Model Recycling: Model component reuse to promote in-context learning

Lindsay M. Smith
Princeton University
lindsay.smith@princeton.edu

Chase Goddard
Princeton University
cgoddard@princeton.edu

Vudtiwat Ngampruetikorn[*]
University of Sydney
vudtiwat.ngampruetikorn@sydney.edu.au

David J. Schwab[*]
The Graduate Center, CUNY
dschwab@gc.cuny.edu

Abstract

In-context learning (ICL) is a behavior seen in transformer-based models where, during inference, the model is able to leverage examples of a novel task in order to perform accurately on that task. Here we study the role of different model components on ICL behavior via model component recycling. Previous work has found a plateau in the training loss before models begin to learn a general-purpose ICL solution. We explore a model recycling experiment related to ICL, investigating whether recycling model components can reduce the early plateau in the training loss and whether certain components impact ICL more than others. We find that transferring embeddings and early layers of the transformer from a trained model to an untrained model results in the elimination of the plateau seen in standard model training. Moreover, transferring only later layers of the transformer does not result in significant plateau reductions, indicating the importance of the embeddings and early transformer layers in ICL performance.

1 Introduction

In-context learning (ICL) is an intriguing phenomenon in transformers where, during inference, examples of a novel task are presented in the context, and the model is able to perform the task without any new training or weight updates [3]. Despite its central role in now widely adopted large language models, what aspects of the model architecture and training data enable ICL remain unclear. Recent works suggest that larger models seem to elicit better ICL [1], and whether ICL emerges depends also on training data diversity [2, 7]. In addition, growing evidence highlights the training dynamics as an important consideration of systems exhibiting ICL [8, 3].

In this paper, we explore how different components of the model affect the phenomena of ICL, focusing on the role of the embeddings and of the transformer itself. We consider the linear regression setting [7], where the transformer model must learn linear functions from the context inputted to the model. We sample tasks drawn from a high task diversity distribution by choosing a set of weight vectors from a hypersphere.

We leverage two experimental setups to probe the role of different aspects of the architecture on the emergence of ICL (Figure 1). In particular, we investigate how to reduce the plateau that occurs in the training loss before an ICL solution begins to be learned [4]. We find that the loss plateau can be removed by transferring just the trained embeddings and the first two layers of the transformer

[*]DJS and VN contributed to this work equally.

to a randomly-initialized transformer, either freezing the embeddings or allowing them to continue training, and then training again (Figure 2). The plateau can be reduced by simply transferring the embeddings, or the embeddings and the first layer of the transformer, but it is not enough to entirely remove the plateau.

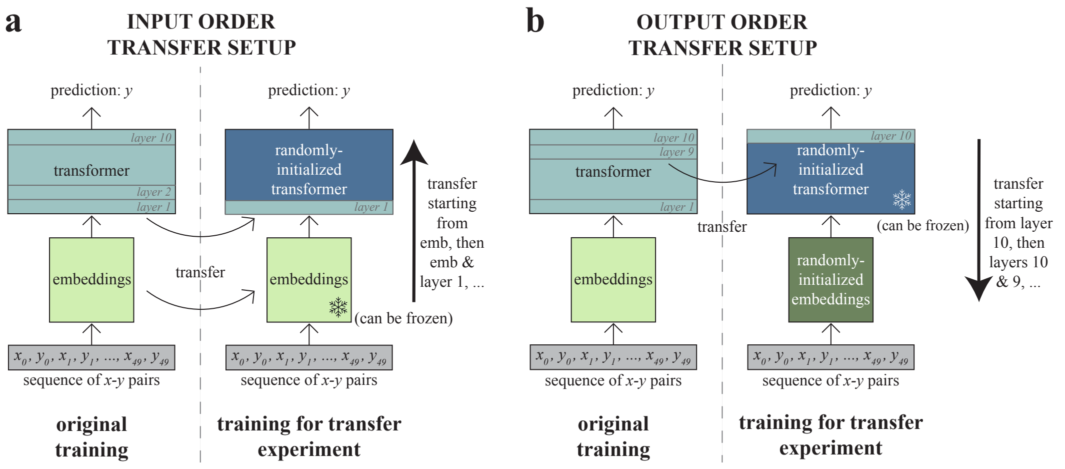
a
INPUT ORDER TRANSFER SETUP
prediction: y
transformer
layer 10
layer 2
layer 1
randomly-initialized transformer
embeddings
transfer
(can be frozen)
transfer starting from emb, then emb & layer 1, ...
x0, y0, x1, y1, ..., x49, y49
sequence of x-y pairs
original training
training for transfer experiment
b
OUTPUT ORDER TRANSFER SETUP
layer 9
randomly-initialized embeddings
transfer starting from layer 10, then layers 10 & 9, ...

Figure 1: The setup of our models for the two different training scenarios. The original model, consisting of embeddings and a transformer, is on the left of the dotted line in both panels. **(a)** We perform our input order transfer experiments by transferring the components of the trained model to a new, randomly-initialized model. We start transferring components from the "input" side, i.e. the embeddings, then the embeddings and the first layer of the transformer, then the embeddings and first two layers of the transformer, and so on through the embeddings and all layers of the transformer. **(b)** The same as in (a), but instead we perform our component recycling experiments by transferring the components from the "output" side, i.e., the last layer of the transformer, then the last two layers of the transformer, and so on through all the layers and the embeddings.

2 Methodology

We use a standard GPT-2 style transformer [9] with a total hidden dimension of $d_h = 128$, 10 layers, and 8 attention heads in all experiments. Our context consists of sequences of $n = 50$ (x, y) pairs for a total sequence length of 100 tokens (50 x's and 50 y's). We focus on $d = 10$ dimensional regression in all experiments. We use a learned linear embedding from $d = 10$ dimensions to $d_h = 128$ dimensions to convert the sequence data into tokens for the transformer and a PyTorch Embedding layer [6] as the learnable positional embeddings. We also pad the target values y_i with $d - 1$ zeroes. These linear embeddings and the positional embeddings are the embeddings that are imported and/or frozen in our experiments.

We define a linear-regression task to be a vector $w \in \mathbb{R}^d$. We construct the tasks the transformer must learn in-context by creating a sequence of example pairs $\{x_1, y_1, \dots, x_n\}$, where $y_i = w^T x_i + \epsilon_i$. Here, $x_i \sim \mathcal{N}(0, I_d)$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. We sample the vectors w from $p(w)$, the uniform distribution on the surface of the unit hypersphere, $\{w \mid w \cdot w = 1\}$. In our experiments, we sample a set of 2^{10} training tasks from $p(w)$ to ensure sufficiently high task diversity.

During training, the transformer T_θ is optimized to minimize the mean squared error (MSE) between a *context* of data $C_k \equiv \{x_1, y_1, \dots, x_k\}$ and the target y_k, thus training on all tokens in the sequence. For each batch, the tasks w are drawn without replacement from the set of 2^{10} training tasks. We minimize the MSE,

$$L_{\text{train}}(\theta) = \mathbb{E}_w \left[\frac{1}{n} \sum_{k=1}^{n} \left(T_\theta(C_k) - y_k\right)^2 \right] \tag{1}$$

over the parameters θ using AdamW [5] with default parameters except for the learning rate, which we set to 3×10^{-4}.

We evaluate the performance of the transformer on unseen ICL tasks by sampling *new* w vectors i.i.d. from $p(w)$. We evaluate models by computing the MSE between the full context C_n and the final target y_n (testing the last token predictions):

$$L_{\text{test}}(\theta) = \mathbb{E}_{w \sim p(w)} \left[(T_\theta(C_n) - y_n)^2 \right] \tag{2}$$

3 Experiments and Results

We perform an experiment exploring the role of architectural components on ICL by investigating the plateau found in the early stage of the training curve.

To explore how to reduce the initial plateau length in the training loss curve, we performed experiments where we transferred parts of trained models to otherwise randomly-initialized models. We hypothesized that by training a model and transferring some pretrained components to newly initialized models, we can reduce the plateau length. Figure 1 shows the setup of the different kinds of experiments: input-order transfer and output-order transfer. We first train for 100 epochs (10,000 steps) on 2^{10} tasks, as this produces a plateau in training before the model begins to learn an ICL solution.

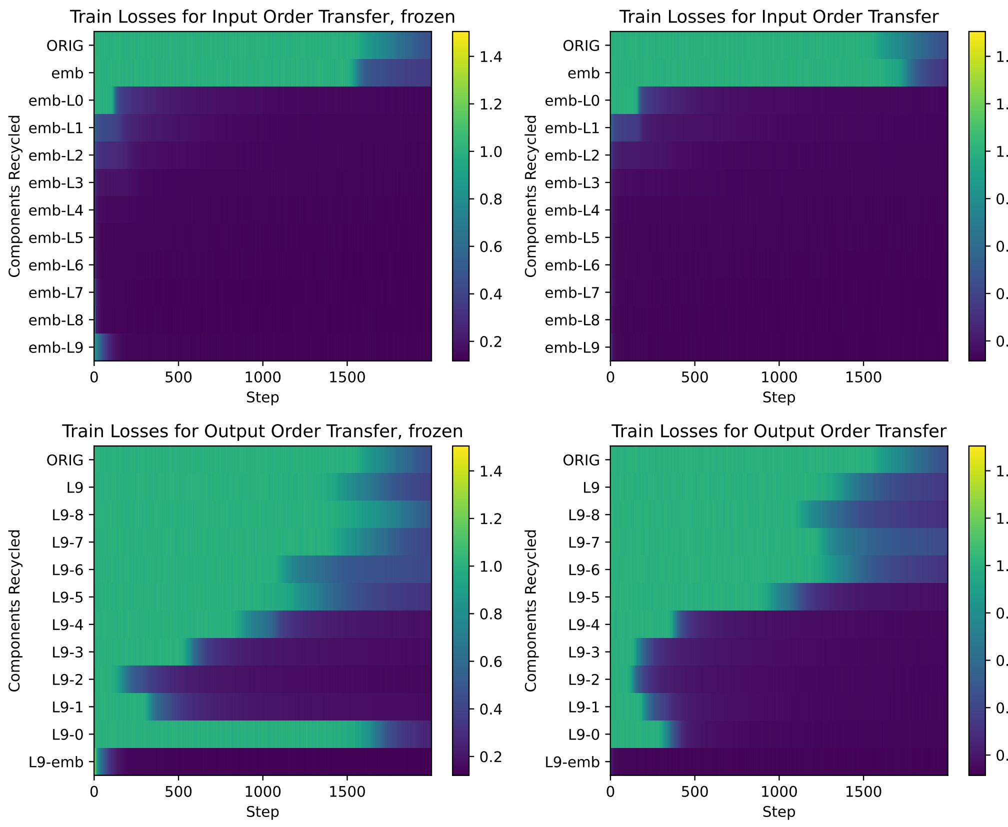

Figure 2: ICL plateau shortening experiments. Only the first 2,000 steps out of 10,000 steps are shown to highlight the plateau. The color represents the training loss from the transfer experiments when training originally on 2^{10} tasks and sampling from the entire hypersphere ('ORIG'), transferring the embeddings ('emb'), and/or layers of the 10-layer transformer ('L0' through 'L9') to a randomly-initialized model, and then training again on 2^{10} tasks. Also tested freezing the transferred embeddings and/or transformer layers (right column) so that they cannot be updated when retraining. Note that the final layer norm and linear output layer of the transformer are not transferred.

For the input-order experiments, we transfer components from the *input* side of the model. First, we transfer only the embeddings (both token and positional). In the subsequent experiment, the first layer of the transformer is transferred together with the embeddings. To complete the set of experiments in this setting, we consider the transfer processes which include more and more layers of the transformer until the last layer is included. We place the transferred components in a new, randomly-initialized model. In our "frozen" experiments, the parameters in these components are fixed; otherwise, they are allowed to be trained with the rest of the model. Figure 1.a shows the setup of the input-order transfer. In the output-order transfer experiment, we perform an analogous set of experiments: we transfer the layers of the transformer and the embeddings from the pretrained model starting from the *output* side of the model, i.e., transferring the final layer of the transformer, and then in the next experiment transferring the final two layers of the transformer, and so on through all the layers and then the embeddings. We either freeze these components or allow them to be further trained, and randomly initialize the rest of the model. Figure 1.b shows the setup of the output-order transfer experiments. We then train these new models – the input-order transfer, frozen-input-order transfer, output-order transfer, and frozen-output-order transfer models – on the same diversity of tasks as in the originally-trained model (2^{10}) and for another 100 epochs.

Transferring from the input side of the model leads to reductions in the plateau in the original training dynamics, and in fact, only transferring the embeddings and the first two layers of the transformer is all that is necessary to completely eliminate the plateau (Figure 2, top left and top right). However, when transferring from the output side of the model, many more layers are needed in order to significantly reduce the plateau. It appears that the deeper layers are less useful in reducing the plateau in the training dynamics, as seen in the output-order experiments (Figure 2, bottom left and bottom right). Interestingly, we observe non-monotonic behavior when transferring more layers in these output-order experiments. We leave the investigation of the origin of the non-monotonicity to future work.

Based on these results, we theorize that the embeddings, which are just a linear layer and a PyTorch Embedding layer, are not expressive enough to allow an otherwise randomly-initialized model to train well, but the embeddings plus the first layer of the transformer have enough information about the tasks to form a robust representation of the ICL tasks, even without the rest of the transformer. We note that we only investigate transferring the embeddings and the layers of the transformer, ignoring the final layer norm and linear output layer of the model.

Freezing the the transferred components so that they cannot be further trained does not seem to have a significant impact on the plateau reduction in the input-order experiments. However, freezing the layers of the transformer in the output-order experiments does affect the plateau when all layers are transferred.

4 Conclusion

We explored an experimental setup to investigate model component recycling for ICL with the aim of reducing the plateau in the training loss commonly seen in ICL work. We found that transferring the embeddings and first two layers of the transformer of a model to an untrained model results in the elimination of the plateau seen in the original training of the model. Transferring only the embeddings and the first layer of the transformer also reduces the plateau length, but does not eliminate it. However, just transferring the final layers of the transformer to an untrained model does not significantly reduce the training plateau. These results have implications both for building a fundamental understanding of ICL and for efficient model training in order to elicit ICL behavior. We hope that the empirical results presented here will help inform future work in developing a theoretical framework for ICL behavior.

Acknowledgments and Disclosure of Funding

LMS is supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2039656. VN acknowledges research funds from the University of Sydney. DJS was partially supported by a Simons Fellowship in the MMLS, a Sloan Fellowship, and the National Science Foundation, through the Center for the Physics of Biological Function (PHY-1734030).

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. URL `http://arxiv.org/abs/2005.14165`. arXiv:2005.14165 [cs].

[2] Stephanie C. Y. 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, November 2022. URL `http://arxiv.org/abs/2205.05055`. arXiv:2205.05055 [cs].

[3] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A Survey on In-context Learning, June 2024. URL `http://arxiv.org/abs/2301.00234`. arXiv:2301.00234 [cs].

[4] Jingwen Fu, Tao Yang, Yuwang Wang, Yan Lu, and Nanning Zheng. Breaking through the learning plateaus of in-context learning in Transformer. June 2024. URL `https://openreview.net/forum?id=2K87GFLYWz`.

[5] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization, January 2019. URL `http://arxiv.org/abs/1711.05101`. arXiv:1711.05101 [cs, math].

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

[7] Allan Raventos, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression. November 2023. URL `https://openreview.net/forum?id=BtAz4a5xDg`.

[8] Aaditya K. Singh, Stephanie C. Y. Chan, Ted Moskovitz, Erin Grant, Andrew M. Saxe, and Felix Hill. The Transient Nature of Emergent In-Context Learning in Transformers, December 2023. URL `http://arxiv.org/abs/2311.08360`. arXiv:2311.08360 [cs].

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

A Appendix / supplemental material

A.1 Additional model and figure details

Our transformer has an expansion factor of 4 and dropout set to 0.1. We use a batch size of 100 sequences and trained for 10,000 steps. Experiments were run on one NVIDIA A100 GPU or one MIG instance (out of 7) on one NVIDIA A100 GPU.