Using transfer learning to identify a neural system's algorithm

John Morrison (jmorrison@barnard.edu)
Professor of Philosophy and Cognitive Science, Barnard College, Columbia University

Nikolaus Kriegeskorte (nk2765@columbia.edu)
Professor of Psychology and Neuroscience, Columbia University

Benjamin Peters (benjamin.peters@ed.ac.uk)
Lecturer in Computational Cognitive Science, University of Edinburgh

Abstract

Algorithms generate input-output mappings through operations on representations. In cognitive science, we use algorithms to explain cognition. For example, we use tree-search algorithms to explain planning, reinforcement learning algorithms to explain exploration, and Bayesian algorithms to explain categorization. There are often many cognitive science algorithms consistent with a subject's performance on a task. How are we supposed to choose? It is natural to think of algorithms as causal models of brain processes. Thus, a natural method for choosing an algorithm is to look for parts in the brain corresponding to the steps of the algorithm. However, we haven't found many cognitive science algorithms using this method. This has led some to view cognitive science algorithms as merely normative, indicating the ideal input-output mapping without attributing any particular operation to the brain. It has led others to view cognitive science algorithms as merely useful fictions; useful insofar as they allow us to predict behavior, but fictional insofar as they inaccurately describe the causes of that behavior. They recommend explaining cognitive processes using other frameworks, such as dynamical systems theory. As an alternative, we suggest identifying a neural system's algorithm by assessing how quickly it learns alternative input-output mappings, that is, its transfer learning profile. The basic idea is that, depending on which algorithm is being used, different input-output mappings will be easier to learn, allowing us to recover its original algorithm from its transfer learning profile. We use artificial neural networks to demonstrate that this proposal productively applies to multiple networks and tasks. We conclude that transfer learning is a promising approach for integrating algorithms with neural networks and thus for integrating cognitive science with systems neuroscience and machine learning.

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

Introduction

Algorithms were originally invented to supplement cognition. For example, column addition was introduced to add large numbers, predicate logic to deduce theorems (Frege, 1893/1903), merge sort to organize data (von Neumann, 1945), tree-search to play chess (Shannon, 1950), recursive grammars to analyze sentences (Panini 350 BC), signal detection theory to identify planes (Marcum, 1947), Bayesian inference (e.g., conditionalization) to update scientific beliefs (Bayes, 1763), and reinforcement learning (e.g., Q-learning) to choose actions that maximize the expectation of cumulative reward (Watkins & Dayan, 1992). Cognitive science'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 cognitive processes (Chun & Most, 2022). At least in principle, this conjecture could also be extended to artificial neural networks. But how are we supposed to choose the relevant algorithm?

There are two standard methods. The first is to search for parts that correspond to the steps of the algorithm. The underlying assumption is that algorithms are causal models of brains and artificial neural networks, and therefore the parts of the relevant algorithm should correspond to parts in the relevant network. The parts in the network can include any mathematically definable construct of the network's basic elements, including points, regions, dimensions, and vectors within its space of possible activity as well as its space of possible weights. Standard parts-based analyses include decoding, RSA, and weight similarity analysis. This approach has had a number of successes. In neuroscience, they include path integration in the ring attractor network of flies (Kim, Rouault, Druckmann, & Jayaraman, 2017), sound localization in the nucleus laminaris of chickens and owls (Ashida & Carr, 2011), sequential sampling in the lateral intraparietal region of macaques (Shadlen & Kandel, 2021), and feature matching in the fusiform gyrus of humans and other primates (Chang & Tsao, 2017). In machine learning, they include modular addition (Nanda, Chan, Lieberum, Smith, & Steinhardt, 2023) and monotonicity natural language inference (Geiger, Wu, Potts, Icard, & Goodman, 2023), both in transformer models. Often, however, it turns out to be challenging or impossible to identify an algorithm through parts-based methods. There are several possible explanations: perhaps we haven't yet developed the right techniques for disentangling the relevant parts, perhaps the relevant parts are often too entangled to ever be disentangled, or perhaps algorithms do not provide accurate causal models. Regardless, given that we often can't find the relevant parts, many are drawn to a second, less demanding method.

The second method is to attribute algorithms based solely on input-output mappings, without searching for parts. This has been a standard method in cognitive science for decades, see e.g., (Niv, 2021), and it has become standard in machine learning as well, see e.g. (De Lillo, Floreano, & Antinucci, 2001; Kay et al., 2023; Lippl, Kay, Jensen, Ferrera, & Abbott, 2023) on transitive algorithms, (Hupkes, Dankers, Mul, & Bruni, 2020; Lake & Baroni, 2023) on compositional algorithms, and (Mathys, Daunizeau, Friston, & Stephan, 2011;

In D. Barner, N.R. Bramley, A. Ruggeri and C.M. Walker (Eds.), *Proceedings of the 47th Annual Conference of the Cognitive Science Society*

Orhan & Ma, 2017) on Bayesian algorithms. The main challenge is that the same input-output mapping is almost always consistent with many different algorithms. We can rule out some of them by taking into account how the network generalizes to new inputs and by enriching the output to include other measures, such as reaction time. Nonetheless, a network's input-output mapping is almost always consistent with a great many different algorithms, and those algorithms can differ in fundamental ways. As a result, algorithms would not be as informative as we would like; fundamental differences between algorithms would not track fundamental differences in the networks they describe. They would be would be mere distractions. More generally, algorithms would merely provide compressed descriptions of a network's input-output mapping.

The challenges of these methods have led some to regard cognitive science algorithms as merely useful fictions about the brain and artificial neural networks (P. S. Churchland, 1986; P. M. Churchland, 1989; Ramsey, 2007). Their view is that, like Ptolemaic earth-centered models of the universe, algorithms allow us to reliably predict behavior even though they misdescribe the causes of that behavior. These challenges have led others to conclude that we should regard cognitive science algorithms are merely normative, indicating the ideal input-output mapping without attributing any particular operation to the brain, e.g., (Griffiths, Chater, Kemp, Perfors, & Tenenbaum, 2010) on Bayesian algorithms. All of these authors recommend using other frameworks, such as dynamical systems theory, to describe processes in neural networks (Lillicrap & Kording, 2019; Lindsay & Bau, 2023). They might ultimately be proven right, but that would be disappointing given that algorithms have the potential to provide unique and valuable insights into neural networks. For example, they potentially allow us to discover meaningful similarities across networks with different architectures (e.g., members of different species), and meaningful differences across networks with similar architectures (e.g., members of the same species).

We will consider a third method: look for a match between an algorithm and a network's transfer learning profile. We assess the viability of this approach in a series of experiments involving artificial neural networks. We show that our method attributes algorithms when parts-based methods don't, preserves fundamental differences that input-output-approaches lose, reliably predicts performance, and provides a framework for investigating the underlying mechanisms. We conclude that transfer learning potentially provides a better method for attributing algorithms to neural networks, and thus a better foundation for cognitive science.

This method has a long history in cognitive science. For example, Tolman (1948) observed that, after mice wandered around a maze, they were faster at learning to navigate between two points. He concluded that they had already learned a map of the maze. Tolman's insight was 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 can't. This insight is also found in the literatures on learning to learn in psychology (Cormier & Hagman, 1987) and transfer learning in machine learning (Thrun & Pratt, 2012; Ruder, 2017). These literatures are full of examples where it takes takes less time to learn a new task after learning another task. For example, it takes less time for people to learn tennis after badminton, chess after checkers, multiplication after addition, and knitting after crocheting. Similarly, it takes less time for neural networks to identify broken bones after identifying faces, and to play Frostbite after Space Invaders. These literatures often have a practical focus; the goal is to help people and neural networks learn faster (Vafaeikia, Namdar, & Khalvati, 2020; Zhou et al., 2023). But their insight gives us a method to test for representations and algorithms in a neural network: consider how quickly it learns related tasks.

This method also builds on an insight found in recent work in machine learning on linear networks. The insight is that networks with the same input-output mapping can have different transfer learning profiles. In particular, Saxe et al. showed that linear networks with the same input-output mappings can nonetheless have different transfer learning profiles (Saxe, McClelland, & Ganguli, 2014). We propose using differences of this kind to recover the networks' algorithms.

It might be tempting to regard transfer learning as just another method for searching for parts corresponding to the steps of an algorithm. And, at least some authors have suggested just this. For example, Davies (1987) suggests it as a test for certain linguistic algorithms. More recently, Maloney and Mamassian (2009) and Koblinger, Fiser, and Lengyel (2021) suggest it as a test for whether a network has parts corresponding to the steps of Bayesian algorithms. But, as the following experiments will illustrate, the transfer learning method attributes algorithms even when parts-based methods do not. In the Discussion, we will consider a network handcrafted so that the transfer learning method and parts-based methods attribute different algorithms. As a result, if we regard parts-based methods as providing the "ground truth" about a network's algorithm, it's unclear whether transfer learning is a reliable method. In the Discussion, we will consider the possibility that transfer learning could itself provide the ground truth about a network's algorithm.

In the next three sections, we consider networks trained on three different tasks: a linear algebraic task (Section 2), a linear 2D image classification task (Section 3), and a non-linear 3D image classification task (Section 4). For each task, the possible algorithms differ in the number and magnitude of changes required to transition to other algorithms. We use a network's learning speed on the input-output mappings corresponding to those other algorithms to identify its algorithm.

First Experiment: Algebra

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 (num-

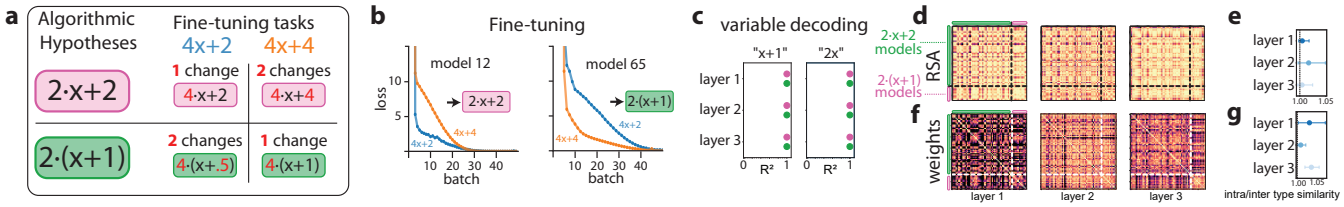

Figure 1: After training on $2x+2$, we fine-tuned networks on pairs of new mappings (**a, b**). We attributed `2 · x + 2` or `2 · (x + 1)` depending on which mapping in each pair it learned faster. These algorithms were not evident in the decoding of intermediate variables (**c**) or their representational similarities (**d**) as evidenced by their within vs. between type similarity ratio (**e**), and only weakly reflected in the permutation-invariant cosine similarities between their weights (**f, g**).

ber of nodes for each layer: 1-8-4-8-1). The 100 networks were feedforward, fully connected, and ReLU and had different random initial weights.

We used stochastic gradient descent (SGD) to minimize their mean squared error on the initial input-output mapping $x \mapsto 2x+2$ for domain $[-50, 50]$. This mapping 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)`?

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. 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. 1a). 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 fine-tuned 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 fine-tuned 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 during the first epoch (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. 1b).

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. Our method attributed `2 · x + 2` to 13 of them and `2 · (x + 1)` to 54 of them. Thus, it attributed algorithms to 67% of the networks.

The odds of getting these results from random fluctuations in the networks is close to zero. Consider that if each fine-tuning pair was a coin flip, the probability of attributing an algorithm to a network would be $2 \times \left(\frac{5}{6}\right)\left(\frac{2}{1}\right)^6 + \left(\frac{6}{6}\right)\left(\frac{2}{1}\right)^6 \approx 21.9\%$ and the probability of attributing an algorithm to at least 67 out of 100 networks would be $\sum_{k=67}^{100}\binom{100}{k}(0.219)^k(0.781)^{100-k}$ and thus $< 10^{-13}$. Note that this calculation assumes that each fine-tuning pair always favors one algorithm or the other. Dropping that assumption would push the probability even lower.

We cross-validated our method by fine-tuning our networks on two additional pairs of input-output mappings: $32x+2$ and $32x+32$ as well as $64x+2$ and $64x+64$. We then considered all 28 ways of grouping our 8 input-output mappings into groups of 6, and re-applied our method to each grouping. We found an average agreement of $\approx .85$ with a standard deviation of $\approx .07$.

How do the categorizations derived from transfer learning compare to common approaches, like within- and out-of-distribution generalization, representational similarity analysis, decoding analyses, and weight similarity?

To start, using an input-output approach, 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 whether it was categorized as `2 · x + 2` or `2 · (x + 1)` (p-values of .263, .264, and .227, respectively.) In fact, for each domain, classifiers trained with logistic regression to predict a network's algorithm reverted to predicting `2 · (x + 1)` for all networks because that maximized accuracy at .54. 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. As noted earlier, a network's input-output mapping does not significantly constrain the space of possible algorithms.

Next, using a parts-based approach, we investigated the

similarities of network activations using 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 within $2 \cdot x + 2$ networks, within $2 \cdot (x + 1)$ networks, and between $2 \cdot x + 2$ and $2 \cdot (x + 1)$ networks. We considered all of the standard metrics: Euclidean distance, correlational distance, cosine distance, procrustes distance, and centered kernel alignment. The similarities within category did not significantly differ from the similarities across categories (see Fig. 1d,e). Moreover, these categories could only partially be recovered using k-nearest neighbors, suggesting that these categories are only weakly in these similarities measures (mean accuracies between .65 and .74 across layers with baseline of .7). Finally, 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.

Next, using a different parts-based approach, we investigated the similarities of network weights using cosine 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 $2 \cdot x + 2$ networks, within $2 \cdot (x + 1)$ networks, and between $2 \cdot x + 2$ and $2 \cdot (x + 1)$ networks. The similarities within each category differed only slightly from the similarities across categories, and that difference was too small to reliably predict whether a network was categorized $2 \cdot x + 2$ or $2 \cdot (x + 1)$ (see Fig. 1i). 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.

Decoding enables a hypothesis-driven search for parts in a network. Using decoding, we found that both of the relevant variables ($x+1$, $2x$) could be perfectly decoded from all three layers using a linear decoder (see Figure). 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 consider decoding again in subsequent experiments.

We also tried to decode whether a network was $2 \cdot x + 2$ or $2 \cdot (x + 1)$ from its final weights. In particular, we trained a linear decoder with an input size of 101 (the number of parameters in our networks) and an output size of 3 (the two algorithms and "neither"). We trained the decoder for 50 epochs using an 80-20 split, with 80 of the networks used for training and 20 of the networks used for testing. We tried 100 different 80-20 splits, and the maximum decoder performance was .65. The next best performance was .55. We then tried more complex decoders with two hidden layers, and again the maximum decoder performance was .65. 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.

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.

Second Experiment: 2D Classification

Our second experiment involved a linear two-dimensional shape categorization task. We created 100,000 64×64 monochromatic images using five latent variables scaled between -1 and 1: `shape` (square, circle, ellipse, capsule), `color` (object's color, from white to black), `background` (background color, from white to black), `area`, and `orientation`. The shapes were divided into two groups, one assigned -1, the other 1. We labeled all of the images 0 or 1 using six different versions of the following:

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

The six different versions corresponded to the six different ways of dividing the shapes into groups. In order to learn variations of this task, a network would presumably have to estimate the values of all the latent variables.

There are at least two algorithms that a network could use to estimate the value of `shape`. First, it could jointly detect the shapes in a group and jointly assign them a value of -1 (or 1). Second, it could separately detect both of the shapes and separately assign them a value of -1 (or 1). These algorithms differ in the number and magnitude of changes required to transition to other algorithms. For example, if a network used a joint detector, it should be slower at learning new labels that group the shapes in alternative ways because it would need to learn to distinguish shapes in the same group, and faster at learning new labels that invert one or more of the other latent variables. In contrast, if a network used separate detectors, it should be faster learning new labels that group the shapes in alternative ways (see Fig. 2a). In light of this contrast, we propose using a network's learning speed to determine whether it was using joint or separate detectors.

We trained 10 CNNs on each of the six different versions of the training labels above, for a total of 60 models. The networks had 16 3×3 filters with 2×2 max pooling, 32 3×3 filters with 2×2 max pooling, 64 3×3 filters with 2×2 max pooling, a flattened dense layer with 128 nodes, and an outpoint node with a sigmoid activation function. We then fine-tuned all 60 networks on the labels described above. Our method attributed joint detectors to 29 networks and separate

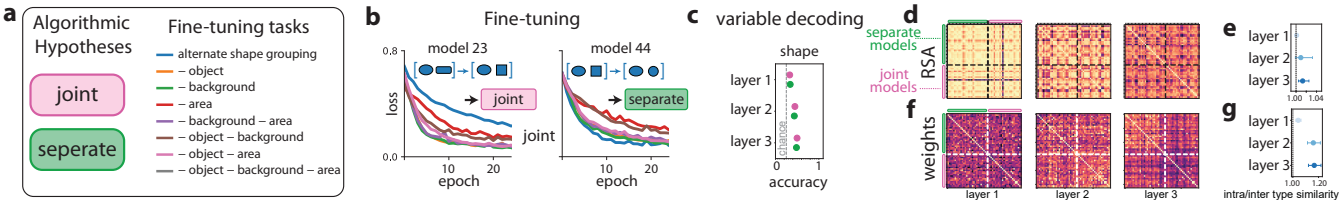

Figure 2: After training on the labels generated by equation (1), we fine-tuned networks on a series of labels generated by modifying (1) (**a, b**). We attributed a joint or separate detector depending on whether it learned the alternative grouping label faster or slower than the other labels. These algorithms were not evident in the decoding of intermediate variables (**c**) or their representational similarities (**d**) as evidenced by their within vs. between type similarity ratio (**e**), and only weakly reflected in the permutation-invariant cosine similarities between their weights (**f, g**).

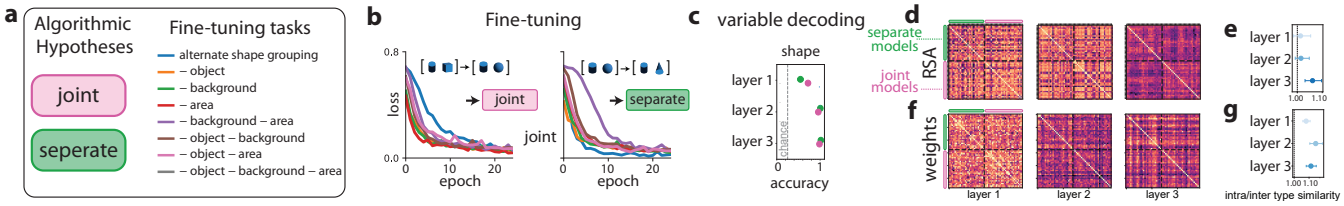

Figure 3: After training on the labels generated by equation (2), we fine-tuned networks on a series of labels generated by modifying (2) (**a, b**). We attributed a joint or separate detector depending on whether it learned the alternative grouping label faster or slower than the other labels. These algorithms were not evident in the decoding of intermediate variables (**c**) or their representational similarities (**d**) as evidenced by their within vs. between type similarity ratio (**e**), and only weakly reflected in the permutation-invariant cosine similarities between their weights (**f, g**).

detectors to 24 (see Fig. 2b).

How do the categorizations derived from transfer learning compare to input-output and parts-based approaches? We found that performance did not significantly correlate its category. We found that the RSA similarities within category did not significantly differ from the similarities across categories (see Fig. 2d,e). Next, we found that cosine similarities between the weights within each category differed only slightly from the similarities across categories, and that difference was small (see Fig. 2f,g), too small to reliably predict a network's category using a decoder or k-nearest neighbors. Finally, we found that the differences in decoding accuracy for individual shapes (square, circle, ellipse, capsule) were not significant (see Fig. 2c).

Third Experiment: 3D Classification

Our third experiment involved a three-dimensional shape categorization task. We used the 3D Shapes database (Burgess & Kim, 2018). It includes 4,800,00 images of colored three-dimensional shapes generated from six latent variables: `object shape` (spheres, cylinders, cubes, tori), `color` (object's color), `floor` (floor's color), `wall` (wall's color), `scale` (object's scale), and `orientation` (object's orientation). Similar to the last experiment, the shapes were divided into two groups, one assigned -1, the other assigned 1, and the other values were scaled between -1 and 1. We labeled all of the images using six different versions of the following:

$$1 \text{ iff } \texttt{shape} \cdot \Big[(\texttt{object} \times \texttt{background}) + (\texttt{floor} \times \texttt{area})\Big] > 0 \quad (2)$$

The six different versions corresponded to the six different ways of dividing the shapes into groups.

We used the same networks and procedures as the last experiment (see Fig. 3a). Our method distinguished between 20 networks with joint detectors and 36 networks with separate detectors (see Fig. 3b) Similar to before, these distinctions were not apparent through standard behavioral and parts-based analyses (see Fig. 3c-g).

Discussion

Across three experiments, we used the speed at which a network learns new tasks to identify its algorithm. We also compared our method to two other methods: 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 are 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, representational similarity of layer activations, similarity of the networks' weights, or by the ability to decode putative parts of the algorithm within networks.

This leaves us with an interesting dilemma. On the one hand, we could regard parts-based methods as our most reliable guide to the "ground truth" about a network's algorithm. According to our current parts-based methods, transfer learning attributes algorithms that we have no reason to believe are implemented. Thus, given current methods, we have no reason to believe that transfer learning is a reliable method for identifying a network's algorithm. But that leaves open

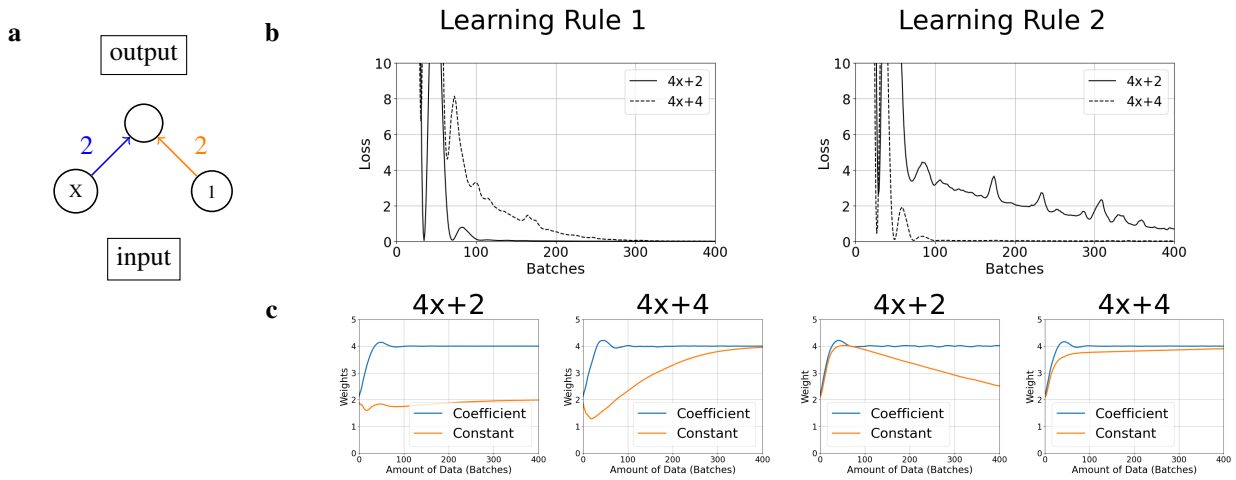

Figure 4: **a** Parts-based approaches attribute `2 · x + 2` to this handcrafted network regardless of its learning rule. But transfer learning approaches attribute `2 · x + 2` for some learning rules and `2 · (x + 1)` for other learning rules. **b** On the left ("Learning Rule 1"), its weights were adjusted using gradient descent to minimize mean squared error. On the right ("Learning Rule 2"), its weights were adjusted using gradient descent to minimize a combination of mean squared error and the absolute distance between its weights ("soft weight sharing"). Given Learning Rule 1, it was faster at learning the mappings $x \mapsto 4x+2$, etc. Given Learning Rule 2, it was faster at learning the mappings $x \mapsto 4x+4$, etc. This asymmetry is illustrated by its learning speeds for $4x+2$ and $4x+4$. The transfer approach therefore attributes `2 · x + 2` given Learning Rule 1 and `2 · (x + 1)` given Learning Rule 2. **c** These plots depict how the network's weights were adjusted during learning for each learning rule. Because the network is so simple, the adjustments are easy to interpret. For networks that are even slightly more complex, there is no obvious interpretation.

the possibility that future parts-based methods might vindicate transfer learning. Perhaps they will discover parts corresponding to the algorithms attributed by transfer learning. For example, our three experiments demonstrate that standard metrics for activity and weight similarity, such as cosine similarity, do not reveal the same algorithmic categories as transfer learning. But it might be possible to work backwards from the algorithms attributed through transfer learning to non-standard metrics that reveal parts corresponding to those algorithms, at least in those experiments.

On the other hand, we could regard transfer learning as by itself providing the ground truth about a network's algorithm. That is, we might decide that, even when transfer learning gives us a perspective that differs from the traditional parts-based perspective, it is still valuable, and should still be used to attribute algorithms. From this transfer learning perspective, we could still search for parts corresponding to the steps of an algorithm, the difference is that a network could implement an algorithm even if there weren't any corresponding parts. Given that the traditional parts-based perspective does not seem to attribute algorithms to many networks, including the networks in our experiments, this might lead to a general preference for the transfer learning perspective. Significantly, this perspectives would require us to rethink what it is for neural networks to implement algorithms, and therefore how cognitive science relates to systems neuroscience (Morrison, 2025).

This dilemma is even sharper for small, handcrafted networks that have obvious parts-based interpretations. Consider the network in Figure 4. Which algorithm should we attribute? From a parts-based perspective, we should attribute `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 and learning rule. From a transfer learning perspective, however, we should attribute `2 · x + 2` for the first learning rule and `2 · (x + 1)` for the second learning rule. From this perspective, a network's algorithm depends on its learning rule.

One limitation is worth noting. If a network can't learn, we cannot use the transfer learning approach to attribute an algorithm to it. For example, if the network responsible for low-level vision is fixed, we would need to use one of the other approaches to attribute an algorithm. There might also be practical reasons to prefer other approaches. For example, the inner workings of our largest language model might require too much fine-tuning data for this approach to be tractable.

Conclusion

We believe that transfer learning provides a productive framework for attributing algorithms to brains and artificial neural networks. In ongoing research, we are extending this approach to other networks and tasks. We are also developing a general framework for measuring the distances between algorithms.

References

Ashida, G., & Carr, C. E. (2011). Sound localization: Jeffress and beyond. *Current Opinion in Neurobiology*, *21*(5), 745-751. (Networks, circuits and computation) doi: https://doi.org/10.1016/j.conb.2011.05.008

Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. by the late Rev. Mr. Bayes, F.R.S. Communicated by Mr. Price, in a Letter to John Canton, A.M. F.R.S. *Philosophical Transactions of the Royal Society of London*, *53*, 370–418. doi: 10.1098/rstl.1763.0053

Burgess, C., & Kim, H. (2018). *3d shapes dataset.* https://github.com/deepmind/3dshapes-dataset/.

Chang, L., & Tsao, D. Y. (2017). The code for facial identity in the primate brain. *Cell*, *169*(6), 1013–1028.

Chun, M., & Most, S. (2022). *Cognition*. Oxford University Press. Retrieved from `https://books.google.com/books?id=GSdIzgEACAAJ`

Churchland, P. M. (1989). *A neurocomputational perspective: The nature of mind and the structure of science*. The MIT Press.

Churchland, P. S. (1986). *Neurophilosophy: Toward a unified science of the mind-brain*. The MIT Press.

Cormier, S. M., & Hagman, J. D. (1987). *Transfer of learning: Contemporary research and applications*. Academic Press.

Davies, M. (1987). Tacit knowledge and semantic theory: Can a five percent difference matter? *Mind*, *96*(October), 441–62.

De Lillo, C., Floreano, D., & Antinucci, F. (2001). Transitive choices by a simple, fully connected, backpropagation neural network: Implications for the comparative study of transitive inference. *Animal Cognition*, *4*, 61-68.

Frege, G. (1893/1903). *Grundgesetze der arithmetik* (Vols. I–II; P. Ebert, M. Rossberg, & C. Wright, Trans.). Jena: Verlag Hermann Pohle.

Geiger, A., Wu, Z., Potts, C., Icard, T., & Goodman, N. D. (2023). *Finding alignments between interpretable causal variables and distributed neural representations.* (arXiv:2303.02536)

Griffiths, T. L., Chater, N., Kemp, C., Perfors, A., & Tenenbaum, J. B. (2010, aug). Probabilistic models of cognition: exploring representations and inductive biases. *Trends in Cognitive Sciences*, *14*(8), 357–364. (PMID: 20576465) doi: 10.1016/j.tics.2010.05.004

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

Kay, K., Biderman, N., Khajeh, R., Beiran, M., Cueva, C. J., Shohamy, D., ... Abbott, L. (2023). Emergent neural dynamics and geometry for generalization in a transitive inference task. *bioRxiv*. doi: 10.1101/2022.10.10.511448

Kim, S. S., Rouault, H., Druckmann, S., & Jayaraman, V. (2017). Ring attractor dynamics in the drosophila central brain. *Science*, *356*(6340), 849–853.

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

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

Lillicrap, T. P., & Kording, K. P. (2019). *What does it mean to understand a neural network?* (arXiv:1907.06374)

Lindsay, G. W., & Bau, D. (2023). Testing methods of neural systems understanding. *Cognitive Systems Research*, *82*, 101156.

Lippl, S., Kay, K., Jensen, G., Ferrera, V. P., & Abbott, L. (2023). A mathematical theory of relational generalization in transitive inference. *bioRxiv*. doi: 10.1101/2023.08.22.554287

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

Marcum, J. I. (1947). *A statistical theory of target detection by pulsed radar*. Santa Monica, CA: RAND Corporation.

Mathys, C., Daunizeau, J., Friston, K. J., & Stephan, K. E. (2011). A bayesian foundation for individual learning under uncertainty. *Frontiers in human neuroscience*, *5*, 39.

Morrison, J. (2025). *Algorithms for neural networks.* (Unpublished manuscript)

Nanda, N., Chan, L., Lieberum, T., Smith, J., & Steinhardt, J. (2023). *Progress measures for grokking via mechanistic interpretability.* Retrieved from `https://arxiv.org/abs/2301.05217`

Niv, Y. (2021, oct). The primacy of behavioral research for understanding the brain. *Behavioral Neuroscience*, *135*(5), 601–609. (PMID: 34096743) doi: 10.1037/bne0000471

Orhan, A. E., & Ma, W. J. (2017). Efficient probabilistic inference in generic neural networks trained with non-probabilistic feedback. *Nature communications*, *8*(1), 138.

Ramsey, W. M. (2007). *Representation reconsidered*. Cambridge University Press.

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

Saxe, A. M., McClelland, J. L., & Ganguli, S. (2014). *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks.* Retrieved from `https://arxiv.org/abs/1312.6120`

Shadlen, M., & Kandel, E. (2021). Decision-making and consciousness. In *Principles of neural science* (6th ed., pp. 1392–1420). McGraw Hill.

Shannon, C. E. (1950). Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(314), 256–275. doi: 10.1080/14786445008521796

Thrun, S., & Pratt, L. (2012). *Learning to learn*. Springer Science & Business Media.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, *55*(4), 189–208. doi: 10.1037/h0061626

Vafaeikia, P., Namdar, K., & Khalvati, F. (2020). *A brief review of deep multi-task learning and auxiliary task learning.* (arXiv:2007.01126)

von Neumann, J. (1945, June 30). *First draft of a report on the EDVAC* (Tech. Rep.). Philadelphia, Pennsylvania: Moore School of Electrical Engineering, University of Pennsylvania. (Contract No. W-670-ORD-4926 between the United States Army Ordnance Department and the University of Pennsylvania)

Watkins, C. J. C. H., & Dayan, P. (1992). *q*-learning. *Machine Learning*, *8*, 279–292. Retrieved from https://doi.org/10.1007/BF00992698 doi: 10.1007/BF00992698

Zhou, C., Li, Q., Li, C., Yu, J., Liu, Y., Wang, G., ... Sun, L. (2023). *A comprehensive survey on pretrained foundation models: A history from BERT to ChatGPT.* (arXiv:2302.09419)