\Big] > 0 \quad (3)$$

$$\texttt{shape} \cdot \Big[-\texttt{color} + \texttt{background} - \texttt{area}\Big] > 0 \quad (4)$$

$$\texttt{shape} \cdot \Big[\texttt{color} - \texttt{background} - \texttt{area}\Big] > 0 \quad (5)$$

Networks that were significantly slower at learning new shape groupings (more than one standard deviation above their maximum speed for variable inversions) were categorized as using joint detectors. Networks with similar speeds for both were categorized as using separate detectors. The remaining networks were classified as other (see Fig. 3A, panel iv).

Our method attributed joint detectors to 27 networks and separate detectors to 25, with 8 networks classified as other. Interestingly, a network's algorithm correlated with its training variant: when two similar shapes (e.g., ellipse and circle) were assigned the same value, networks predominantly developed joint detectors, and when dissimilar shapes (e.g., square and circle) were assigned the same value, networks predominantly developed separate detectors. Only two variants yielded both joint and separate detector networks (see Fig. 3B).

We found that standard methods could not detect the algorithmic differences identified by transfer learning (see Fig. 3C–E). In order to control for any differences in initial training conditions, we analyzed networks from one of the variants that produced models implementing both algorithms, namely variant 3.

Fifth Experiment

To test whether these results extend to three-dimensional categorization, we ran a similar experiment using Google's 3DShapes database. We trained the networks on labels generated by a similar combination of the latent variables: $\texttt{shape} \cdot \Big[\texttt{color} + \texttt{wall} + \texttt{scale}\Big]$ where `wall` is wall color. Our method attributed joint detectors to 8 networks and separate detectors to 43 networks. Once again, transfer learning revealed algorithmic differences beyond the reach of standard methods (see the appendix for results).

Sixth Experiment

To test whether these results extend to labels generated by a non-linear function, we then trained the networks on labels generated by a different combination of latent variables: $\texttt{shape} \cdot \Big[(\texttt{color} \cdot \texttt{wall}) + (\texttt{scale} \cdot \texttt{floor})\Big] > 0$. Our method attributed joint detectors to 19 networks and separate detectors to 38 networks. Standard methods again failed to detect these distinctions (see the appendix for results).

Discussion

Across six experiments, we used the speed at which a network learns new tasks to identify its algorithm by matching distances in transfer learning space with distances in algorithmic space. We also compared our method to behavioral methods which attribute algorithms based only on a network's input-output mapping, and parts-based methods which attribute algorithms by looking for the parts of an algorithm in a network. We found that the transfer learning method discovers meaningful distinctions between networks that were not apparent from these other approaches. In particular, our method attributes algorithms to networks in a way that is not predicted by within- or out-of-distribution generalization performance, error patterns, representational similarity of layer activations, similarity of the networks' weights, or by the ability to decode algorithmic parts from either activations or weights.

Why does transfer learning attribute algorithms to these networks when parts-based methods do not? One answer is that standard parts-based methods are not sensitive enough, but new methods might be. We could try to develop more sensitive methods by working backwards from the algorithms attributed by transfer learning. For example, we demonstrated that standard metrics for activity and weight similarity, such as cosine similarity, do not reveal the same algorithmic categories as transfer learning. Non-standard metrics might, and we can try to identify them by searching for metrics that sort networks into the same categories as transfer learning. If we succeed, parts-based methods could provide the ground truth about a network's algorithm, and that ground truth might vindicate transfer learning.

Another answer is that the two methods provide different kinds of evidence about a network's algorithm. Transfer learning constrains which algorithms are consistent with a network's learning dynamics, while parts-based methods constrain which algorithms are consistent with its internal structure. When the constraints from both methods converge on the same algorithm, they confirm each other. When one method fails to yield an answer — for example, when parts-based methods do not find corresponding groupings, as in our experiments — the broader pattern of convergence across cases gives us reason to trust the other. Unlike in the first answer, parts-based methods are not taken to provide the ground truth about a network's algorithm. Rather, the ground truth is determined by the combination of both constraints.

Either way, transfer learning has a lot to offer. To start, even when parts-based methods suggest that networks share the same algorithm, they often cannot tell us which algorithm. Transfer learning can provide that interpretation. Transfer learning can also make our parts-based search for algorithms more efficient: by narrowing down candidate algorithms, transfer learning reduces the search space that parts-based methods must explore. And in cases where parts-based methods fail to yield an answer, transfer learning can sometimes attribute algorithms on its own.

Both answers are appealing, but neither is guaranteed to

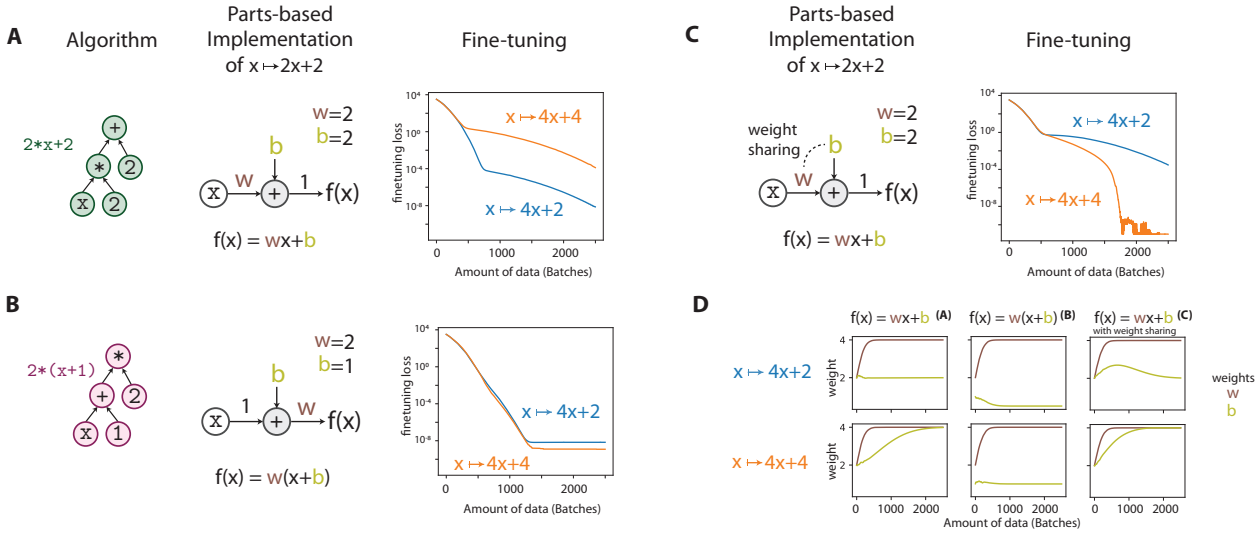

Figure 4: Small, handcrafted networks illustrate how parts-based and transfer learning approaches can both converge and diverge. (A) By construction, $w = 2$ and $b = 2$. As a result, this network implements `2 · x + 2` in the parts-based sense. The transfer learning approach attributes `2 · x + 2` because the network learns $4x + 2$ faster than $4x + 4$. (B) By construction, $w = 2$ and $b = 1$. As a result, this network implements `2 · (x + 1)` in the parts-based sense. The transfer learning approach attributes `2 · (x + 1)` because the network learns $4x + 4$ faster than $4x + 2$ during transfer learning. (C) The two approaches diverge under soft weight sharing (gradient descent with a penalty on divergent weights). When we train the network from (A, top) with soft weight sharing, it learns $4x + 4$ faster than $4x + 2$, and thus implements `2 · (x + 1)` in the transfer learning sense even though it still implements `2 · x + 2` in the parts-based sense. (D) Learning dynamics of weights w and b for the three examples. Because the network is so simple, changes in the weights are easy to interpret. For networks that are even slightly more complex, there is no obvious interpretation.

succeed. In complex networks, algorithmic parts may be impossible to identify — or may not exist at all. If so, we could neither vindicate transfer learning with more sensitive parts-based methods, nor expect the two methods to converge. The fact that our transfer learning attributions generalize to held-out mappings indicates that they are not mere noise. But without independent support from parts-based methods, we could not rule out the possibility that they are systematic byproducts of training rather than reflections of an underlying algorithm. This would push us towards the conclusion that cognitive algorithms are merely useful fictions [33–35].

But there is a way to avoid this conclusion. A more radical, revisionary answer is that transfer learning provides an alternative ground truth about a network's algorithm. Just as Shannon and Fisher information provide different but complementary measures of the information in a system, transfer learning and parts-based methods may provide different but complementary senses of what it is for a network to implement an algorithm. In one sense, a network implements an algorithm in virtue of having parts that correspond to algorithmic steps. In another sense, it implements an algorithm in virtue of its transfer learning profile. A network could implement an algorithm in the second sense even if it does not implement an algorithm in the parts-based sense.

This answer would have deep consequences, forcing us to rethink what it is for neural systems to implement algorithms, and therefore how cognitive psychology relates to systems neuroscience and machine learning. One might worry that it abandons mechanistic explanation. But we could still investigate the mechanisms behind a network's transfer learning profile; we just would not require those mechanisms to have parts corresponding to algorithmic parts. And where parts-based methods do succeed, a network would still implement an algorithm in that sense. There would simply be more than one sense in which a network can implement an algorithm, and thus multiple ground truths. Given that current parts-based methods do not attribute algorithms to many networks, including the networks in our experiments, the transfer learning sense may often be our only option.

These three answers are sharply distinguished by small, handcrafted networks, such as the network in Figure 4. We used transfer learning speed to attribute algorithms to networks with known parts-based implementation. With a standard learning rule method, transfer learning and parts-based agreed on the algorithm. Transfer learning and parts-based interpretations may however diverge at times. For example, when we introduced a non-standard learning rule (soft weight sharing, Fig. 4C), transfer learning attributed `2 · (x + 1)` to the network, while the conventional parts-based interpretation is `2 · x + 2`. How should we think about this disagreement? On the first answer, we should attribute the parts-based interpretation (`2 · x + 2`) regardless of its learning rule, and conclude that transfer learning attributes the wrong algorithm for the second learning rule. We might still hold out hope that transfer learning is reliable in general; perhaps its error in this case is due to the network's unusual size or its unusual learning rule. Rules that penalize a network for divergent weights could be dismissed as edge cases because standard learning rules do not push the network's weights closer together. Alternatively, one could take the learning rule into account when

identifying a network's parts so that these weights no longer count as "separate" parts. On the second, convergent answer, the divergence between the two methods would leave the network's algorithm for the second learning rule undetermined. On the third, revisionary answer, there is a parts-based sense in which the network implements `2·x + 2` under both learning rules, but also a transfer-learning sense in which it implements `2·x + 2` under the first learning rule and `2·(x + 1)` under the second.

Whichever answer one adopts, there are practical limitations to the transfer learning approach. One is that, if a network cannot learn, we cannot use it to attribute an algorithm. For example, if the biological network responsible for low-level vision is fixed, we might need to rely on parts-based or input-output methods instead. Another limitation is that, even if a network can learn, practical constraints may make transfer learning infeasible. For example, the inner workings of the largest language models might require too much training data for this approach to be tractable. Biological networks present additional challenges because transfer learning in a deeply embedded network might require invasive stimulation and recording techniques. In such cases, artificial network models may serve as our best proxy: we can apply transfer learning to them to potentially infer which algorithms the biological networks are using. A third limitation is that transfer learning attributes a class of algorithms, not a single algorithm. For example, as we noted earlier, attributing `2·x + 2` to a network does not mean that it is using `2·x + 2` exactly. There are many algorithms more similar to `2·x + 2` than `2·(x + 1)`, and attributing `2·x + 2` only means that the network is using an algorithm in that class. Additional transfer learning experiments can narrow down the class, but they are unlikely to converge on a single algorithm in finitely many steps.

The transfer learning approach relies on the intuition that algorithms differ in how easily they can be transformed into one another. We operationalized this intuition by counting the number of changes required. We thereby created an informal algorithmic space in which some algorithms are closer than others. A more systematic approach would represent each algorithm as a computation graph and define the distances between them via a graph edit distance. This metric can account for the type of change (modifying a coefficient versus a constant, or substituting one operation for another) as well as the location of the change within the computational hierarchy (whether it occurs near the inputs or near the outputs). Different metrics define different algorithmic spaces. By fitting the costs of different edit types to transfer learning data from multiple experiments, we could derive a domain-specific distance metric. This metric would capture which algorithmic changes are easy or difficult to learn within a domain. If successful, we could use it to make predictions about how quickly networks would learn novel functions. Developing such metrics remains an important direction for future work.

A related limitation is that our experiments used small networks with simple architectures. Demonstrating that the method scales to larger and more complex networks remains another important direction for future work. Our hope is that, just as the study of deep linear networks has already yielded insights into training dynamics that generalize to nonlinear settings [12], the study of simple networks will yield insights into algorithms that generalize to more complex networks.

Conclusion

Cognitive algorithms offer simple perspectives on complex neural systems. Across algebraic and categorization tasks, we identified algorithms by how networks learned new tasks rather than by their weights, activations, or behavior on the original task. We believe that this is a productive framework for integrating cognitive psychology with systems neuroscience and machine learning.

References

[1] G Frege, *Grundgesetze der Arithmetik*. (Verlag Hermann Pohle, Jena) Vol. I–II, (1893/1903).

[2] J von Neumann, First draft of a report on the EDVAC, (Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania), Technical report (1945).

[3] CE Shannon, Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **41**, 256–275 (1950).

[4] Pāṇini, *Aṣṭādhyāyī of Pāṇini*. (University of Texas Press, Austin), (1987).

[5] JI Marcum, A statistical theory of target detection by pulsed radar, (RAND Corporation, Santa Monica, CA), Technical Report RM-754-PR (1947).

[6] T Bayes, An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London* **53**, 370–418 (1763).

[7] CJCH Watkins, P Dayan, *q*-learning. *Machine Learning* **8**, 279–292 (1992).

[8] M Chun, S Most, *Cognition*. (Oxford University Press), (2022).

[9] JV Haxby, AC Connolly, JS Guntupalli, Decoding neural representational spaces using multivariate pattern analysis. *Annual Review of Neuroscience* **37**, 435–456 (2014).

[10] N Kriegeskorte, M Mur, P Bandettini, Representational similarity analysis — connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience* **2**, 4 (2008).

[11] T Naselaris, KN Kay, S Nishimoto, JL Gallant, Encoding and decoding in fMRI. *Neuroimage* **56**, 400–410 (2011).

[12] AM Saxe, JL McClelland, S Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks in *International Conference on Learning Representations (ICLR)*. (2014).

[13] SK Ainsworth, J Hayase, S Srinivasa, Git rebasin: Merging models modulo permutation symmetries (2023) arXiv:2209.04836.

[14] SS Kim, H Rouault, S Druckmann, V Jayaraman, Ring attractor dynamics in the drosophila central brain. *Science* **356**, 849–853 (2017).

[15] G Ashida, CE Carr, Sound localization: Jeffress and beyond. *Current Opinion in Neurobiology* **21**, 745–751 (2011).

[16] S Grillner, P Wallén, Central pattern generators for locomotion, with special reference to vertebrates. *Annual Review of Neuroscience* **8**, 233–261 (1985).

[17] DA Robinson, Integrating with neurons. *Annual Review of Neuroscience* **12**, 33–45 (1989).

[18] MN Shadlen, ER Kandel, Decision-making and consciousness in *Principles of Neural Science*, eds. ER Kandel, JD Koester, SH Mack, SA Siegelbaum. (McGraw Hill), 6 edition, pp. 1392–1420 (2021).

[19] W Schultz, Predictive reward signal of dopamine neurons. *Journal of Neurophysiology* **80**, 1–27 (1998).

[20] L Chang, DY Tsao, The code for facial identity in the primate brain. *Cell* **169**, 1013–1028 (2017).

[21] N Nanda, L Chan, T Lieberum, J Smith, J Steinhardt, Progress measures for grokking via mechanistic interpretability (2023) arXiv:2301.05217.

[22] A Geiger, Z Wu, C Potts, T Icard, ND Goodman, Finding alignments between interpretable causal variables and distributed neural representations (2023) arXiv:2303.02536.

[23] JW Kable, PW Glimcher, The neural correlates of subjective value during intertemporal choice. *Nature Neuroscience* **10**, 1625–1633 (2007).

[24] C Olsson, et al., In-context learning and induction heads (2022) arXiv:2209.11895.

[25] JW Krakauer, AA Ghazanfar, A Gomez-Marin, MA MacIver, D Poeppel, Neuroscience needs behavior: Correcting a reductionist bias. *Neuron* **93**, 480–490 (2017).

[26] Y Niv, The primacy of behavioral research for understanding the brain. *Behavioral Neuroscience* **135**, 601–609 (2021).

[27] K Kay, et al., Emergent neural dynamics and geometry for generalization in a transitive inference task. *PLOS Computational Biology* **20**, e1011954 (2024).

[28] S Lippl, K Kay, G Jensen, VP Ferrera, L Abbott, A mathematical theory of relational generalization in transitive inference. *Proceedings of the National Academy of Sciences* **121**, e2314511121 (2024).

[29] D Hupkes, V Dankers, M Mul, E Bruni, Compositionality decomposed: How do neural networks generalise? (2020) arXiv:1908.08351.

[30] BM Lake, M Baroni, Human-like systematic generalization through a meta-learning neural network. *Nature* **623**, 115–121 (2023).

[31] AE Orhan, WJ Ma, Efficient probabilistic inference in generic neural networks trained with non-probabilistic feedback. *Nature Communications* **8**, 138 (2017).

[32] G Gigerenzer, Why heuristics work. *Perspectives on Psychological Science* **3**, 20–29 (2008).

[33] PS Churchland, *Neurophilosophy: Toward a Unified Science of the Mind-Brain*. (The MIT Press), (1986).

[34] PM Churchland, *A Neurocomputational Perspective: The Nature of Mind and the Structure of Science*. (The MIT Press), (1989).

[35] WM Ramsey, *Representation Reconsidered*. (Cambridge University Press), (2007).

[36] TL Griffiths, N Chater, C Kemp, A Perfors, JB Tenenbaum, Probabilistic models of cognition: exploring representations and inductive biases. *Trends in Cognitive Sciences* **14**, 357–364 (2010).

[37] TP Lillicrap, KP Kording, What does it mean to understand a neural network? (2019) arXiv:1907.06374.

[38] GW Lindsay, D Bau, Testing methods of neural systems understanding. *Cognitive Systems Research* **82**, 101156 (2023).

[39] EC Tolman, Cognitive maps in rats and men. *Psychological Review* **55**, 189–208 (1948).

[40] SM Cormier, JD Hagman, *Transfer of Learning: Contemporary Research and Applications*. (Academic Press, San Diego), (1987).

[41] S Thrun, L Pratt, *Learning to Learn*. (Springer, Boston, MA), (1998).

[42] S Ruder, An overview of multi-task learning in deep neural networks (2017) arXiv:1706.05098.

[43] AM Williams, P Ward, JM Knowles, NJ Smeeton, Anticipation skill in a real-world task: Measurement, training, and transfer in tennis. *Journal of Experimental Psychology: Applied* **8**, 259–270 (2002).

[44] T Odlin, *Language Transfer: Cross-Linguistic Influence in Language Learning*, Cambridge Applied Linguistics. (Cambridge University Press), (1989).

[45] H Ringbom, *The Role of the First Language in Foreign Language Learning*. (Multilingual Matters), (1987).

[46] J Scholtz, S Wiedenbeck, Using unfamiliar programming languages: Effects on expertise. *Interacting with Computers* **5**, 13–30 (1993).

[47] A Robins, J Rountree, N Rountree, Learning and teaching programming: A review and discussion. *Computer Science Education* **13**, 137–172 (2003).

[48] JR Flanagan, et al., Composition and decomposition of internal models in motor learning under altered kinematic and dynamic environments. *Journal of Neuroscience* **19**, RC34 (1999).

[49] P Rajpurkar, et al., Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning (2017) arXiv:1711.05225.

[50] B Tsuda, KM Tye, HT Siegelmann, TJ Sejnowski, A modeling framework for adaptive lifelong learning with transfer and savings through gating in the prefrontal cortex. *Proceedings of the National Academy of Sciences* **117**, 29872–29882 (2020).

[51] WJ Johnston, S Fusi, Abstract representations emerge naturally in neural networks trained to perform multiple tasks. *Nature Communications* **14**, 1040 (2023).

[52] E Parisotto, JL Ba, R Salakhutdinov, Actor-mimic: Deep multitask and transfer reinforcement learning in *International Conference on Learning Representations (ICLR)*. (2016) arXiv:1511.06342.

[53] M Davies, Tacit knowledge and semantic theory: Can a five percent difference matter? *Mind* **96**, 441–462 (1987).

[54] L Maloney, P Mamassian, Bayesian decision theory as a model of human visual perception: Testing Bayesian transfer. *Visual Neuroscience* **26**, 147–155 (2009).

[55] Á Koblinger, J Fiser, M Lengyel, Representations of uncertainty: Where art thou? *Current Opinion in Behavioral Sciences* **38**, 150–162 (2021).

[56] I Papadimitriou, D Jurafsky, Learning music helps you read: Using transfer to study linguistic structure in language models (2020) arXiv:2004.14601.

[57] C Burgess, H Kim, 3d shapes dataset (https://github.com/deepmind/3dshapes-dataset/) (2018) DeepMind, GitHub repository.

Appendix

First Experiment

We categorized a network as faster at learning one new mapping rather than another if its loss was lower than the other on 60% of batches. This isn't the only approach. We explored summing difference between the two loss curves. However, that sum was typically dominated by the losses on the first few batches. As a result, it typically did not reflect the overall shapes of the loss functions. We also explored using the number of batches until one of the loss functions crossed a given threshold. However, it was often unclear which threshold to choose because the loss for different functions (e.g., 4x+2 and 32x+2) and different networks converged to different values. We concluded that our method for assigning algorithms to networks was the simplest and most robust.

Our method uses two hyperparameters: a threshold for the percentage of batches one loss curve needed to be below the other (60%), and the number of finetuning pairs required to vote in favor of a particular algorithm (5 out of 6). We tested the sensitivity of our results to these hyperparameter settings.

Threshold	Majority ≥ 4	Majority ≥ 5	Majority $= 6$
0.50	82/15/2	76/13/10	75/9/16
0.55	78/12/9	73/11/15	68/5/27
0.60	72/10/17	69/8/22	63/3/34
0.65	69/10/20	60/7/32	55/2/43
0.70	60/8/31	52/5/42	42/2/56

Table 1: Sensitivity analysis for the first experiment. Each cell shows the number of networks categorized as `2·x + 2` / `2·(x + 1)` / other.

Critically, we also assessed whether the choice of hyperparameters impacted generalization to the held-out pairs of new functions. Across three hyperparameter choices, generalization accuracy remained well above chance (33%) and with similar patterns: highest accuracy for within-range pairs and a consistent decline for the most extreme out-of-range pair. With the most conservative hyperparameters (70% and 6/6 networks), generalization accuracy was 64–67% for within-range pairs and 59% for out-of-range pairs. With the most liberal hyperparameters (50% and 4/6 networks), accuracy was 80–89% for within-range pairs and 71–80% for out-of-range pairs. Thus, the overall pattern of generalization was stable across hyperparameter choices. More liberal hyperparameters yield higher generalization accuracy because they categorize fewer networks as "other" and a network categorized as "other" might nonetheless have most pairs of training functions favoring one algorithm, a lost opportunity to generalize to that algorithm on held-out pairs. By categorizing fewer networks as "other," these hyperparameters result in fewer mismatches. Our chosen hyperparameter settings (60% and 5/6 networks) balance generalization accuracy against the risk of miscategorizing networks with ambiguous transfer learning profiles.

Second Experiment

Our second experiment involved a linear algebraic task with *two* variables. The procedure and results were similar to the first experiment.

We created 100 simple networks with an architecture similar to the first experiment (2-8-4-8-1) and random initial weights. We used gradient descent (Adam) to minimize their mean squared error on the initial input-output mapping $(x,y) \mapsto -x-y$ for domain $[-50,50] \times [-50,50]$. If one of the networks was unable to learn the task (loss $>$ .05), we reinitialized and retrained it. This guaranteed that all 100 networks learned the initial task.

The initial mapping $(x,y) \mapsto -x-y$ could be generated by two different algorithms: `-x - y` and `-(x + y)`. We wanted to know: Is the network using an algorithm more like `-x - y` or `-(x + y)`? We would expect a network using `-x - y` to learn the mapping $(x,y) \mapsto -x-4y$ faster than the mapping $(x,y) \mapsto -4x-4y$, and a network using `-(x + y)` to learn the mapping $(x,y) \mapsto -4x-4y$ faster than the mapping $(x,y) \mapsto -x-4y$. In light of this contrast, we used a network's learning speed to determine whether it was using an algorithm more like `-x - y` or `-(x + y)`.

We trained our networks on six pairs of new input-output mappings: $(x,y) \mapsto -x-4y$ and $-4x-4y$, $-x-8y$ and $-8x-8y$, $-x-10y$ and $-10x-10y$, $-x-12y$ and $-12x-12y$, $-x-16y$ and $-16x-16y$, and $-x-20y$ and $-20x-20y$. Our method attributed $-x-y$ to 22 networks and $-(x+y)$ to 32 networks. Thus, it attributed algorithms to 78% of the networks.

We tested the sensitivity of our results to our hyperparameter settings:

Threshold	Majority ≥ 4	Majority ≥ 5	Majority $= 6$
0.50	36/51/13	31/40/29	21/28/51
0.55	33/46/21	27/36/37	16/23/61
0.60	29/42/29	23/32/45	10/21/69
0.65	26/36/38	16/30/54	8/18/74
0.70	21/31/48	12/29/59	8/15/77

Table 2: Sensitivity analysis for the second experiment. Each cell shows the number of networks categorized as `-x - y` / `-(x + y)` / other.

We validated our method using two held-out pairs of input-output mappings within the range of the other new input-output mappings: $-x-6y$ and $-6x-6y$, and $-x-14y$ and $-14x-14y$. Our categorizations predicted performance on these other input-output mappings 79% and 79%, respectively, roughly the same as the input-output mappings used by our categorization. We also trained our networks on two more pairs of input-output mappings outside the range of the other mappings: $-x-32y$ and $-32x-32y$, and $-x-64y$ and $-64x-64y$. Our categorizations predicted performance on these other input-output mappings 62% and 45%, respectively.

As we now show, the categorizations were not apparent

from standard behavioral or parts-based approaches. Like in the first experiment, our analyses subsampled the models from a larger run to ensure equal number of networks across categories in our analyses (n=32 each for `-x - y`, `-(x + y)`, and 'other').

Generalization: We considered the networks' performance on the original domain, $[-50,50] \times [-50,50]$, as well as on two out-of-distribution domains, $[50,150] \times [50,150]$ and $[500,600] \times [500,600]$. We found that, for each domain, performance did not significantly correlate with category (all Pearson and Spearman correlations non-significant, $p > .05$). The median performance values were nearly identical, with substantial overlap in distributions. Moreover, classifiers trained with logistic regression to predict a network's category were at or below chance level (28%, 28%, and 31% for the three domains, respectively; chance: 33%). Finally, clustering by performance (k-means, k=3) failed to recover our algorithmic categories (adjusted rand indices ≈ 0 for all three domains).

Error pattern analysis: We divided the inputs into bins, averaged the error within each bin, and clustered networks by these error profiles. However, this produced clusters that bore little relationship to our algorithmic categories (adjusted rand index ≈ 0 across bin sizes 4, 6, and 10). We also attempted to decode a network's category from its error patterns using a neural network classifier, but accuracy was not significantly above chance (35%, chance: 33%). Even when restricting to the most discriminative input locations (top 5–50 features, selected using k-NN single-feature classification with k=5 or k=10), accuracy remained near chance (34–48%, none significantly above chance).

Representational similarity analysis (RSA): The representational similarities of networks within each category differed only slightly from the similarities across categories. The within/across similarity ratios ranged from 0.996 to 1.019 across all layers.

Weight similarity analysis: The weight similarities of networks within each category differed only slightly from the similarities across categories. Using permutation-invariant weight similarity with cosine metric, the within/across similarity ratios ranged from 0.998 to 1.000 across all layers.

Representation Decoding: We found that both of the relevant variables ($-x$, $-y$, $x+y$) could be decoded from all layers using a linear decoder. The variable $x+y$ was decoded nearly perfectly (mean $R > 0.999$) across all layers and model types. The variables $-x$ and $-y$ were decoded well in early layers (mean $R^2 \approx 0.84$–0.88 in layers 2–3) but with reduced accuracy at the output layer (mean $R^2 \approx 0.53$–0.57). Critically, decoding accuracy was nearly identical across algorithmic categories (e.g., for $-x$ in layer 2: Joint = 0.84 ± 0.10, Separate = 0.85 ± 0.12, Other = 0.86 ± 0.11).

Weight Decoding: We also tried to decode whether a network was `-x - y` or `-(x + y)` from its final weights. We used logistic regression decoders with inputs from individual layers as well as across all layers. The best individual-layer decoder (layer 3 weights) achieved 69% test accuracy for binary classification, but for multinomial classification (all three categories), the best decoder (also layer 3 weights) achieved only 50% test accuracy (chance level: 33%).

Magnitude of change during transfer learning: We investigated whether a network's learning speed reflected how much it changed during transfer learning. For each function pair, we asked whether the function that required more cumulative weight change was also the one whose loss was higher on more training batches. We found a weak association: cumulative weight change correctly predicted which function had higher loss on more batches for 64% of pairs, and the magnitude of the difference in cumulative weight change correlated with the magnitude of the learning speed difference as measured by percentage of batches with higher loss (Spearman $r = 0.38$, $p < .001$; per-pair range: $r = 0.27$–0.48). We then performed the same analysis using representational and weight similarity between the models before and after transfer learning. For representational similarity, the strongest continuous correlation was weak (Pearson $r = -0.27$ for whole-network RDMs using correlation distance and cosine RSA metric), and binary prediction was near chance (52%, dense layer, euclidean RDM, cosine RSA metric). For weight similarity, the strongest correlation was also weak (Pearson $r = 0.10$ for layer 3 weights using cosine similarity), with a best binary prediction of 54% (output layer, euclidean metric).

NTK: Finally, we tested whether the Neural Tangent Kernels (NTK) could predict which mapping in each pair a network would learn faster. The NTK analysis showed a systematic bias, predicting that the same function would be learned faster for all 96 networks in every pair. As a result, NTK predictions performed at chance level, with an average binary accuracy of 49% across the new input-output mapping pairs (range: 44%–53%).

Third Experiment

Our third experiment involved a *nonlinear* algebraic task with two variables. The procedure and results were similar to the first and second experiments.

We created 100 simple networks with the same architecture as the second experiment (2-8-4-8-1) and random initial weights. We used gradient descent (Adam) to minimize their mean squared error on the initial input-output mapping $(x,y) \mapsto (-x-y)\tanh(xy)$ for domain $[-50,50] \times [-50,50]$. The function $\tanh(xy)$ is positive in the first and third quadrants and negative in the second and fourth quadrants, creating a saddle-like surface that is essentially a continuous version of XOR, the simplest kind of nonlinearity. If one of the networks was unable to learn the task (loss > 10), we reinitialized and retrained it. This guaranteed that all 100 networks learned the initial task.

The initial mapping $(x,y) \mapsto (-x-y)\tanh(xy)$ could be generated by two different algorithms: `(-x - y)tanh(xy)` and `-(x + y)tanh(xy)`. We wanted to know: Is the network using an algorithm more like `(-x - y)tanh(xy)` or

`-(x + y)tanh(xy)`? We would expect a network using `(-x - y)tanh(xy)` to learn the mapping $(x,y) \mapsto (-x - 4y)\tanh(xy)$ faster than the mapping $(x,y) \mapsto -2.5(x + y)\tanh(xy)$, and a network using `-(x + y)tanh(xy)` to learn the mapping $(x,y) \mapsto -2.5(x+y)\tanh(xy)$ faster than the mapping $(x,y) \mapsto (-x-4y)\tanh(xy)$. In light of this contrast, we used a network's learning speed to determine whether it was using an algorithm more like `(-x - y)tanh(xy)` or `-(x + y)tanh(xy)`.

We tested our trained networks on six pairs of new input-output mappings, each multiplied by $\tanh(xy)$. The coefficients were adjusted so that each pair of functions had a similar range of values, ensuring a fair comparison: $(x,y) \mapsto (-x-2y)$ and $-1.5(x+y)$, $(-x-6y)$ and $-3.5(x+y)$, $(-x-8y)$ and $-4.5(x+y)$, $(-x-10y)$ and $-5.5(x+y)$, $(-x-14y)$ and $-7.5(x+y)$, and $(-x-16y)$ and $-8.5(x+y)$. Our method attributed `(-x - y)tanh(xy)` to 66 networks and `-(x + y)tanh(xy)` to 18 networks. Thus, it attributed algorithms to 84% of the networks.

We tested the sensitivity of our results to our hyperparameter settings:

Threshold	Majority ≥ 4	Majority ≥ 5	Majority $= 6$
0.50	112/34/4	104/32/14	94/27/29
0.55	111/32/7	102/28/20	93/25/32
0.60	109/29/12	101/26/23	91/23/36
0.65	109/27/14	98/24/28	89/19/42
0.70	108/24/18	96/23/31	83/17/50

Table 3: Sensitivity analysis for the third experiment (150 networks). Each cell shows the number of networks categorized as `(-x - y)tanh(xy)` / `-(x + y)tanh(xy)` / other.

We validated our method using two held-out pairs of input-output mappings within the range of the other input-output mappings: $(-x-4y)\tanh(xy)$ and $-2.5(x+y)\tanh(xy)$, and $(-x-12y)\tanh(xy)$ and $-6.5(x+y)\tanh(xy)$. Our categorizations predicted performance on these other input-output mappings 71% and 77%, respectively. We also tested our trained networks on two more pairs of input-output mappings outside the range of the other input-output mappings: $(-x-20y)\tanh(xy)$ and $-10.5(x+y)\tanh(xy)$, and $(-x-32y)\tanh(xy)$ and $-16.5(x+y)\tanh(xy)$. Our categorizations predicted performance on these other input-output mappings 62% and 10%, respectively. The poor performance on the most extreme out-of-range pair suggests the transfer learning profile may not generalize as well to very distant functions in this nonlinear setting.

As we now show, the categorizations were not apparent from standard behavioral or parts-based approaches, with the possible exception of error pattern decoding. Like in the previous experiments, our analyses subsampled the models from a larger run to ensure equal representation across categories in our analyses (n=32 per category).

Generalization: We considered the networks' performance on the original domain, $[-50,50] \times [-50,50]$, as well as on two out-of-distribution domains, $[50,150] \times [50,150]$ and $[500,600] \times [500,600]$. We found that, for each domain, performance did not significantly correlate with category (all Pearson and Spearman correlations non-significant, $p > .05$, except for one within-group correlation). The median performance values differed somewhat between groups but with substantial overlap in distributions. Classifiers trained with logistic regression to predict a network's category were at or below chance level (28%, 34%, and 28% for the three domains, respectively; chance: 33%). Clustering by performance (k-means, k=3) failed to recover our algorithmic categories (adjusted rand indices ≈ 0 for all three domains).

Error pattern analysis: We divided the inputs into bins, averaged the error within each bin, and clustered networks by these error profiles. However, this produced clusters that bore little relationship to our algorithmic categories (adjusted rand index ≈ 0.01 across bin sizes 4, 6, and 10). We also attempted to decode a network's category from its error patterns using a neural network classifier. Unlike the previous experiment, decoding accuracy was well above chance (73%, chance: 33%). When restricting to the most discriminative input locations (top 5–50 features, selected using k-NN single-feature classification with k=5 or k=10), accuracy remained well above chance (70% for 50 features and k=5; worse performance for fewer features or higher k, ranging down to 49%). This suggests that error patterns contain more information about algorithmic category in this nonlinear setting than in the linear setting. But error patterns alone cannot tell us which algorithm a network is using because those patterns still need to be interpreted.

Representational similarity analysis (RSA): The similarities within each category differed only slightly from the similarities across categories. The within/across similarity ratios ranged from 0.991 to 1.103 across all layers.

Weight similarity analysis: The similarities within each category differed only slightly from the similarities across categories. Using permutation-invariant weight similarity with cosine metric, the within/across similarity ratios ranged from 0.999 to 1.006 across all layers.

Representation Decoding: We found that the relevant variables ($-x$, $-y$, $x+y$) could be decoded from all layers using a linear decoder, though with more variation than in the second experiment. The variable $x+y$ was decoded nearly perfectly in early layers (mean $R > 0.99$) but with reduced accuracy at the output layer (mean $R^2 \approx 0.63$). The variables $-x$ and $-y$ were decoded reasonably well in early layers (mean $R^2 \approx 0.77$–0.86 in layers 2–3) but with reduced accuracy at the output layer (mean $R^2 \approx 0.27$–0.43). Decoding accuracy was similar across algorithmic categories (e.g., for $-y$ in layer 2: Joint = 0.82 ± 0.12, Separate = 0.78 ± 0.13, Other = 0.77 ± 0.14).

Weight Decoding: We also tried to decode whether a network was `(-x - y)tanh(xy)` or `-(x + y)tanh(xy)` from its final weights. We used logistic regression decoders with inputs from individual layers as well as across all layers. The best individual-layer decoder (layer 2 weights) achieved 70%

test accuracy for binary classification (chance: 50%), and for multinomial classification (all three categories), the best decoder (layer 3 weights) achieved 66% test accuracy (chance: 33%).

Magnitude of change during transfer learning: We investigated whether a network's learning speed reflected how much it changed during transfer learning. For each function pair, we asked whether the function that required more cumulative weight change was also the one whose loss was higher on more training batches. We found a weak association: cumulative weight change correctly predicted which function had higher loss on more batches for 68% of pairs, and the magnitude of the difference in cumulative weight change correlated with the magnitude of the learning speed difference as measured by percentage of batches with higher loss (Spearman $r = 0.57$, $p < .001$; per-pair range: $r = 0.51$–0.64). We then performed the same analysis using representational and weight similarity between the models before and after transfer lerning. For representational similarity, the strongest continuous correlation was weak (Pearson $r = 0.18$ for layer 1 RDMs using correlation distance and Pearson RSA metric), and binary prediction was near chance (57%, all layers concatenated, correlation RDM, Pearson RSA metric). For weight similarity, the strongest correlation was also weak (Pearson $r = 0.12$ for layer 2 weights using correlation metric), with a best binary prediction of 57% (output layer, euclidean metric).

NTK: Finally, we tested whether Neural Tangent Kernels (NTK) could predict which finetuning function in each pair a network would learn faster. The NTK analysis showed a systematic bias, predicting that the same function would learn faster for 90 out of 96 networks in every pair. As a result, NTK predictions performed at chance level, with an average binary accuracy of 49% across the input-output mappings (range: 47%–52%).

Fourth Experiment

Our fourth experiment involved a shape categorization task. Networks were trained on:

$$\texttt{shape} \cdot \Big[\texttt{color} + \texttt{background} + \texttt{area}\Big] > 0 \qquad (1)$$

They were then trained on new input-output mappings:

$$\texttt{altshape} \cdot \Big[\texttt{color} + \texttt{background} + \texttt{area}\Big] > 0 \qquad (2)$$

$$\texttt{shape} \cdot \Big[-\texttt{color} - \texttt{background} + \texttt{area}\Big] > 0 \qquad (3)$$

$$\texttt{shape} \cdot \Big[-\texttt{color} + \texttt{background} - \texttt{area}\Big] > 0 \qquad (4)$$

$$\texttt{shape} \cdot \Big[\texttt{color} - \texttt{background} - \texttt{area}\Big] > 0 \qquad (5)$$

There were six variants corresponding to the six different ways of grouping shapes for training and subsequent transfer learning.

Variant	`shape`	`altshape`
v1	ellipse+capsule	ellipse+circle
v2	ellipse+square	ellipse+circle
v3	ellipse+circle	ellipse+square
v4	ellipse+square	ellipse+capsule
v5	ellipse+capsule	ellipse+square
v6	ellipse+circle	ellipse+capsule

Table 4: Shape groupings for the six training variants in the fourth experiment.

To categorize each network, we compared its loss on the shape recombination task (2) to its losses on the variable inversion tasks (3-5). At each step during transfer learning, we computed the mean and standard deviation (σ) of the losses on the variable inversion tasks. The network received a score of 1 if its loss on the shape recombination task was much higher than its losses on the comparison losses, a score of 0 if its loss was around the same or lower than the comparison losses, and a score of .5 if it was in the gray zone in the middle. More precisely:

$$\text{score} = \begin{cases} 1 & \text{if } \ell_2 \geq \max(\ell_3, \ell_4, \ell_5) + \sigma \\ 0 & \text{if } \ell_2 < \text{mean}(\ell_3, \ell_4, \ell_5) + \sigma \\ 0.5 & \text{otherwise} \end{cases}$$

It was categorized based on its average score across all steps: Joint if that average exceeded .60, Separate if it fell below .40, and Other otherwise. Below is the distribution across variants:

Variant	Joint	Separate	Other
v1	10	0	0
v2	0	10	0
v3	4	3	3
v4	0	10	0
v5	10	0	0
v6	3	2	5
Total	27	25	8

Table 5: Distribution of algorithmic categories across training variants in the fourth experiment.

In order to control for any differences in initial training conditions, our analyses focused on a variant that produced models belonging to all three categories, namely v3.

Sensitivity Analysis: We initially trained 362 networks on v3. We tested the sensitivity of our categorization to score threshold settings:

Thresholds	Joint	Separate	Other
0.50/0.50	275	75	12*
0.55/0.45	239	55	68
0.60/0.40	166	36	160
0.65/0.30	99	25	238
0.70/0.30	49	19	294

*12 networks had average scores of exactly 0.50.

Table 6: Sensitivity analysis for the fourth experiment.

Across all threshold settings, the ratio of joint to separate detectors remained consistently around 3–4:1.

Our subsequent analyses used the default thresholds (0.60/0.40) and subsampled the models to ensure equal representation across categories (n=25 per category). As we now show, the categorizations were not apparent from standard behavioral or parts-based approaches.

Performance: We evaluated all 75 subsampled networks on 20,000 test images. Performance did not differ across algorithmic categories (Joint mean accuracy $= 0.983 \pm 0.004$; Separate mean accuracy $= 0.983 \pm 0.005$; Other mean accuracy $= 0.984 \pm 0.005$). A one-way ANOVA confirmed no significant difference ($F = 0.46$, $p = .63$), and a logistic regression classifier trained to predict a network's category from its accuracy achieved only 33% (chance level: 33%). Performance was also nearly identical when broken down by shape (e.g., for ellipses: Joint $= 0.980 \pm 0.006$, Separate $= 0.980 \pm 0.005$, Other $= 0.980 \pm 0.007$; similar patterns for squares, circles, and capsules).

Performance-based clustering: Clustering networks by their overall accuracy (k-means, $k = 3$) failed to recover our algorithmic categories (adjusted Rand index $= 0.004$; silhouette score $= 0.625$; $\chi^2 = 3.19$, $p = .53$).

Error pattern analysis: We computed per-shape error rates for each network and clustered networks by these error profiles (k-means, $k = 3$). The mean error rates were low across all shapes (ellipse: 0.020 ± 0.006; square: 0.015 ± 0.005; circle: 0.018 ± 0.006; capsule: 0.014 ± 0.005). Clustering by these error profiles bore little relationship to our algorithmic categories (adjusted Rand index $= 0.004$; silhouette score $= 0.373$). We also attempted to decode a network's category from its error patterns using a neural network classifier trained on binary error vectors (2,000 test images per model, 5-fold cross-validation). Accuracy was not significantly above chance for raw errors (37%, chance: 33%) and only marginally above chance for per-model normalized errors (41%). When restricting to the most discriminative images (top 10–200 images, selected using k-NN single-feature classification with $k = 10$), the best accuracy was 61% for 25 features. Performance varied with feature count (51% for 10 features, 41% for 50, 48% for 100, 40% for 200) and only the 25-feature result was significantly above chance.

Representational similarity analysis (RSA): For each network and layer (conv1, conv2, conv3, dense), we generated a representational dissimilarity matrix (RDM) using Euclidean distance and measured RDM similarity using cosine distance. The similarities within each category differed only slightly from the similarities across categories. Using cosine-euclidean RSA, the within/across similarity ratios were: layer $1 = 1.003$, layer $2 = 1.001$, layer $3 = 1.000$, dense $= 1.001$, all layers concatenated $= 0.999$. None of these ratios indicated meaningful clustering by algorithmic category.

Weight similarity analysis: We computed pairwise weight similarity between layers of the convolutional networks using cosine similarity of their flattened weight vectors after optimally aligning spatial kernels using the Hungarian algorithm. This algorithm finds the permutation of spatial kernels in one network that maximizes pairwise cosine similarity with the other, making the metric invariant to the arbitrary ordering of spatial kernels. For fully connected layers, we aligned individual weights. We computed these similarities across all four layers (conv1, conv2, conv3, dense). The similarities within each category differed only slightly from the similarities across categories. The within/across similarity ratios were essentially 1.0 across all layers: layer $1 = 1.00$ ($p = .92$), layer $2 = 1.00$ ($p = .03$), layer $3 = 1.00$ ($p = .23$), dense $= 0.99$ ($p = 1.00$). Only layer 2 showed a marginally significant permutation test, and the actual ratio difference was negligible (within: 0.220, across: 0.219).

Representation decoding: We extracted activations from all four layers (conv1, conv2, conv3, dense) for 4,000 test images and linearly decoded the latent variables (`color`, `background`, `area`, `scale`, `orientation`) from each layer. The task-relevant variables were decoded well from the convolutional layers, with R^2 values generally increasing from conv1 to conv3 (e.g., `area`: mean $R^2 \approx 0.73$ in conv1, 0.88 in conv2, 0.93 in conv3), but decoding accuracy dropped at the dense layer for `color` (mean $R^2 = 0.28$) and `background` (mean $R^2 = 0.45$). Critically, decoding accuracy was similar across algorithmic categories, with only small absolute differences (e.g., `color` R^2 in conv3: Joint $= 0.971 \pm 0.007$, Separate $= 0.962 \pm 0.006$, Intermediate $= 0.967 \pm 0.008$).

Weight decoding: We tried to decode whether a network was a joint or separate detector from its final weights. We used logistic regression decoders with PCA-reduced inputs (30 components) from individual layers as well as across all layers. For multinomial classification (all three categories), the best individual-layer decoder (layer 2 weights, 46.9% variance explained by 30 PCA components) achieved only 41% test accuracy (chance level: 33%; CV $= 0.413 \pm 0.065$). The other layers performed at or near chance: layer $1 = 36\% \pm 0.116$, layer $3 = 32\% \pm 0.050$, dense $= 37\% \pm 0.090$. The all-layer decoder achieved 37% (± 0.033). These results indicate that the algorithmic differences identified by transfer learning are not readily decodable from the networks' weights.

Magnitude of change during transfer learning: We investigated whether the magnitude of change during transfer learning differed across algorithmic categories, using both weight similarity and representational similarity before and after transfer learning (cosine metric). For weight similarity, changes were similar across categories for most layer-

task combinations ($p > .05$), with the few significant results involving negligible absolute differences (e.g., layer 1, shape recombination: Joint $= 0.997 \pm 0.001$, Separate $= 0.998 \pm 0.001$). For representational similarity, the pattern was similar: most comparisons were non-significant, and where differences reached significance the absolute differences were small (e.g., layer 3, input-output mapping (2): Joint $= 0.740 \pm 0.041$, Separate $= 0.705 \pm 0.041$; $p = .036$). We also compared the magnitude of change on the alternative shape task to the comparison variable inversion tasks. For both weight and representational similarity, all three algorithmic categories showed the same pattern: less change on the alternative shape task than on the comparison tasks, with the difference growing in deeper layers. These patterns did not differ meaningfully across categories.

We further asked whether the magnitude of change during transfer learning correlated with learning speed. We computed Spearman correlations between categorization score and the difference in similarity (alternative shape task minus comparison tasks) at each layer. For weight similarity, correlations were weak and mostly non-significant (strongest: dense layer, $\rho = -0.27$, $p = .02$). For representational similarity, correlations were also weak (strongest: layer 1, $\rho = -0.40$, $p < .001$). Correlations between loss difference and change difference were uniformly non-significant across all layers for both metrics. We also tested whether a network that changed less on the alternative shape task than the comparison tasks was more likely to be categorized as Joint. Binary prediction accuracy ranged from 42–60% across layers and metrics (chance: 50%). Clustering networks by their change profiles across layers and tasks (k-means, $k = 3$) failed to recover our algorithmic categories (adjusted Rand index $= 0.002$).

NTK: Finally, we tested whether Neural Tangent Kernels (NTK) could predict a network's algorithmic category. We computed each network's average NTK on the shape recombination task (2) and the variable inversion tasks (3-5). The NTK captures how easily the network can adjust its outputs to fit the finetuning tasks under gradient descent in a locally linear regime. The averaged NTK summarizes this over the full input space. If a network's NTK predicted its algorithmic category, Joint networks should have lower average NTK on (2) relative to (3-5), because shape recombination is harder for joint detectors. Separate networks should have similar average NTK across all tasks. We fit thresholds on this relative score to predict each network's category. For three-way classification (Joint vs. Separate vs. Intermediate), accuracy was 44.6% (chance: 33%). For binary classification (Joint vs. Separate only), accuracy was 57.1% (chance: 50%). Thus, NTK does not meaningfully distinguish the algorithmic categories.

Fifth Experiment

Our fifth experiment involved a 3D shape categorization task using Google's 3DShapes dataset [57].

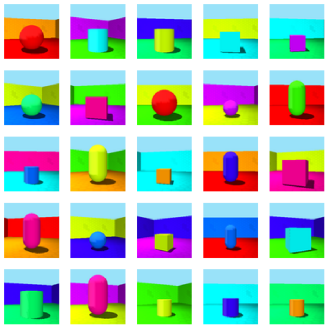

Table 7: Example images from Google's 3DShapes dataset.

Networks were trained on:

$$\texttt{shape} \cdot \Big[\texttt{color} + \texttt{wall} + \texttt{scale}\Big] > 0 \quad (1)$$

They were then trained on new mappings (transfer learning):

$$\texttt{altshape} \cdot \Big[\texttt{color} + \texttt{wall} + \texttt{scale}\Big] > 0 \quad (2)$$

$$\texttt{shape} \cdot \Big[\texttt{color} - \texttt{wall} - \texttt{scale}\Big] > 0 \quad (3)$$

$$\texttt{shape} \cdot \Big[-\texttt{color} + \texttt{wall} - \texttt{scale}\Big] > 0 \quad (4)$$

$$\texttt{shape} \cdot \Big[-\texttt{color} - \texttt{wall} + \texttt{scale}\Big] > 0 \quad (5)$$

There were six variants corresponding to the six different ways of grouping shapes for training and subsequent transfer learning.

Variant	shape	altshape
v1	cylinder+cone	cylinder+sphere
v2	cylinder+cube	cylinder+sphere
v3	cylinder+sphere	cylinder+cube
v4	cylinder+cube	cylinder+cone
v5	cylinder+cone	cylinder+cube
v6	cylinder+sphere	cylinder+cone

Table 8: Shape groupings for the six training variants in the fifth experiment.

The categorization procedure was the same as in the fourth experiment.

Variant	Joint	Separate	Other
v1	6	1	3
v2	0	10	0
v3	1	9	0
v4	0	10	0
v5	1	3	6
v6	0	10	0
Total	8	43	9

Table 9: Distribution of algorithmic categories across training variants in the fifth experiment.

In order to control for any differences in initial training conditions, our analyses focused on a variant that produced models belonging to all three categories, namely variant v1.

Sensitivity Analysis: We initially trained 200 networks on v1. We tested the sensitivity of our categorization to score threshold settings:

Thresholds	Joint	Separate	Other
0.50/0.50	115	73	12*
0.55/0.45	102	61	37
0.60/0.40	84	49	67
0.65/0.35	60	33	107
0.70/0.30	40	25	135

*12 networks had average scores of exactly 0.50.

Table 10: Sensitivity analysis for the fifth experiment.

Across all threshold settings, the ratio of joint to separate detectors remained consistently around 1.5–1.7:1. Our subsequent analyses used the default thresholds (0.60/0.40) and subsampled the models to ensure equal representation across categories (n=25 per category). As we now show, the categorizations were not apparent from standard behavioral or parts-based approaches.

Performance: We evaluated all 75 subsampled networks on 96,000 test images. Performance did not differ across algorithmic categories (Joint mean accuracy $= 0.994 \pm 0.002$; Separate mean accuracy $= 0.994 \pm 0.003$; Other mean accuracy $= 0.994 \pm 0.003$). A one-way ANOVA confirmed no significant difference ($F = 0.39$, $p = .68$), and a logistic regression classifier trained to predict a network's category from its accuracy achieved only 33% (chance level: 33%). Performance was also similar when broken down by shape (e.g., for cubes: Joint $= 0.992 \pm 0.003$, Separate $= 0.992 \pm 0.003$, Other $= 0.992 \pm 0.003$; similar patterns for spheres, cylinders, and cones).

Performance-based clustering: Clustering networks by their overall accuracy (k-means, $k = 3$) produced clusters that were weakly associated with our algorithmic categories (adjusted Rand index $= 0.078$; silhouette score $= 0.574$; $\chi^2 = 15.43$, $p = .004$). However, the clusters primarily reflected differences in overall accuracy rather than algorithmic category, and no cluster was dominated by a single type (e.g., the largest cluster of 33 networks contained 6 Joint, 16 Separate, and 11 Intermediate).

Error pattern analysis: Error rates were very low, with only cube shapes producing errors (mean error rate: 0.006 ± 0.003). Clustering by error profiles produced the same results as performance-based clustering (adjusted Rand index $= 0.078$; silhouette score $= 0.574$), since errors on the single shape were effectively equivalent to overall performance. We also attempted to decode a network's category from its error patterns using a neural network classifier trained on binary error vectors (2,000 test images per model, 5-fold cross-validation). Accuracy was not significantly above chance for raw errors (36%, chance: 33%) or for per-model normalized errors (28%). When restricting to the most discriminative images (top 10–200 images, selected using k-NN single-feature classification with $k = 10$), no feature count achieved accuracy significantly above chance (best: 39% for 50 features).

Representational similarity analysis (RSA): For each network and layer (conv1, conv2, conv3, dense), we generated a representational dissimilarity matrix (RDM) using Euclidean distance and measured RDM similarity using cosine distance. The within/across similarity ratios were close to 1.0 across all layers: layer 1 $= 1.001$, layer 2 $= 0.995$, layer 3 $= 1.001$, dense $= 1.015$, all layers concatenated $= 0.995$. None of these ratios were statistically significant (all $p > .20$).

Weight similarity analysis: We computed pairwise weight similarity using the same permutation-invariant cosine similarity method as in the fourth experiment. The within/across similarity ratios were essentially 1.0 across all layers: layer 1 $= 1.00$ ($p = .53$), layer 2 $= 1.00$ ($p = .94$), layer 3 $= 1.00$ ($p = .57$), dense $= 1.02$ ($p < .001$). The dense layer showed a statistically significant ratio, but the absolute difference was negligible (within: 0.040, across: 0.039).

Representation decoding: We extracted activations from all four layers (conv1, conv2, conv3, dense) for a random subset of 4,000 test images and linearly decoded the latent variables (`color`, `wall`, `scale`, `floor`, `orientation`) from each layer. The task-relevant variables were decoded well from the convolutional layers (e.g., `color` $R^2 \approx 0.95$ in conv1, 0.99 in conv2), but decoding accuracy dropped at the dense layer for `floor` ($R^2 = 0.12$) and was near chance for `orientation` ($R^2 = 0.01$), consistent with the network discarding task-irrelevant features. Critically, decoding accuracy was similar across algorithmic categories. Kruskal-Wallis tests reached significance for some variables in conv3 (`wall`: $p = .030$; `color`: $p = .032$; `scale`: $p = .002$), but the absolute differences were small (e.g., `scale` R^2 in conv3: Joint $= 0.910 \pm 0.034$, Separate $= 0.924 \pm 0.021$, Intermediate $= 0.938 \pm 0.018$). No comparisons were significant in the dense layer (all $p > .10$).

Weight decoding: We used logistic regression decoders with PCA-reduced inputs (30 components) from individual layers as well as across all layers. For multinomial classification (all three categories), all layers performed at or below chance: layer $1 = 31\% \pm 0.124$, layer $2 = 31\% \pm 0.100$, layer $3 = 39\% \pm 0.107$, dense $= 31\% \pm 0.053$ (chance level: 33%). The all-layer decoder achieved 33%. These results indicate that the algorithmic differences identified by transfer learning are not readily decodable from the networks' weights.

Magnitude of change during transfer learning: We investigated whether the magnitude of changes during transfer learning differed across algorithmic categories, using both weight similarity and representational similarity before and after transfer learning (cosine metric). For weight similarity, Joint networks showed slightly more change than Separate and Intermediate networks, particularly in the dense layer (e.g., dense, shape recombination: Joint $= 0.833 \pm 0.061$, Separate $= 0.881 \pm 0.040$, Intermediate $= 0.875 \pm 0.030$; $p = .003$). Several layer-task combinations reached signifi-

cance in layers 1 and dense, but the absolute differences remained small. For representational similarity, most comparisons were non-significant (only one layer-task combination reached significance $p = .031$). We also compared the magnitude of change on the alternative shape task to the comparison variable inversion tasks. For representational similarity, all three algorithmic categories showed the same pattern: less change on the alternative shape task than on the comparison tasks. For weight similarity, the pattern was reversed in layers 2–3 and dense, where more change occurred on the alternative shape task, but again these patterns did not differ meaningfully across categories.

We further asked whether the magnitude of change during transfer learning correlated with learning speed. We computed Spearman correlations between categorization score and the difference in similarity (alternative shape task minus comparison tasks) at each layer. For weight similarity, correlations were weak (strongest: layer 2, $\rho = -0.40$, $p < .001$). For representational similarity, correlations were also weak (strongest: dense layer, $\rho = -0.39$, $p < .001$). Correlations between loss difference and change difference were also weak across all layers for both metrics. We also tested whether a network that changed less on the alternative shape task than the comparison tasks was more likely to be categorized as Joint. Binary prediction accuracy ranged from 39–56% across layers and metrics (chance: 50%). Clustering networks by their change profiles across layers and tasks (k-means, $k = 3$) failed to recover our algorithmic categories (adjusted Rand index = 0.009).

NTK: Finally, we tested whether Neural Tangent Kernels (NTK) could predict a network's algorithmic category. We computed each network's average NTK on the shape recombination task (2) and the variable inversion tasks (3-5). If a network's NTK predicted its algorithmic category, Joint networks should have lower average NTK on (2) relative to (3-5), because shape recombination is harder for joint detectors. Separate networks should have similar average NTK across all tasks. We fit thresholds on this relative score to predict each network's category. For three-way classification (Joint vs. Separate vs. Intermediate), accuracy was 45.3% (chance: 33%). For binary classification (Joint vs. Separate only), accuracy was 60.0% (chance: 50%). Thus, NTK does not meaningfully distinguish the algorithmic categories.

Sixth Experiment

Our sixth experiment involved a nonlinear 3D shape categorization task using Google's 3DShapes dataset. Networks were trained on:

$$\texttt{shape} \cdot \Big[(\texttt{color} \cdot \texttt{wall}) + (\texttt{scale} \cdot \texttt{floor})\Big] > 0 \quad (1)$$

They were then trained on new mappings (transfer learning):

$$\texttt{altshape} \cdot \Big[(\texttt{color} \cdot \texttt{wall}) + (\texttt{scale} \cdot \texttt{floor})\Big] > 0 \quad (2)$$

$$\texttt{shape} \cdot \Big[-(\texttt{color} \cdot \texttt{wall}) + (\texttt{scale} \cdot \texttt{floor})\Big] > 0 \quad (3)$$

$$\texttt{shape} \cdot \Big[(\texttt{color} \cdot \texttt{wall}) - (\texttt{scale} \cdot \texttt{floor})\Big] > 0 \quad (4)$$

$$\texttt{shape} \cdot \Big[-(\texttt{color} \cdot \texttt{wall}) - (\texttt{scale} \cdot \texttt{floor})\Big] > 0 \quad (5)$$

$$\texttt{shape} \cdot \Big[(\texttt{color} \cdot \texttt{scale}) + (\texttt{wall} \cdot \texttt{floor})\Big] > 0 \quad (6)$$

There were six variants corresponding to the six different ways of grouping shapes for training and subsequent transfer learning.

Variant	shape	altshape
v1	cylinder+cone	cylinder+sphere
v2	cylinder+cube	cylinder+sphere
v3	cylinder+sphere	cylinder+cube
v4	cylinder+cube	cylinder+cone
v5	cylinder+cone	cylinder+cube
v6	cylinder+sphere	cylinder+cone

Table 11: Shape groupings for the six training variants in the sixth experiment.

The categorization procedure was the same as in the fourth experiment.

Variant	Joint	Separate	Other
v1	10	0	0
v2	0	8	2
v3	0	10	0
v4	0	10	0
v5	9	0	1
v6	0	10	0
Total	19	38	3

Table 12: Distribution of algorithmic categories across training variants in the sixth experiment.

In order to control for any differences in initial training conditions, our analyses focused on a variant that produced models belonging to all three categories in a larger run, namely v5.

Sensitivity Analysis: We trained 750 networks and tested the sensitivity of our categorization to score threshold settings:

Thresholds	Joint	Separate	Other
0.50/0.50	705	42	3*
0.55/0.45	702	36	12
0.60/0.40	688	30	32
0.65/0.35	662	27	61
0.70/0.30	636	25	89

*3 networks had average scores of exactly 0.50.

Table 13: Sensitivity analysis for the sixth experiment.

Across all threshold settings, the ratio of joint to separate detectors remained consistently around 17–23:1. Our subsequent analyses used the default thresholds (0.60/0.40) and subsampled the models to ensure equal representation across categories (n=25 per category). As we now show, the categorizations were not apparent from standard behavioral or parts-based approaches.

Performance: We evaluated all 75 subsampled networks on 96,000 test images. Performance did not differ across algorithmic categories (Joint mean accuracy $= 0.981 \pm 0.004$; Separate mean accuracy $= 0.980 \pm 0.004$; Other mean accuracy $= 0.980 \pm 0.004$). A one-way ANOVA confirmed no significant difference ($F = 1.62$, $p = .21$), and a logistic regression classifier trained to predict a network's category from its accuracy achieved only 35% (chance level: 33%). Performance was also similar when broken down by shape (e.g., for cubes: Joint $= 0.978 \pm 0.004$, Separate $= 0.976 \pm 0.005$, Other $= 0.977 \pm 0.005$; similar patterns for spheres, cylinders, and cones).

Performance-based clustering: Clustering networks by their overall accuracy (k-means, $k = 3$) failed to recover our algorithmic categories (adjusted Rand index $= 0.000$; silhouette score $= 0.562$; $\chi^2 = 4.39$, $p = .36$). The resulting clusters reflected differences in overall accuracy rather than algorithmic category, with model types distributed roughly uniformly within each cluster.

Error pattern analysis: Error rates were low across all shapes (cube: 0.023 ± 0.005; sphere: 0.018 ± 0.004; cylinder: 0.018 ± 0.004; cone: 0.018 ± 0.004). Clustering by error profiles bore little relationship to our algorithmic categories (adjusted Rand index $= -0.002$; silhouette score $= 0.351$). We also attempted to decode a network's category from its error patterns using a neural network classifier trained on binary error vectors (2,000 test images per model, 5-fold cross-validation). Accuracy was at chance for raw errors (33%) and marginally above chance for per-model normalized errors (47%, significantly above chance). When restricting to the most discriminative images (top 10–200 images, selected using k-NN single-feature classification with $k = 10$), the best accuracy was 56% for 25 features, and performance was variable across feature counts (39% for 10, 41% for 50, 43% for 100) and only the 200-feature result was significantly above chance (45%).

Representational similarity analysis (RSA): For each network and layer (conv1, conv2, conv3, dense), we generated a representational dissimilarity matrix (RDM) using Euclidean distance and measured RDM similarity using cosine distance. The within/across similarity ratios were close to 1.0 across all layers: layer 1 $= 0.999$, layer 2 $= 0.985$ ($p < .001$), layer 3 $= 0.999$, dense $= 1.005$, all layers concatenated $= 0.995$. The layer 2 ratio reached statistical significance but the absolute difference was small (within: 0.678, across: 0.679).

Weight similarity analysis: We computed pairwise weight similarity using the same permutation-invariant cosine similarity method as in the fourth experiment. The within/across similarity ratios were: layer 1 $= 1.01$ ($p < .001$), layer 2 $= 1.00$ ($p = .031$), layer 3 $= 1.00$ ($p = .64$), dense $= 0.98$ ($p = 1.00$). Layer 1 showed a statistically significant ratio, but the absolute difference was negligible (within: 0.353, across: 0.351).

Representation decoding: We extracted activations from all four layers (conv1, conv2, conv3, dense) for 4,000 test images and linearly decoded the latent variables (`color`, `wall`, `scale`, `floor`, `orientation`) from each layer. The task-relevant variables were decoded well from the convolutional layers (e.g., `color` $R^2 \approx 0.93$ in conv1, 0.97 in conv2), but decoding accuracy dropped at the dense layer for `floor` ($R^2 = 0.58$) and was near chance for `orientation` ($R^2 = 0.02$), consistent with the network discarding task-irrelevant features. Critically, decoding accuracy was similar across algorithmic categories. Kruskal-Wallis tests reached significance for `color` and `scale` in conv3 ($p = .013$ and $p = .043$, respectively) and in the dense layer ($p = .029$ and $p = .018$), but the absolute differences were small (e.g., `scale` R^2 in conv3: Joint $= 0.940 \pm 0.027$, Separate $= 0.918 \pm 0.034$, Intermediate $= 0.934 \pm 0.027$).

Weight decoding: We used logistic regression decoders with PCA-reduced inputs (30 components) from individual layers as well as across all layers. For multinomial classification (all three categories), the best individual-layer decoder (layer 3, 54.6% variance explained by 30 PCA components) achieved 43% test accuracy (chance level: 33%; CV $= 0.427 \pm 0.068$). The other layers performed at or below chance: layer 1 $= 23\% \pm 0.100$, layer 2 $= 31\% \pm 0.090$, dense $= 33\% \pm 0.112$. The all-layer decoder achieved 32%. These results indicate that the algorithmic differences identified by transfer learning are not readily decodable from the networks' weights.

Magnitude of change during transfer learning: We investigated whether the magnitude of changes during transfer learning differed across algorithmic categories, using both weight similarity and representational similarity between before and after transfer learning (cosine metric). For weight similarity, nearly all layer-task combinations were non-significant, with only two reaching significance ($p < .05$) and absolute differences remaining small. For representational similarity, most comparisons were also non-significant, though the full sign-inversion task (4) showed significant differences in layers 2–3 and dense, driven by Separate networks showing more representational change on this task (e.g., layer 3: Joint $= 0.886 \pm 0.064$, Separate $= 0.598 \pm 0.302$, Intermediate $= 0.824 \pm 0.106$). We also compared the magnitude of change on the alternative shape task to the comparison tasks. For both weight and representational similarity, all three categories showed more change on the alternative shape task than on the comparison tasks at all layers, and these patterns did not differ meaningfully across categories.

We further asked whether the magnitude of change during transfer learning correlated with learning speed. We computed Spearman correlations between categorization score and the difference in similarity (alternative shape task minus

comparison tasks) at each layer. For weight similarity, correlations were weak and non-significant across all layers. For representational similarity, correlations were weak in deeper layers (strongest: layer 3, $\rho = -0.48$, $p < .001$). Correlations between loss difference and change difference were weak for weight similarity and representational similarity (strongest: representational similarity, dense layer, $\rho = -0.54$, $p < .001$). We also tested whether a network that changed less on the alternative shape task than the comparison tasks was more likely to be categorized as Joint. Binary prediction accuracy ranged from 31–52% across layers and metrics (chance: 50%). Clustering networks by their change profiles across layers and tasks (k-means, $k = 3$) failed to recover our algorithmic categories (adjusted Rand index $= 0.022$).

NTK: Finally, we tested whether Neural Tangent Kernels (NTK) could predict a network's algorithmic category. We computed each network's average NTK on the shape recombination task (2) and the comparison tasks (3-6). If a network's NTK predicted its algorithmic category, then Joint networks should have lower average NTK on (2) relative to (3-6), because shape recombination is harder for joint detectors. Separate networks should have similar average NTK across all tasks. We fit thresholds on this relative score to predict each network's category. For three-way classification (Joint vs. Separate vs. Intermediate), accuracy was 40.0% (chance: 33%). For binary classification (Joint vs. Separate only), accuracy was 56.0% (chance: 50%). Thus, NTK does not meaningfully distinguish the algorithmic categories.