General-Purpose In-Context Learning by Meta-Learning Transformers

Louis Kirsch[1 2], **James Harrison**[1], **Jascha Sohl-Dickstein**[1], **Luke Metz**[1]
[1]Google Research, Brain Team [2]The Swiss AI Lab IDSIA, USI, SUPSI
`louis@idsia.ch, {jamesharrison,jaschasd,lmetz}@google.com`

Abstract

Modern machine learning requires system designers to specify aspects of the learning pipeline, such as losses, architectures, and optimizers. Meta-learning, or learning-to-learn, instead aims to *learn* those aspects, and promises to unlock greater capabilities with less manual effort. One particularly ambitious goal of meta-learning is to train general-purpose learning algorithms from scratch, using only black box models with *minimal inductive bias*. Such a model takes in training data, and produces test-set predictions, without any explicit definition of an inference model, training loss, or optimization algorithm. In this paper we show that Transformers and other black-box models can be meta-trained to act as general-purpose in-context learners. We characterize phase transitions between algorithms that generalize, algorithms that memorize, and algorithms that fail to meta-train at all, induced by changes in model size, number of tasks, and meta-optimization. We further show that the capabilities of meta-trained algorithms are bottlenecked by the accessible state size (memory) determining the next prediction, unlike standard models which are thought to be bottlenecked by parameter count.

Meta-learning is the process of automatically *discovering new learning algorithms* instead of designing them manually [1]. An important quality of human-engineered learning algorithms is their applicability to a wide range of tasks or environments. For learning-to-learn to exceed those capabilities, the meta-learned learning algorithms must be similarily *general-purpose*. Recently, there has been significant progress toward this goal [2–7]. The improved generality of the discovered learning algorithms has been achieved by introducing inductive bias, such as by bottlenecking the architecture or by hiding information, which encourages learning over memorization.

While enabling generalization, these inductive biases come at the cost of increasing the effort to design these systems and potentially restrict the space of discoverable learning algorithms. Instead, we seek to explore general-purpose meta-learning systems with *minimal inductive bias*. Good candidates for this are black-box sequence-models as meta-learners such as LSTMs [8–11] or Transformers [12, 13]. These models take in training data and produce test-set predictions without any explicit definition of an inference model, training loss, or optimization algorithm. This is known as memory-based or in-context learning.

In this work, we investigate how such black-box meta-learners can be trained to (meta-)generalize and learn on significantly different datasets than used during meta-training. For this we propose a Transformer-based *General-Purpose In-Context Learner* (GPICL). We characterize transitions—induced by scaling the number of tasks or the model size used for meta-training—between memorization, learning, and generalization. We further show that the capabilities of meta-trained algorithms are bottlenecked by their accessible state size (memory) determining the next prediction (such as the hidden state size in a recurrent network), unlike standard models which are thought to be bottlenecked by parameter count.

What is a (supervised) learning algorithm? In this paper, we focus on the setting of meta-learning supervised learning algorithms. Consider a mapping

$$\left(\{x_i, y_i\}_{i=1}^{N_D}, x'\right) \mapsto y' \tag{1}$$

from the training (support) set $D = \{x_i, y_i\}_{i=1}^{N_D}$ and a query input x' to the query's prediction y' where $x_i, x' \in \mathbb{R}^{N_x}$, $y_i, y' \in \mathbb{R}^{N_y}$ and $N_D, N_x, N_y \in \mathbb{N}^+$. The subset of these functions that qualify as learning algorithms are those that improve their predictions y' given an increasingly larger training set D. Meta-learning then corresponds to finding these functions via meta-optimization. As in other black-box meta-learning models, we use a neural network to represent such functions, for example based on RNNs [8], memory-augmented NNs [14], or Transformers [15].

What is a general-purpose learning algorithm? We focus in particular on obtaining general-purpose learning algorithms. A learning algorithm can be considered general-purpose if it learns on a wide range of possible tasks D and their respective related queries x', y'. For example, gradient-descent on a suitable loss function can be considered a very general-purpose human-engineered learning algorithm (where the gradient is obtained via backpropagation or other means).

1 General-Purpose In-Context Learning Algorithms with Transformers

Due to the small number of inductive biases in black-box models, we can only expect (meta-)generalization when meta-training with an appropriately broad data distribution. Thus, changes in the data distribution affect whether and how a model meta-learns and meta-generalizes. We classify algorithms along two different dimensions: To what extent it learns (improving predictions given increasingly larger training sets), and to what extent it generalizes (performs well on instances, tasks, or datasets not seen before). Algorithms can then be categorized as follows:

Learning	Generalization	Algorithm Description
No	No	Instance memorization
Yes	No	System identification
No	Yes	Zero-shot generalization
Yes	Yes	General-purpose learning algorithm

We demonstrate that sharp phase transitions occur between these learning modalities, and empirically investigate these transitions.

Generating tasks for learning-to-learn In this work, we augment existing datasets, in effect increasing the breadth of the task distribution based on existing task regularities. We do so by randomly projecting inputs and permuting classification labels. While the random projection removes spatial structure from the inputs, this structure is not believed to be central to the task (for instance, the performance of SGD-trained fully connected networks is invariant to projection by a random orthogonal matrix [16]). Task augmentation allows us to investigate fundamental questions about learning-to-learn in the regime of many tasks without relying on huge amounts of existing tasks or elaborate schemes to generate those. A task or dataset D is then defined by its corresponding base dataset $\bar{D} = \{\bar{x}_i, \bar{y}_i\}$, (linear) projection $A \in \mathbb{R}^{N_x \times N_x}$, with $A_{ij} \sim \mathcal{N}(0, \frac{1}{N_x})$, and output permutation ρ, $D = \{A\bar{x}_i, \rho(\bar{y}_i)\}$.

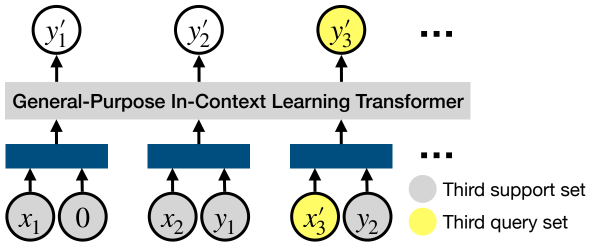

Figure 1: Our *General-Purpose In-Context Learner* (GPICL) is based on the vanilla Transformer which is trained to make predictions for queries x' given any prefix of a dataset $D := \{x_i, y_i\}_{i=1}^{N_D}$ as in Equation 2.

Meta-learning Given those generated tasks, we then meta-train jointly on a mini-batch sampled from the whole distribution. We minimize $J(\theta)$, the sum of losses on the query prediction after observing any prefix of a dataset D sampled from the augmented task distribution $p(D)$

$$J(\theta) = \mathbb{E}_{D \sim p(D)} \left[\sum_{j=1}^{N_D} l(f_\theta(D_{1:j-1}, x_j), y_j) \right], \tag{2}$$

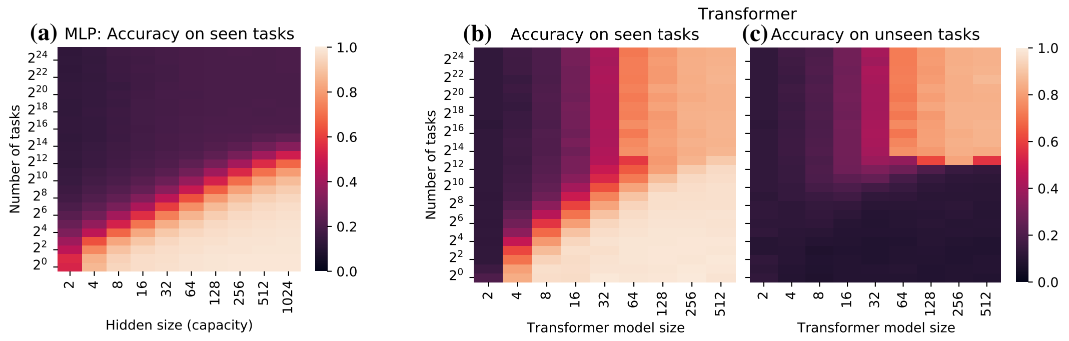

Figure 2: **GPICL is able to generalize to unseen tasks.** Each cell is a separate meta-training run. **(a)** An MLP classifier trained in a multi-task fashion across various numbers of tasks (generated based on MNIST) and network sizes is able to fit linearly more tasks, the larger its capacity. **(b)** A sequence model (here the GPICL Transformer) that observes a dataset D of inputs and labels transitions into generalizing to an seemingly unbounded number of tasks with an increase in model size. This is achieved by switching from a memorization solution to a learning solution that **(c)** generalizes to unseen tasks. This generalization does not occur with the MLP.

where in the classification setting, l is the cross entropy loss between the label y_j and prediction $y' = f_\theta(D_{1:j-1}, x_j)$, f_θ is a neural network mapping to predictions y' as in Equation 1. During meta-training, we take gradient steps in $J(\theta)$ by backpropagation and Adam [17]. To investigate the effect of the data distribution, we train on various numbers of tasks (1). Finally, we need to choose a black-box model for the function f_θ. We use a vanilla Transformer [12] with learned positional embeddings, visualized in Figure 1. We call it the *General-Purpose In-Context Learner* (GPICL). Each token corresponds to a concatenated and transformed input x_i and one-hot encoded label y_{i-1} predicting the corresponding logits $y' = y_i$ for the current input $x' = x_i$.

Algorithm 1 Meta-Training for General-Purpose In-Context Learning (GPICL)

```
Require: Base dataset $\bar{D} = \{\bar{x}_i, \bar{y}_i\}$, Number of tasks $K \in \mathbb{N}^+$
  $\{A_{ij}^{(k)}\}_{k=1}^{K} \sim \mathcal{N}(0, \frac{1}{N_x})$          ▷ Sample input projections
  $\{\rho^{(k)}\}_{k=1}^{K} \sim p(\rho)$          ▷ Sample output permutations
  $D^{(k)} = \{A^{(k)}\bar{x}_i, \rho^{(k)}(\bar{y}_i)\}$
  $p(D) := \text{Uniform}[\{D^{(k)}\}_{k=1}^{K}]$
  while not converged do
      $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$          ▷ Meta-train across tasks $p(D)$ (Equation 2)
```

Meta-testing At meta-test time, no gradient-based learning is used. Instead, we simply obtain a prediction y' by evaluating the neural network f_θ on the training dataset D and query point x'.

2 Experiments on the Emergence of General Learning-To-Learn

Multi-task training with standard classifiers Given a task distribution of many different classification tasks, we first ask under what conditions we expect "learning-to-learn" to emerge. We train a single model across many tasks where each task corresponds to a random transformation of the MNIST dataset, but where the MLP only receives a single datapoint instead of a whole sequence as input. This corresponds to $N_D = 1$ in Equation 2. We would expect such a non-sequential classifier to be able to correctly predict for more tasks as its number of parameters increases. When plotting the network capacity against the number of tasks, we indeed observe a linear boundary where an increasing number of tasks can be fit the larger the network (Figure 2a). This is consistent with results in Collins et al. [18], which found that a constant number of bits about the data distribution can be stored per model parameter, across a variety of model architectures and scales.

Learning-to-learn with large sequential models and data In contrast to the MLP classifier, a sequence model that observes multiple observations and their labels from the same task, could exceed that linear performance improvement by learning at inference time. Indeed, we observe that when switching to a Transformer that can observe a sequence of datapoints before making a prediction about the query, more tasks can be simultaneously fit (Figure 2b). At a certain model size and number of tasks, the model undergoes a phase transition, allowing to generalize to a seemingly unbounded number of tasks. We hypothesize that this is due to switching the prediction strategy from memorization to learning-to-learn. Further, when (meta-)testing the same trained models from the previous experiment on an unseen task (new random transformation of MNIST), they generalize only in the regime of large numbers of tasks and model size (Figure 2c). As with all black-box learners, meta-testing does not involve any gradient updates but only running the model in forward mode.

Simple invariances in data lead to the emergence of learning-to-learn To verify whether the observed generalizing solutions actually implement learning algorithms (opposed to e.g. zero-shot generalization), we analyze the meta-test time behavior. We plot the accuracy for a given query point given varying numbers of seen examples in Figure 3. As it is typical for learning algorithms, the performance improves given an increasingly large set of seen examples (inputs and labels).

Generalization Naturally, the question arises to what extent these learning algorithms are general. While we have seen generalization to unseen tasks consisting of novel projections of the same dataset, do the learned algorithms also generalize to unseen datasets? In Figure 3 we observe out-of-distribution performance on Fashion MNIST after having trained on MNIST (b, blue). In this direction, there is no generalization gap to directly training on Fashion MNIST (b, orange). Similarly, when meta training on Fashion MNIST and meta testing on MNIST (a, orange) we observe that the learning algorithm generalizes, albeit with a larger generalization gap.

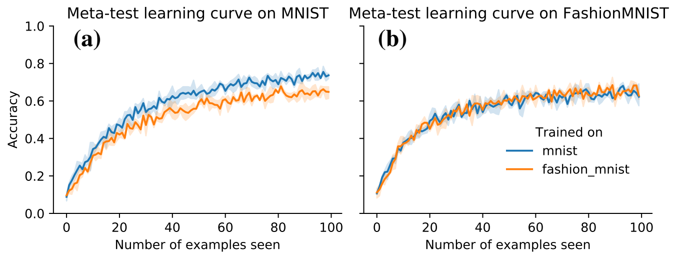

Figure 3: **GPICL learns from examples at test time, and generalizes to unseen tasks and datasets.** We meta-trained the Transformer on a set of tasks defined by random transformations of either MNIST (blue) or FashionMNIST (orange). We then meta-test on unseen tasks, and seen (ab) or unseen (ba) datasets. The plot shows the accuracy averaged across multiple runs at each inner step, with shading indicating 95% confidence intervals. The increase in performance at each step suggests we have discovered a learning algorithm.

Comparison to other methods Other datasets and baselines are shown in Table 1. In particular, rather than focusing on SOTA, we aim to validate whether methods with less inductive bias (such as our GPICL), can compete with methods that include more biases suitable to learning-to-learn. This includes stochastic gradient descent (SGD) that updates the parameters online after observing each datapoint. MAML [19] proceeds like SGD, but uses a meta-learned neural network initialization. Both methods that rely on backpropagation and gradient descent, learn more slowly compared to our Transformer. In the case of MAML, this may be due to the main mechanism being feature reuse [20] which is less useful when training across our wider task distribution. Among methods that do not hard-code gradient descent at meta-test time, we test VSML [6] that discovered learning algorithms significantly generalizing between tasks. Our GPICL comes surprisingly close to VSML without requiring the associated inductive bias. Finally, we compare to a standard LSTM that is trained with the same inputs as our Transformer. We observe that it performs worse, which we investigate further.

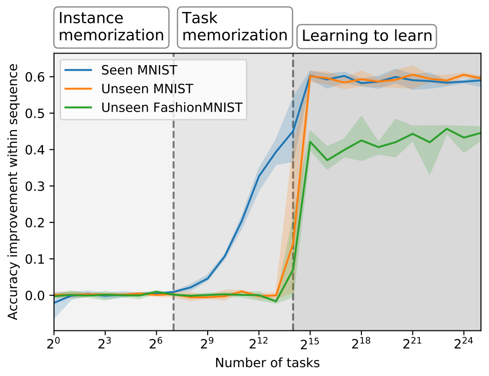

Figure 4: **Transformers exhibit three different phases in terms of meta-learned behavior.** (1) When training on a small number of tasks, specific instances are memorized. (2) Tasks are memorized, which is evident as a within-sequence increase of performance. (3) When training across many tasks, we discover a learning algorithm that generalizes to unseen tasks and unseen datasets.

Transitioning from memorization to learning to generalizing When do the found solu-

Table 1: Meta-test generalization to various datasets after meta-training on augmented MNIST and seeing 99 examples, predicting the 100th. We report the mean across 3 meta-training seeds, 16 sequences from each task, 16 tasks sampled from each base dataset.

Method / Dataset	Inductive bias	MNIST	Fashion MNIST	KMNIST	Random	CIFAR10	SVHN
SGD	Backprop, SGD	70.31%	50.78%	37.89%	100.00%	14.84%	10.16%
MAML	Backprop, SGD	53.71%	48.44%	36.33%	99.80%	17.38%	11.33%
VSML	Parameter sharing, Permutation invariance	79.04%	68.49%	54.69%	100.00%	24.09%	17.45%
LSTM	black-box	25.39%	28.12%	18.10%	58.72%	12.11%	11.07%
GPICL Transformer (ours)	black-box	73.70%	62.24%	53.39%	100.00%	19.40%	14.58%

tions correspond to memorizing, learning, and generalizing solutions? In Figure 4, we plot the accuracy difference between the last and first prediction for a seen task, an unseen task, and an unseen task with a different base dataset. We observe three phases: In the first phase, we memorize each instance, resulting in no within-sequence performance improvement. In the second phase, we memorize tasks and learn to identify tasks, resulting in a within-sequence improvement confined to seen task instances. In the final and third phase, we observe a more general learning-to-learn, a performance improvement for unseen tasks, even different base datasets (here FashionMNIST).

Architecture: A large state is crucial for learning In the previous experiments we observed that given sufficient task diversity and model size, Transformers can learn general-purpose learning algorithms. This raises the question how essential the Transformer architecture is and whether other black-box models could be used. We hypothesize that for learning-to-learn the size of the memory at meta-test time (or state more generally) is particularly important in order to be able to store learning progress. Through self-attention, Transformers have a particularly large state. We test this by training several architectures with various state sizes in our meta-learning setting. In Figure 5a, we observe that when we vary the respective hyper-parameters which most influence the state size, we observe that for a specific state size we obtain similar performance of the discovered learning algorithm across architectures. In contrast, these architectures have markedly different numbers of parameters (Figure 5b). This suggests that the model size in terms of numbers of parameters plays a smaller role in the setting of learning-to-learn and Transformers have benefited in particular from an increase in state size by self-attention. Beyond learning-to-learn, this likely applies to other tasks that rely on storing large amounts of sequence-specific information.

Discussion and Conclusion By generating tasks from existing datasets, we demonstrated that black-box models such as Transformers can be used to meta-learn general-purpose in-context learning algorithms (GPICL). Compared to previous work, this can be done without having to put strong inductive biases in the inner learning algorithm or architecture. We observed that learning-to-learn arises in the regime of large models and large numbers of tasks with several phase transitions from instance memorization, to system identification, to general learning. The size of the memory or model state significantly determines how well any architecture can learn how to learn across various neural network architectures. For additional experiments, limitations, and related work, see appendix. In the appendix, we also identified difficulties in meta-optimization and proposed interventions in terms of optimizers, hyper-parameters, and a biased data distribution acting as a curriculum. We believe our findings open up new possibilities of data-driven general-purpose meta-learning with minimal inductive bias.

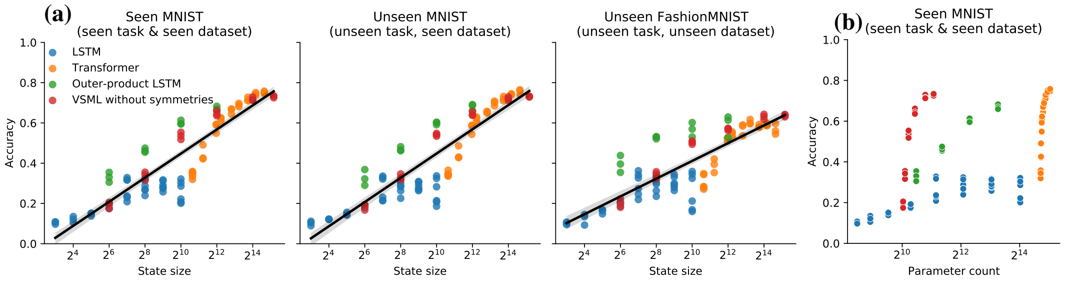

Figure 5: **The state size (accessible memory) of an architecture most strongly predicts its performance as a general-purpose learning algorithm.** **(a)** A large state is crucial for learning-to-learn to emerge. **(b)** The parameter count correlates less well with learning capabilities.

References

[1] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[2] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[3] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

[4] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020.

[5] John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Quoc V Le, Sergey Levine, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021.

[6] Louis Kirsch and Jürgen Schmidhuber. Meta learning backpropagation and improving it. *arXiv preprint arXiv:2012.14905*, 2020.

[7] Louis Kirsch, Sebastian Flennerhag, Hado van Hasselt, Abram Friesen, Junhyuk Oh, and Yutian Chen. Introducing symmetries to black box meta reinforcement learning. *arXiv preprint arXiv:2109.10781*, 2021.

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

[9] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[10] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[11] Vladimir Mikulik, Grégoire Delétang, Tom McGrath, Tim Genewein, Miljan Martic, Shane Legg, and Pedro Ortega. Meta-trained agents implement bayes-optimal agents. *Advances in neural information processing systems*, 33:18691–18703, 2020.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[13] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[14] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[15] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022.

[16] Neha Wadia, Daniel Duckworth, Samuel S Schoenholz, Ethan Dyer, and Jascha Sohl-Dickstein. Whitening and second order optimization both make information in the dataset unusable during training, and can reduce or prevent generalization. In *International Conference on Machine Learning*, pages 10617–10629. PMLR, 2021.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

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

[20] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2020.

[21] Yann LeCun, Corinna Cortes, and C J Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.0, 2017.

[23] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep Learning for Classical Japanese Literature, 2018.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[26] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[27] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[28] Joachim Winther Pedersen and Sebastian Risi. Evolving and merging hebbian learning rules: increasing generalization by decreasing the number of rules. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 892–900, 2021.

[29] Sebastian Risi. The future of artificial intelligence is self-organizing and self-assembling. *sebastianrisi.com*, 2021. URL `https://sebastianrisi.com/self_assembling_ai`.

[30] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[31] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[32] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565. PMLR, 2019.

[33] Luke Metz, Niru Maheswaranathan, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020.

[34] Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[35] Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*, 2020.

[36] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, et al. Towards learning universal hyperparameter optimizers with transformers. *arXiv preprint arXiv:2205.13320*, 2022.

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

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

[39] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[40] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *Advances in Neural Information Processing Systems*, 34:7703–7717, 2021.

[41] Mark Sandler, Max Vladymyrov, Andrey Zhmoginov, Nolan Miller, Andrew Jackson, Tom Madams, et al. Meta-learning bidirectional update rules. *arXiv preprint arXiv:2104.04657*, 2021.

[42] Louis Kirsch and Jürgen Schmidhuber. Self-referential meta learning. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*, 2022.

[43] Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In *International Conference on Machine Learning*, pages 27075–27098. PMLR, 2022.

[44] Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 22574–22587, 2021.

[45] Joachim Winther Pedersen and Sebastian Risi. Minimal neural network models for permutation invariant agents. *arXiv preprint arXiv:2205.07868*, 2022.

[46] Pedro A Ortega, Jane X Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alex Pritzel, Pablo Sprechmann, et al. Meta-learning of sequential strategies. *arXiv preprint arXiv:1905.03030*, 2019.

[47] Jürgen Schmidhuber. A 'self-referential'weight matrix. In *International conference on artificial neural networks*, pages 446–450. Springer, 1993.

[48] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[49] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[50] Stephanie CY Chan, Adam Santoro, Andrew K Lampinen, Jane X Wang, Aaditya Singh, Pierre H Richemond, Jay McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *arXiv preprint arXiv:2205.05055*, 2022.

[51] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *arXiv preprint arXiv:2208.01066*, 2022.

[52] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code or code snippet will be released at a later date.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix Section A.8
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section A.9
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use the datasets in Table 1 [21–25]
   (b) Did you mention the license of the assets? [Yes] See respective prior publication for the license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Classifying Algorithms: Learning & Generalization

We can classify algorithms along two different dimensions: To what extent it learns (improving predictions given increasingly larger training sets), and to what extent it generalizes (performs well on instances, tasks, or datasets not seen before). Algorithms can then be categorized as follows:

Learning	Generalization	Algorithm Description
No	No	Instance memorization
Yes	No	System identification
No	Yes	Zero-shot generalization
Yes	Yes	General-purpose learning algorithm

A.2 Solutions are memorizing or generalizing

When do the found solutions correspond to memorizing vs generalizing solutions? In Figure 2 we observe a fairly discrete transition between memorizing and generalizing solutions as a function of the number of tasks. To analyze this transition, we perform multiple training runs with varying seeds and numbers of tasks in Figure 6, reporting the final training loss. We find that the distribution is bi-modal. Solutions at the end of training are memorizing or generalizing. Memorization cluster: The larger the number of tasks, the more difficult it is to memorize all of them with a fixed model capacity. Generalization cluster: At a certain number of tasks (here 6 thousand), a transition point is reached where optimization sometimes discovers a lower training loss that corresponds to a generalizing solution. For larger numbers of tasks the solutions always settle in the generalizing cluster.

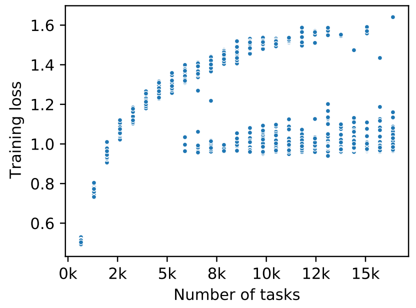

Figure 6: **Solutions found by GPICL after meta-training are bi-modal, with a memorization and generalization mode.** Each point represents the training loss at the end of meta-training for runs with different seeds and for various numbers of tasks that include the transition boundary previously observed. Almost all solutions are either in a memorization cluster or in a generalization cluster.

A.3 What corresponds to state (memory) in various architectures?

We hypothesize that for learning-to-learn the size of the memory N_S at meta-test time (or state more generally) is particularly important in order to be able to store learning progress. We test this by training several architectures with various N_S in our meta-learning setting. Memory in the context of recurrent neural networks corresponds to the hidden state or context vector of size N_H, thus $N_S \in \mathcal{O}(N_H)$. More generally, we can describe the state as the information bottleneck that the sequence has to pass through before making predictions. In the context of learning-to-learn, this state has to hold information about everything that has been learned so far. Standard learning algorithms such as neural networks trained via SGD would have a state that corresponds to the neural network parameters, iteratively updated via SGD. In transformers, self-attention allows for a particularly large state of $N_S \in \mathcal{O}(N_K N_L N_T)$ where N_K is the size of key, value, and query, N_L is the number of layers, and N_T is the length of the sequence.

A.4 Summary of Insights

Insight 1: It is possible to learn-to-learn with black-box models Effective learning algorithms can be realized using black-box models with few inductive biases, given sufficient meta-training task diversity and large enough model sizes. To transition to the learning-to-learn regime, we needed at least $2^{13} = 8192$ tasks.

Insight 2: Simple data augmentations are effective for general learning-to-learn The generality of the discovered learning algorithm can be controlled via the data distribution. Even when large task distributions are not (yet) naturally available, simple augmentations that promote permutation and scale invariance are effective.

Insight 3: The meta-learned behavior has phase transitions When increasing the number of tasks, the meta-learned behavior transitions from instance memorization, to task identification, to general learning-to-learn. The last transition is discrete, with two unique clusters.

Insight 4: Large state is more crucial than parameter count We conclude that the specific inductive biases of each architecture matter to a smaller degree. The driving factor behind their ability to learn how to learn is the size of their state. Furthermore, this suggests that the model size in terms of numbers of parameters plays a smaller role in the setting of learning-to-learn and Transformers have benefited in particular from an increase in state size by self-attention. In non-meta-learning sequence tasks parameter count is thought to be the performance bottleneck [18]. Beyond learning-to-learn, this likely applies to other tasks that rely on processing and storing large amounts of sequence-specific information.

A.5 Challenges in Meta-Optimization

Meta-optimization is known to be challenging. Meta gradients [19, 26, 27] are unstable for long inner loops consisting of many gradient updates. Works relying on parameter-sharing or weight updates in their architecture [6, 28, 29] observed various difficulties: Slower convergence, local minima, unstable training, or loss plateaus at the beginning of training. In VSML, in particular deeper architectures make meta-optimization more difficult (Figure 17). In this section, we show that some of these problems also occur with black-box models and propose effective interventions.

Loss plateaus when meta-learning with black-box models By training across a large number of randomly transformed tasks, memorizing any task-specific information is difficult. Instead, the model is forced to find solutions that are directly learning. We observe that this results in (meta-)loss plateaus during meta-training where the loss only decreases slightly for long periods of time (Figure 7a). Only after a large number of steps (here around 35 thousand) does a drop in loss occur.

In Figure 7b we zoom into the plateau. We observe that while in the plateau the training loss decreases only slightly, at the same time the generalization loss increases on unseen tasks from both the same and a different base dataset. This suggests that being able to first memorize slightly in the plateau enables the following learning-to-learn phase whereas directly optimizing for learning-to-learn is difficult. When analyzing the gradients in the loss plateau, we observe that all gradients have a very small norm with exception of the last layer (Figure 13).

Intervention 1: Increasing the batch size High variance gradients appear to be one reason training trajectories become trapped on the loss plateau (see Appendix Figures 11, 12). This suggests increasing the meta-batch size as a straightforward solution. When plotting various batch sizes against numbers of tasks we obtain three kinds of solutions at the end of meta-training (Figure 8a): (1) Solutions that generalize and learn, (2) Solutions that memorize, and (3) Solutions that are still in the loss plateau (due to maximum of 50 thousand optimization steps). The larger the batch size, the more tasks we can train on without getting stuck in a loss plateau. When plotting the length of the loss plateau against the task batch size (Figure 8b) we observe a power-law relationship with increasing batch sizes decreasing the plateau length. At the same time, the batch size also increases the number of total tasks seen in the plateau (Appendix Figure 14). Thus, this intervention relies on parallelizability. An increase in the number of tasks also increases the plateau length (Figure 8c). This may be due to a larger number of tasks making the initial memorization phase more difficult.

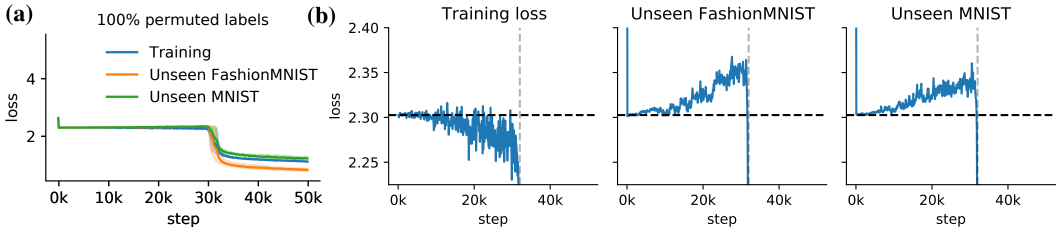

Figure 7: **Meta-training dynamics often involve an extended period where GPICL's performance is stuck on a plateau.** **(a)** Meta-loss vs. meta-training step, for a uniform distribution over meta-training tasks. Training tasks are generated by random transformations of FashionMNIST. **(b)** A zoomed in view of the plateau. The loss only decreases slightly and the model memorize small biases in the training data (decreasing generalization) before the loss drops sharply.

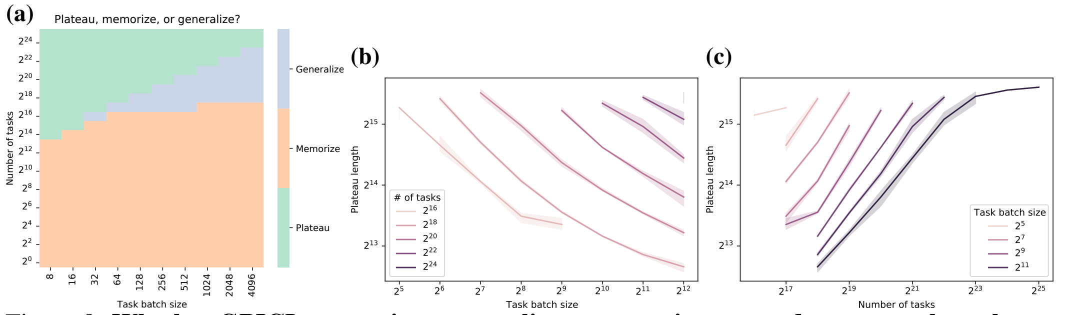

Figure 8: **Whether GPICL memorizes, generalizes, or remains trapped on a meta-loss plateau depends on the number of meta-training tasks, and the meta-training batch size. (a)** A phase diagram showing GPICL's behavior at the end of meta-training (50k steps). Solutions either memorize, generalize and learn, or remain in the loss plateau. With additional training steps, configurations in the plateau might eventually transition to memorization or generalization. Generalization only occurs with large enough batch sizes and sufficient, but not too many, tasks. **(b)** This behavior is explained by the plateau length decreasing with the increasing batch sizes (reducing the noise contribution), and **(c)** increasing with larger numbers of tasks.

Intervention 2: Changes in the meta-optimizer Can we address the loss plateau on an optimizer-level? Given that many gradients in the loss plateau have very small norm, Adam would rescale those element-wise, potentially alleviating the issue. In practice, we observe that the gradients are so small that the ϵ in Adam's gradient-rescaling denominator (for numerical stability) limits the up-scaling of small gradients. Instead of using the default ϵ of 10^{-8} we investigate 10^{-16} and smaller. While this doesn't alleviate the loss plateau entirely, it results in more than halving the plateau length (Appendix Figure 15). Alternatively, discarding the magnitude of the gradient entirely by applying the sign operator to an exponential moving average of the gradient (replacing Adam's approximate magnitude normalization with direct magnitude normalization) has a similar effect while also increasing the numerical stability over Adam with small ϵ (Figure 15).

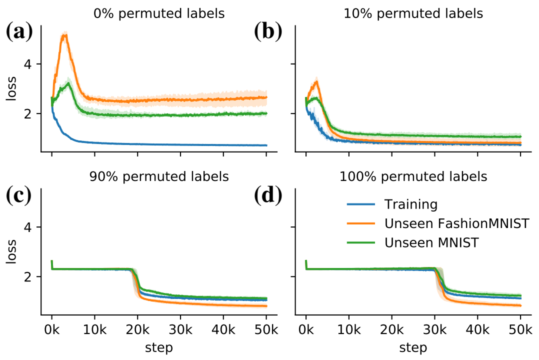

Figure 9: **Biasing the training distribution is an effective intervention which prevents a meta-loss plateau.** A uniform distribution over tasks leads to a long plateau **(d)**, while increasing the training fraction that corresponds to a single task reduces the plateau **(abc)**.

Intervention 3: Biasing the data distribution / Curricula The previous interventions may also be useful with optimization difficulties in methods that have a larger number of inductive biases in the learning algorithm and architecture. In this paper, due to the nature of using black-box models for the learning algorithm, we rely mainly on the data distribution to govern learning dynamics, learning-to-learn, and generalization. This enables a different intervention that relies on modifications to the data distribution. This is inspired by the observation that within the loss plateau, a decrease in loss goes with memorizing biases in the data distribution. We propose a simple remedy that enables initial memorization, is cheap, and simple to implement: biasing the data distribution. Instead of sampling label permutations uniformly at random, we bias towards a specific permutation by using a fixed permutation for a fraction of each batch. This completely eliminates the loss plateau, enabling a smooth path from memorizing to learning (Figure 9). Surprisingly, even when heavily biasing the distribution, memorization is followed by generalization. This biased data distribution can be viewed as a curriculum, essentially solving an easier problem first, a 'stepping stone', that enables solving the harder learning-to-learn problem. Further investigation is required to understand how this transition occurs. This may be connected to grokking [30] which we investigate in Appendix A.10. We hypothesize that many natural data distributions—including language—contain such sub-tasks that are easy to memorize followed by generalization.

A.6 Related work

Meta-learning inductive biases Meta-learning approaches exist with a wide range of inductive biases, usually inspired by existing human-engineered learning algorithms. Some methods pre-wire the entire learning algorithm [19], pre-wire backpropagation and the structure of a gradient-based optimizer [31–33], or hard-code gradient-based optimization but learn the loss function [2, 27, 34]. Many methods search over hyper-parameters that alter existing learning algorithms [26, 35, 36]. Fast weight programmers or hyper-networks update the weights of the same or another neural network [37–43]. Our work aims to keep such inductive biases to a minimum.

General-purpose meta-learning There has been growing interest in meta-learning more general-purpose learning algorithms. The improved generality of the discovered learning algorithm has been achieved by introducing inductive bias, such as by bottlenecking the architecture or by hiding information, encouraging learning over memorization. Methods include enforcing learning rules to use gradients [2, 3, 32], symbolic graphs [4, 5], or parameter sharing and symmetries [6, 7]. Parameter sharing and symmetries have also been discussed in the context of self-organization [29, 44, 45].

Black-box meta-learning: MetaRNNs, RL^2, in-context learning In contrast to these inductive biases, neural networks can also learn-to-learn purely in their activations with little architectural and algorithmic biases [8–10, 46]. This requires a feedback signal in the inputs that allows for learning such as the reward in reinforcement learning or label in supervised learning [47]. A frequently used architecture is the LSTM [48, 49]. Recently this mechanism has also been used with other architectures such as Transformers [13, 50] under the name of in-context learning. We refer to these networks simply as black-box meta learners. Our method GPICL is in the class of these black-box meta learners. In contrast to previous methods, GPICL implements *general-purpose* learning algorithms. Independently, Garg et al. [51] recently studied generalization on synthetic functions compared to our augmented datasets. PFNs [11] demonstrated learning to learn on small tabular datasets when meta-training on synthetically generated problems. Experiments on more complex classification settings such as Omniglot relied on fine-tuning. In comparison, our method investigated generalization of learning algorithms directly to datasets such as MNIST, Fashion MNIST, and CIFAR10.

A.7 Limitations

Varying input and output sizes Compared to some previous works in meta-learning [6, 19, 31], the discovered learning algorithms are not applicable to an arbitrary input and output size which makes it more difficult to apply the learning algorithm to a new, unseen problem. This problem also applies to Transformers applied to multiple tasks and modalities. Related work has solved this problem by tokenizing inputs to compatible, unified representations [52]. We expect these techniques or others to be useful in the learning-to-learn context too.

Processing large datasets Learning algorithms often process millions of inputs before outputting the final model. In the black-box setting, this is still difficult to achieve. Recurrency-based models usually suffer from accumulating errors, whereas Transformers computational complexity grows quadratically in the sequence length. Additional work is required to build models capable of processing and being trained on long sequences. Alternatively, parallel processing, similar to batching in learning algorithms, may be a useful building block.

A.8 Architectural Details and Hyper-parameters

Transformer details See Figure 1 for a visualization. When querying for the first x_1, there is no previous label, so we feed zeros. By default, all Transformers have a key, value, and query size of 32, 8 heads, and 4 layers, and model size of $N_M = 256$. The model size defines the dimensionality of each token, and the MLP between layers scales this size up to a hidden representation of $4 \times N_M$ where N_M corresponds to the model size.

Outer-product LSTM We slightly modify an LSTM by replacing the context state with an outer-product update and inner-product read-out.

```
x_and_h = jnp.concatenate([inputs, prev_state.hidden], axis=-1)

gated = hk.Linear(8 * size * self.num_heads)(x_and_h)
gated = gated.reshape((batch_size, self.num_heads, 8 * size))
gated = checkpoint_name(gated, 'gated')

# i = input, g = cell_gate, f = forget_gate,
# q = query, o = output_gate
sizes = (3 * size, 3 * size, size, size)
indices = np.cumsum(sizes[:-1])
k1, k2, q, o = jnp.split(gated, indices, axis=-1)
scale = jax.nn.softplus(
    hk.get_parameter('key_scale', shape=(), dtype=k1.dtype,
                     init=jnp.zeros))
i, g, f = jnp.einsum('bhki,bhkj->kbhij',
                     jax.nn.tanh(split_axis(k1, (3, size))) * scale,
                     jax.nn.tanh(split_axis(k2, (3, size))))
f = jax.nn.sigmoid(f + 1)  # Forget bias
c = f * prev_state.cell + jax.nn.sigmoid(i) * g
read = jnp.einsum('bhij,bhi->bhj', c, q)
h = hk.Flatten()(jax.nn.sigmoid(o) * jnp.tanh(read))
```

VSML We use a version of VSML with a single layer and self-messages [7] of size 8. Each LSTM has a hidden size of 16. For each LSTM update we use two micro-ticks. We train on 2^{25} tasks with a 90% biased permutation distribution. The task batch size is 8. All images are scaled to a size of $32 \times 32 \times 3$

VSML without symmetries Before activations are fed to a standard instantiation of VSML, all inputs are projected using a learnable linear projection. Logits are generated using another linear projection, followed by a softmax. We use a version of VSML with a single layer and self-messages [7] of size 8. The LSTMs are on a grid of $k \times k$ LSTMs, where $k \in \{1, 2, 4, 8, 16, 24\}$. Each LSTM has a hidden size of 64. For each LSTM update we use two micro-ticks. We train on 2^{25} tasks with a 90% biased permutation distribution. The task batch size is 128. All images are scaled to a size of 14×14.

LSTM For the results in Table 1, we used a hidden size of 256 and 10^5 optimization steps. Larger hidden sizes were harder to optimize. We train on 2^{25} tasks with a 90% biased permutation distribution. The task batch size is 128. All images are scaled to a size of $32 \times 32 \times 3$

A.9 Experimental Details

Most experiments can be run on a single GPU, some require 16 GPUs due to sequence length and large batch sizes, with sufficient GPU memory (around 16 GB each). Some experiments, such as Figure 2, require up to 1000 runs of that kind to produce the final heat-map.

Input normalization Each dataset is z-normalized by its mean and standard deviation across all examples and pixels.

Number of seeds and shading If not noted otherwise, line plots use 8 seeds for meta-training and at least 512 seeds for meta-testing. Shading indicates 95% confidence intervals.

Figure 2 The MLP has two hidden layers of varying size with relu activations. The Transformer has the default parameters as defined above.

Figure 3 We use a transformer model with a model size of 256. We train on 2^{25} tasks with a 90% biased permutation distribution. The task batch size is 128. All images are scaled to a size of $32 \times 32 \times 3$ Inputs are z-normalized across the dataset and all input dimensions.

Table 1 The SGD baseline was obtained by sweeping over learning rates from 10^{-4} to 0.5, optimizers SGD, Adam and Adam with weight decay, one or two layers, and hidden sizes of 32, 64, or 128 on MNIST. The best configuration (most sample efficient) corresponds to a learning rate of 10^{-3}, Adam, and no hidden layers. SGD performs updates online on each one out of the 100 data points. MAML is equivalent to SGD up to the difference that we meta-train the weight initialization according to Equation 2 where θ are the initial parameters of the classifier that is then updated using SGD at meta-test time. All black-box approaches do not use gradient descent at meta-test time. All meta-learning approaches where meta-trained and tuned via grid search on MNIST.

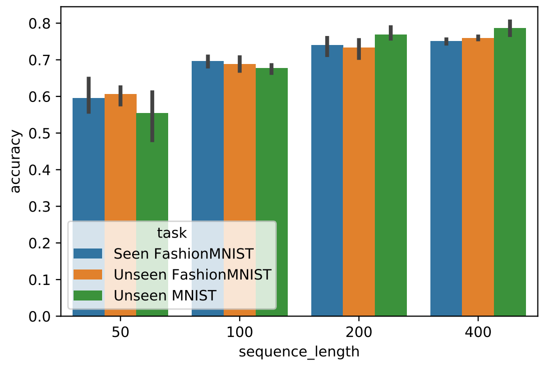

Figure 10: Increasing the sequence length during meta-training and meta-testing improves the predictive performance of the final query in the sequence. Error bars indicate 95% confidence intervals.

Figure 6 We trained a Transformer with model size 64 and 32 seeds for each number-of-tasks-configuration.

Figure 4 Input normalization is disabled.

Figure 5 The Transformer uses a task batch size of 512.

Figure 7 Trained on 2^{16} tasks generated from FashionMNIST with labels fully permuted.

Figure 8 Trained on 2^{16} tasks generated from FashionMNIST with labels fully permuted.

Figure 9 Trained on 2^{16} tasks generated from FashionMNIST with label permutations varied.

A.10 Additional Experiments

Sequence length In all experiments of the main paper we have meta-trained on a sequence length (number of examples) of 100. This is a small training dataset compared to many human-engineered learning algorithms. In general, as long as the learning algorithm does not overfit the training data, more examples should increase the predictive performance. In Figure 10 we investigate how our model scales to longer sequence lengths. We observe that the final accuracy of the last query in the sequence consistently increases with longer sequences. The generalization to longer sequences than those seen during meta-training is another important direction for future work.

Gradient and update statistics To better understand the properties of the loss plateau, we visualize different statistics of the gradients, optimizer, and updates. In Figure 11, we track the exponential moving average statistics of Adam before the loss plateau and after (dashed vertical line). In Figure 12 we investigate how gradients differ between settings with a plateau and settings with a biased distribution where the plateau is avoided. We plot the cosine similarity between consecutive optimization steps, the gradient L2-norm, and the similarity and norm of the weight updates after normalization with Adam. The statistics are plotted cumulatively or smoothed with a Gaussian filter for better readability. The gradient and update cosine similarity differ only marginally between cases with a plateau and cases without. We observe that the gradient L2-norm in the plateau is half as big as in the biased distribution case, although the updates that Adam applies are going towards zero. This also results in not moving far from parameter initialization when in the plateau. We hypothesize this has to do with varying gradient norms when looking at individual parameter tensors (Figure 13). We observe that the gradients have a small norm for most tensors, except for the last layer.

Batch size and number of tasks influence on plateau length Instead of looking at the plateau length in terms of the number of steps (Figure 8), we may also be concerned with the total number of tasks seen within the plateau. This is relevant in particular when the task batch is not processed fully in parallel but gradients are accumulated. Figure 14 shows the same figure but with the number of tasks in the plateau on the y-axis instead. It can be observed that larger batch-sizes actually increase the data requirement to leave the plateau, despite decreasing the plateau in terms of the number of optimization steps. Similarly, a larger task training distribution requires a larger number of tasks to be seen within the plateau.

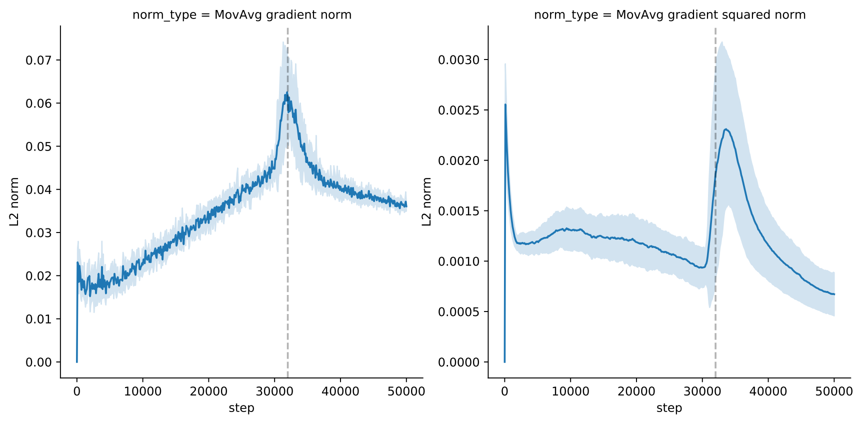

Figure 11: L2-norms of the gradient and squared gradient exponential moving average in Adam. The dashed line corresponds to the loss drop at the end of the loss plateau.

Adjusting Adam's ϵ or changing the optimizer As discussed in the main paper and visualized in Figure 15b, decreasing ϵ significantly shortens the plateau. This is due to the rescaling of very small gradient magnitudes being limited by ϵ. At the same time it incurs some instability. Directly normalizing the gradient by applying the sign function element-wise (Figure 15a) to the exponential gradient average shortens the plateau even further.

When memorization happens, can we elicit grokking? In Figure 8a we have seen that an insufficiently large task distribution can lead to memorization instead of general learning-to-learn. At the same time, Figure 9 showed that biasing the data distribution is helpful to avoid loss plateaus. Power et al. [30] observed a phenomenon which they called "grokking" in which even after having converged in terms of training loss, test loss may suddenly decrease. Large amounts of regularization, like weight decay with a coefficient of 1.0 were found to facilitate this behavior. Is grokking connected to the optimization behavior we observe, and if so, do similar interventions help in our setting? We look in particular at the boundary of memorization and generalization ($2^{14} = 16384$) where doubling the number of tasks a few more times would lead to generalization. Figure 16 shows three task settings, $2^{10}, 2^{14}, 2^{16}$, and three different weight decay coefficients, $0.01, 0.1, 1.0$. The setting of 2^{16} tasks shows generalization by default and only serves as a baseline for the weight decay coefficient analysis. In the cases of memorization due to too few tasks, we have not been able to produce grokking behavior.

Optimization difficulties in VSML Previous work has observed several optimization difficulties: Slower convergence, local minima, unstable training, or loss plateaus at the beginning of training. Figure 17 shows some of these difficulties in the context of VSML [6]. Because VSML has permutation invariance built into the architecture as an inductive bias, changing the number of tasks has only a small effect. We observe that in particular deeper architectures make meta-optimization more difficult.

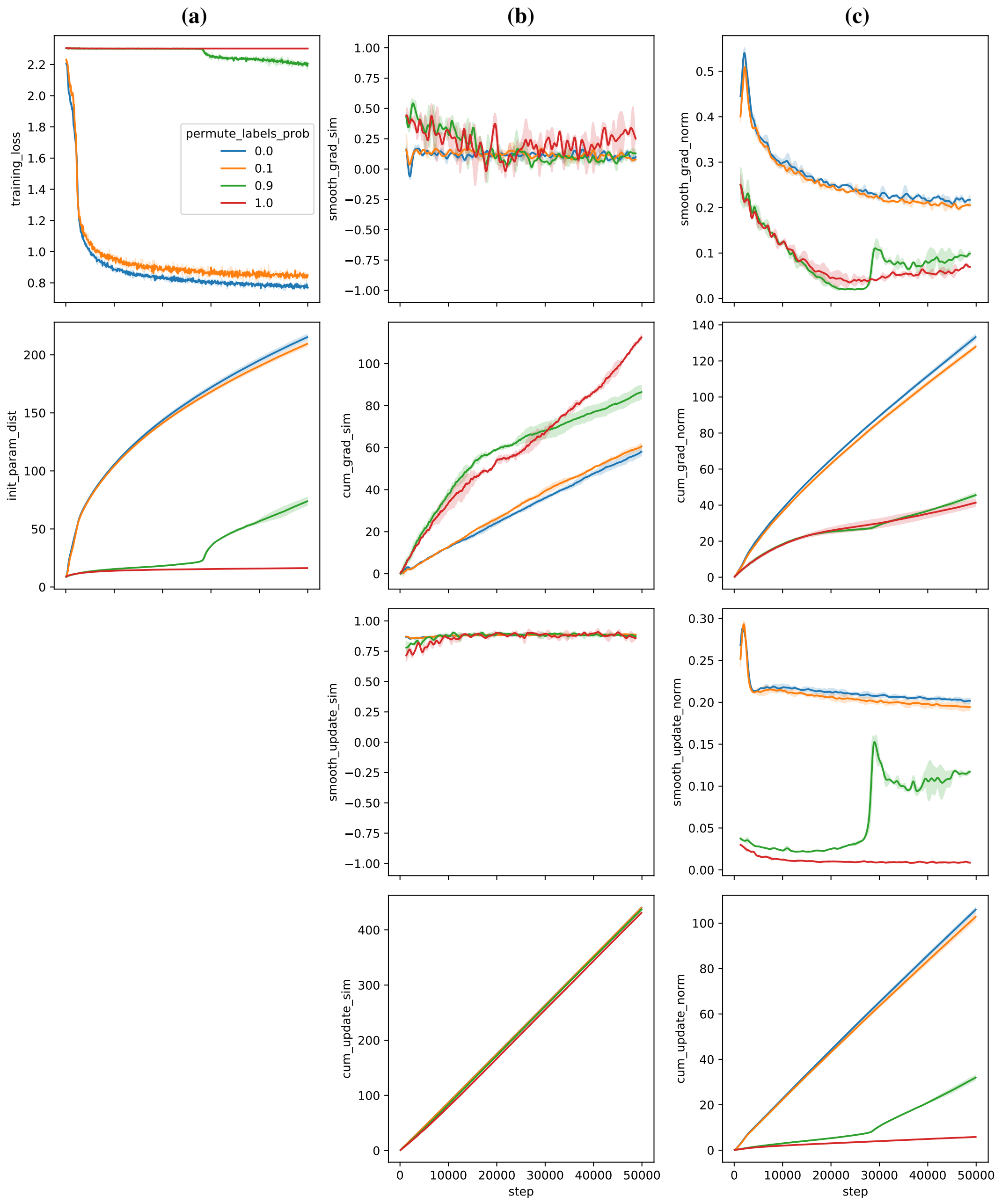

Figure 12: Gradient and Adam update statistics for differently biased data distributions. **(a)** Plateaus in the loss are influenced by the bias in the data distribution. Plateaus result in moving away slowly from the parameter initialization. **(b)** The cosine similarity of both gradients and updates in consecutive steps is only marginally different with or without a loss plateau. **(c)** While the gradient norm is about half as big when a plateau exists, the updates are going towards zero.

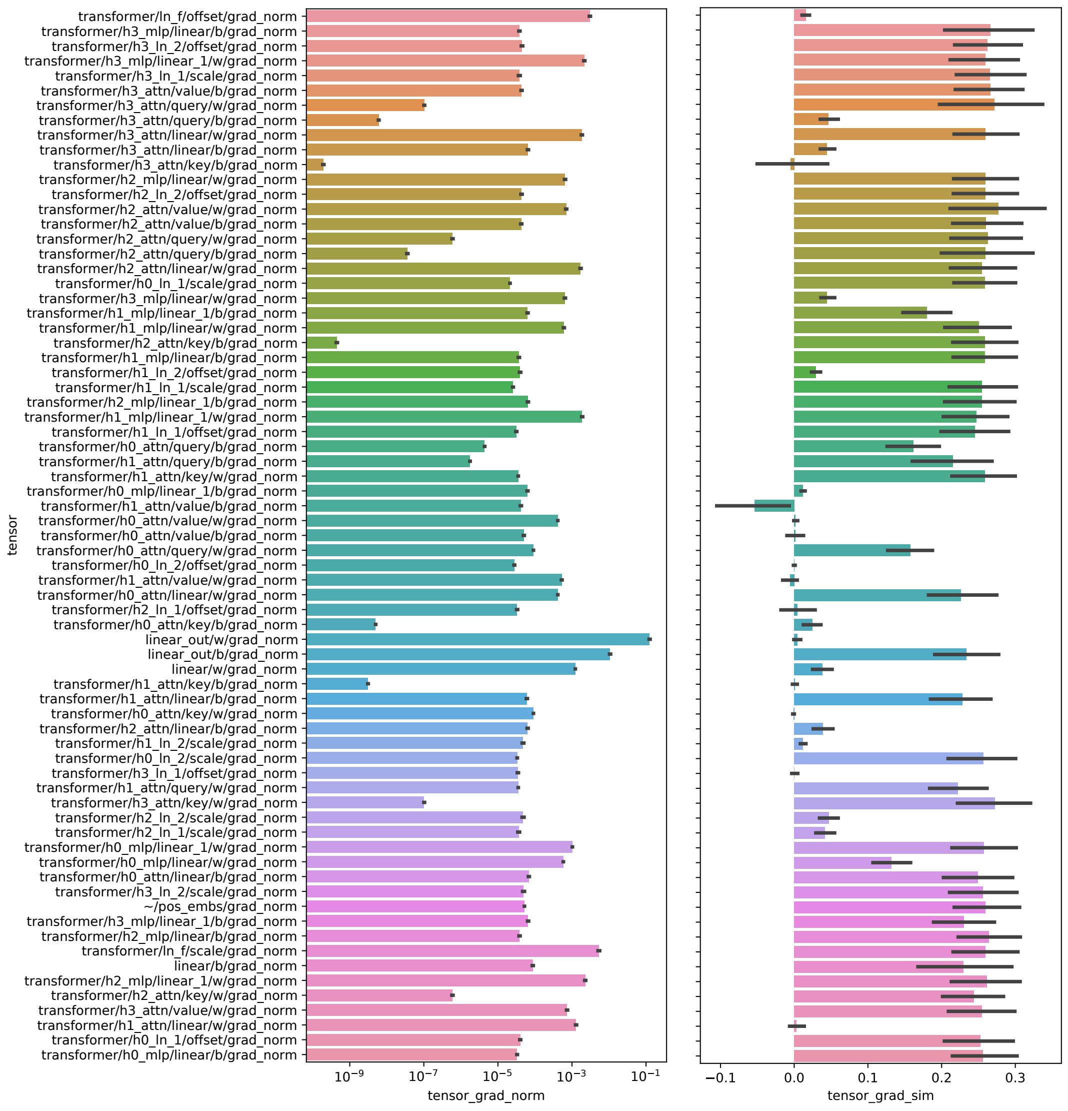

Figure 13: Gradient L2 norms (left) and gradient cosine similarity for consecutive optimization steps (right) for different parameter tensors. The last (output) layer has the largest gradients. Most other gradients are small.

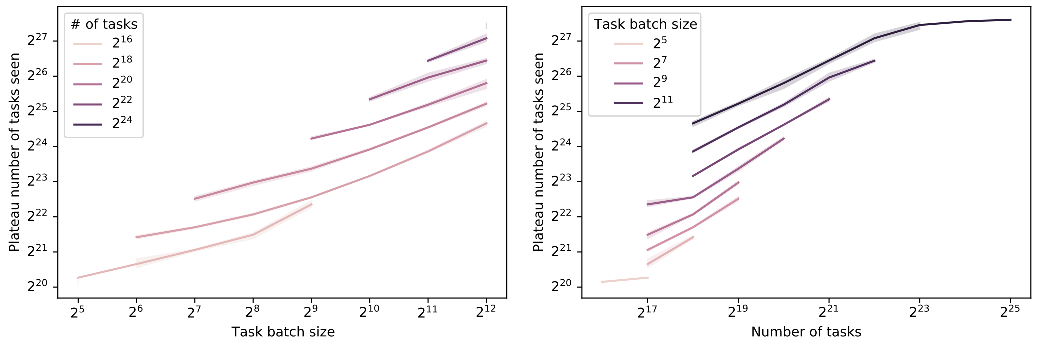

Figure 14: Instead of plotting the loss plateau length in terms of optimization steps, we look at the total number of tasks seen within the plateau as a function of the task batch size and the number of tasks in the training distribution. An increase in the task batch size leads to more tasks to be processed to leave the plateau.

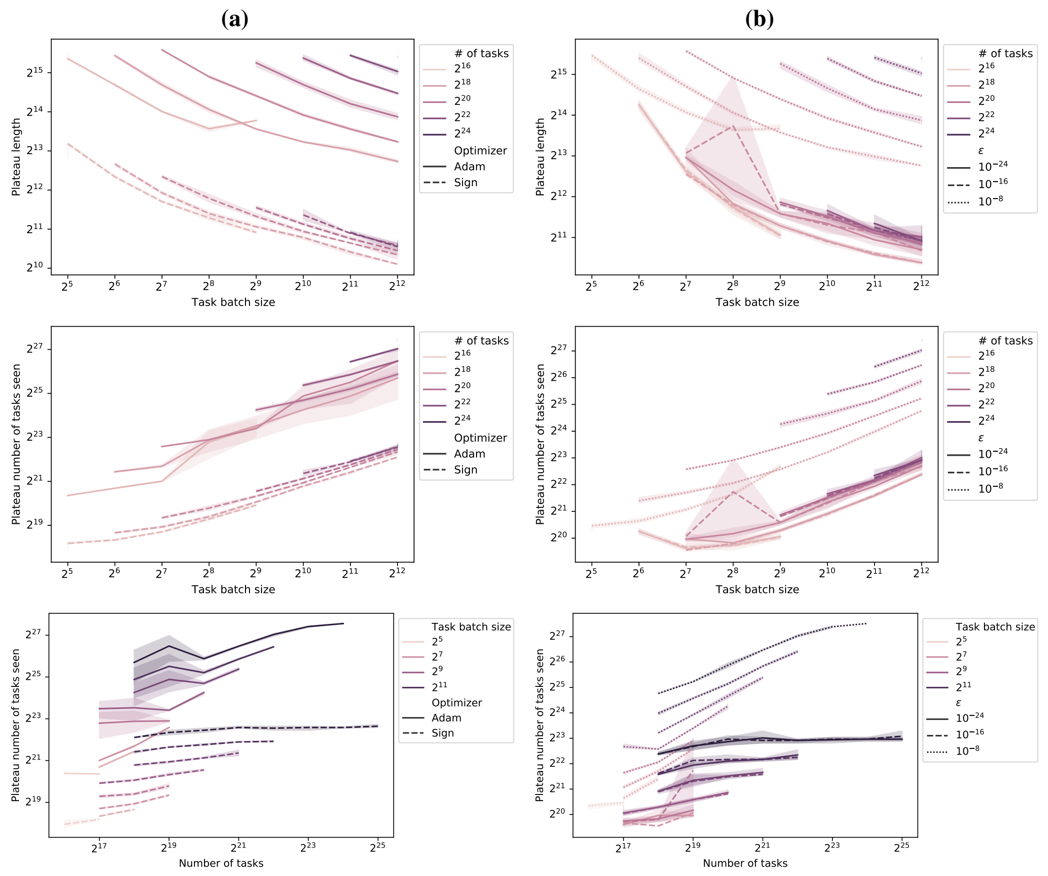

Figure 15: **(a)** When replacing Adam with a sign normalization of the gradient or **(b)** reducing ϵ the plateau length is significantly shorter.

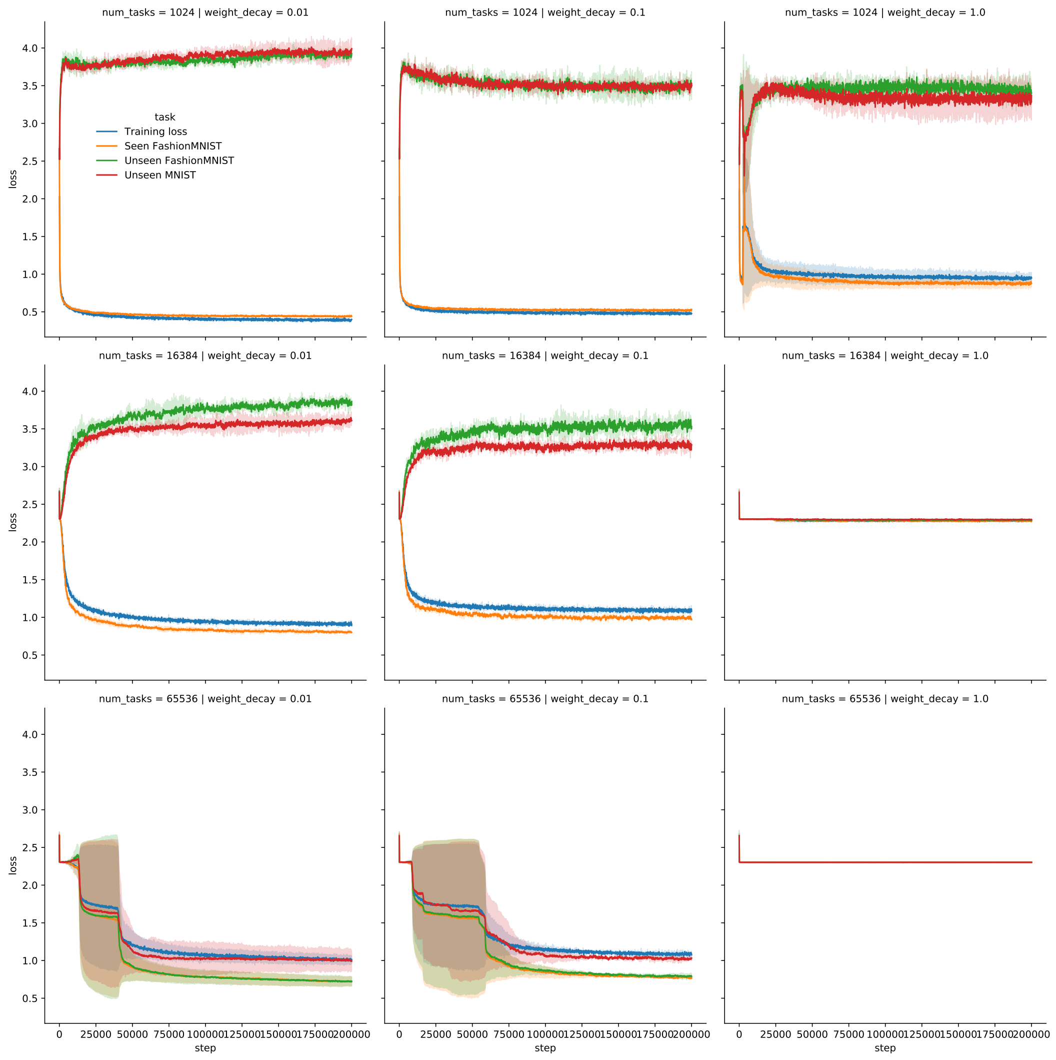

Figure 16: We investigate whether grokking as defined in Power et al. [30] can be produced when we observe memorization on a smaller numbers of tasks. This would correspond to the test loss decreasing long after the training loss has converged. We have not been able to elicit this behavior when looking at different numbers of tasks and weight decay coefficients.

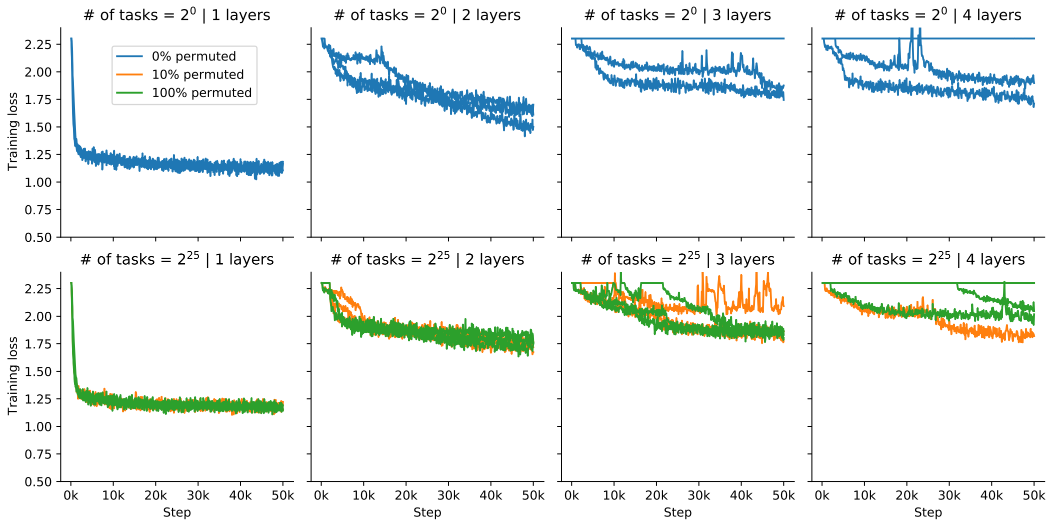

Figure 17: Loss plateaus and slow convergence with deeper variants of VSML.