Comparing Representational and Functional Similarity in Small Transformer Language Models

Dan Friedman, Andrew Lampinen, Lucas Dixon, Danqi Chen, Asma Ghandeharioun

Abstract

What kinds of solutions do neural networks learn? In many situations, it would be helpful to be able to characterize the solution learned by a neural network, including for answering scientific questions (e.g. how do architecture changes affect generalization) and addressing practical concerns (e.g. auditing for potentially unsafe behavior). One approach is to try to understand these models by studying the representations that they learn—for example, comparing whether two networks learn similar representations. However, it is not always clear how much representation-level analyses can tell us about how a model makes predictions. In this work, we explore this question in the context of small Transformer language models, which we train on a synthetic, hierarchical language task. We train models with different sizes and random initializations, evaluating performance over the course of training and on a variety of systematic generalization splits. We find that existing methods for measuring representation similarity are not always correlated with behavioral metrics—i.e. models with similar representations do not always make similar predictions—and the results vary depending on the choice of representation. Our results highlight the importance of understanding representations in terms of the role they play in the neural algorithm.

1 Introduction

Representation similarity analysis [10] is a common method for characterizing the relationship between different neural networks [e.g. 11, 18, 20, 13, 9, 6, 15, 16]. However, it is unclear whether we can expect models with similar representations to make similar predictions [4, 8]. In general, the relationship between representation and behavior depends on how the representations are used by later parts of a model, which is a function of the model architecture and training setting.

Therefore, in this work, we study the relationship between representational and functional measures empirically in the context of a particular architecture and setting—small Transformer language models [19], trained on the Dyck balanced parenthesis languages. The Dyck languages exhibit fundamental properties of natural languages—recursive, hierarchical structure, which give rise to long-distance dependencies—but are simple enough to admit simple, human-interpretable algorithms [24, 22]. For this reason, they have been widely studied in prior work on the expressivity of Transformer language models [7, 24], and in interpretability [23]. A benefit of this setting is that we can largely reverse-engineer the solutions that these models learn, to better understand the relationship between representations and behavior.

Specifically, we train models with different embedding sizes and random initializations and evaluate performance on different systematic generalization splits. We measure representation similarity using standard metrics from prior work [4], and evaluate whether models that have similar representations also make similar predictions. We find that measures of representational similarity do not always agree with functional similarity. In particular, the differences are greatest on out-of-distribution evaluation settings, and representations from different model components lead to different similarity profiles. This analysis highlights some of the interpretability challenges that have

Preprint.

been noted in prior work, including the limitations of methods that analyze models with respect to a particular data distribution [2], and that analyze individual model components in isolation [23]. Our findings can motivate future work on understanding neural networks in terms of the algorithms that they implement.

2 Setting

Dyck languages Dyck-k is the family of balanced parenthesis languages with up to k bracket types. Following the notation of Wen et al. [23], the vocabulary of Dyck-k is the words $\{1, \ldots, 2k\}$, where, for any $t \in [k]$, the words $2t-1$ and $2t$ are the opening and closing brackets of type t, respectively. Given a sentence $w_1, \ldots, w_n$, the nesting depth at any position i is defined as the difference between the number of opening brackets in $w_{1:i}$ and the number of closing brackets in $w_{1:i}$. As in prior work [24, 14, 23], we focus on bounded-depth Dyck languages [7], denoted Dyck-(k, m), where m is the maximum nesting depth. We train models on Dyck-$(20, 10)$ and evaluate on two generalization splits. First, we recreate the structural generalization split described by Murty et al. [14] by sampling sentences with bracket structures that were not in the training set (**Unseen struct**). The bracket structure of a sentence is defined as the sequence of opening and closing brackets (e.g., the structure of "([])[]" is "001101"). Second, we create a **Unseen depth** generalization set by sampling sentences from Dyck-$(20, 20)$ and only keeping those sentences with a maximum nesting depth of at least 10. We evaluate models' accuracy at predicting closing brackets that are at least 10 positions away from the corresponding opening bracket, and score the prediction by the closing bracket to which the model assigns the highest likelihood.

Transformer language models The Transformer [19] is a neural network architecture for processing sequence data. The input is a sequence of tokens $w_1, \ldots, w_N \in \mathcal{V}$ in a discrete vocabulary $\mathcal{V}$. At the input layer, the model maps the tokens to a sequence of d-dimensional embeddings $\boldsymbol{X}^{(0)} \in \mathbb{R}^{N \times d}$, which is the sum of a learned token embedding and a positional embedding. Each subsequent layer i consists of a multi-head attention layer (MHA) followed by a multilayer perceptron layer (MLP): $\boldsymbol{X}^{(i)} = \boldsymbol{X}^{(i-1)} + \mathrm{MHA}^{(i)}(\boldsymbol{X}^{(i-1)}) + \mathrm{MLP}^{(i)}(\boldsymbol{X}^{(i-1)} + \mathrm{MHA}^{(i)}(\boldsymbol{X}^{(i-1)}))$.[1] Multi-head attention (with H heads) can be written as

$$\mathrm{MHA}(\boldsymbol{X}) = \sum_{h=1}^{H} \mathrm{softmax}(\boldsymbol{X}\boldsymbol{W}_Q^h(\boldsymbol{X}\boldsymbol{W}_K^h)^\top)\boldsymbol{X}\boldsymbol{W}_V^h\boldsymbol{W}_O^h,$$

where $\boldsymbol{W}_Q^h, \boldsymbol{W}_K^h, \boldsymbol{W}_V^h \in \mathbb{R}^{d \times d_h}$ are referred to as the *query*, *key*, and *value* projections respectively, and $\boldsymbol{W}_O^h \in \mathbb{R}^{d_h \times d}$ projects the output value back to the model dimension. The MLP layer operates at each position independently; we use a two-layer feedforward network: $\mathrm{MLP}(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2$, where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}, \boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$, and σ is the ReLU function. The output of the model is a sequence of token embeddings, $\boldsymbol{X}^{(L)} \in \mathbb{R}^{N \times d}$. We focus on autoregressive Transformer language models, which define a distribution over next words, given a prefix $w_1, \ldots, w_{i-1} \in \mathcal{V}$: $p(w_i \mid w_1, \ldots, w_{i-1}) \propto \exp(\theta_{w_i}^\top \boldsymbol{X}_{i-1}^{(L)})$, where $\theta_{w_i} \in \mathbb{R}^d$ is a vector of output weights for word w_i.

Transformer algorithms for Dyck The main algorithmic task that a Transformer has to learn is to identify, at each position, the most recent un-closed opening bracket, which determines the type of closing bracket that can appear in the next position. For example, given the prefix "[](([]", the only possible closing bracket is ")". Yao et al. [24] developed a construction for accomplishing this task with a two-layer Transformer: The first attention layer calculates the bracket depth at each position, defined as the number of opening brackets minus the number of closing brackets; the first-layer MLP embeds each depth to a separate direction; and the second attention layer uses depth embeddings to find the most recent unmatched opening bracket, which is the most recent bracket at the current depth. Empirically, we find that the Transformers we train resemble this construction, with the second-layer attention head generally attending to the correct matching bracket.

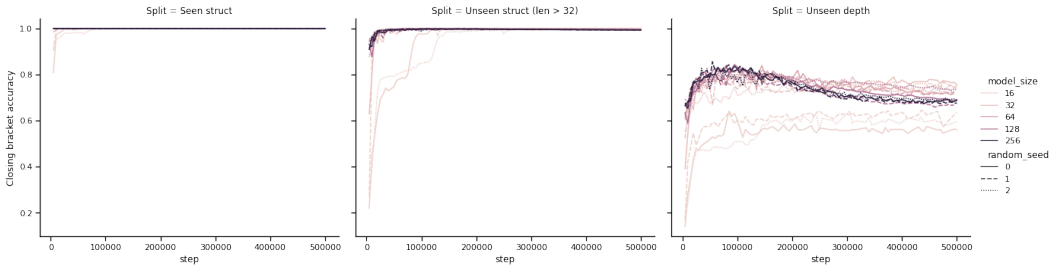

Figure 1: Accuracy at predicting closing brackets over the course of training, for models with different hidden dimensions and random initializations.

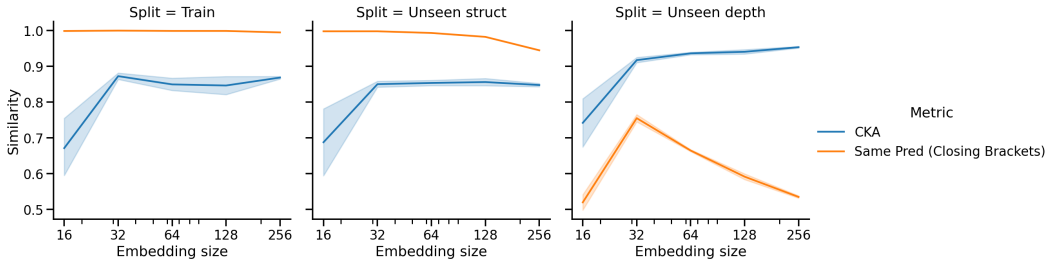

Figure 2: For each embedding size, we compare the similarity between models with different random initializations and the same size. The representations are the key and query embeddings from the second attention layer. The representation similarity metric increases with model width, but the prediction similarity score decreases.

3 Experiments

Bracket-matching accuracy We train two-layer Transformer language models on the Dyck-$(20, 10)$ training data described in the previous section. Each layer has one attention head, one MLP, and layer normalization, and we vary the embedding dimension. For each embedding size, we train models with three random initializations. Details about the model and training procedure are in Appendix A.2 and A.3. Fig. 1 plots the bracket-matching accuracy over the course of training. All models eventually achieve almost perfect accuracy on the unseen structure generalization set, but performance varies on the depth generalization set both across models and over the course of training. In particular, the widest models achieve the highest accuracy earlier in training, after which point generalization performance degrades.

Comparing key and query embeddings We start by examining the key and query embeddings from the second attention layer. These embeddings play a central role in the Transformer algorithm described above, by encoding the nesting depth at each position, and, from visual inspection, we find that the Transformers we train resemble this construction (Appendix Fig. 5). Therefore, we would expect that similarity between these key and query embeddings should be correlated with functional similarity. In Figure 2, we compare the average similarity of each model to the other models with the same embedding size. We measure representational similarity using Centered Kernel Analyis (CKA) [9] and define Prediction Similarity as the percentage of cases for which both models make the same prediction, considering only cases where the true next token is a closing bracket. See results with other similarity measures in Appendix B. Representational and behavioral similarity metrics paint different pictures: representational similarity increases with model size, consistent with findings in prior work [13, 9]. However, on the out-of-distribution splits, prediction similarity decreases with embedding size.

Measuring similarity over time In Figure 3, we plot the same metrics over the course of training. In terms of representational similarity, the smallest models become more similar to each other over time, while the larger models become slightly less similar, but the overall order of model sizes

[1]The standard Transformer also includes a layer-normalization layer [1], which we omit here.

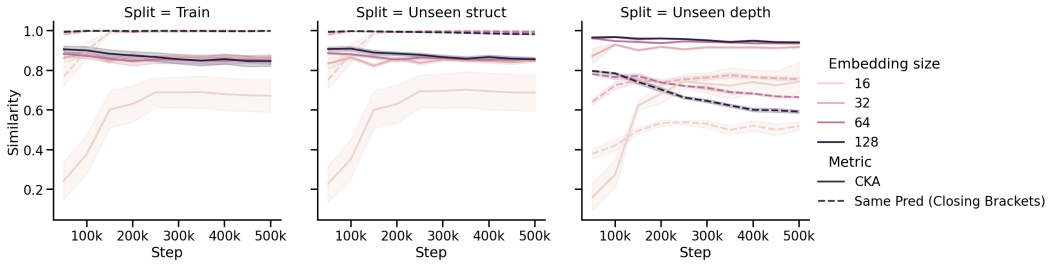

Figure 3: Average similarity between models with the same width, over the course of training. The representations are the key and query embeddings from the second attention layer.

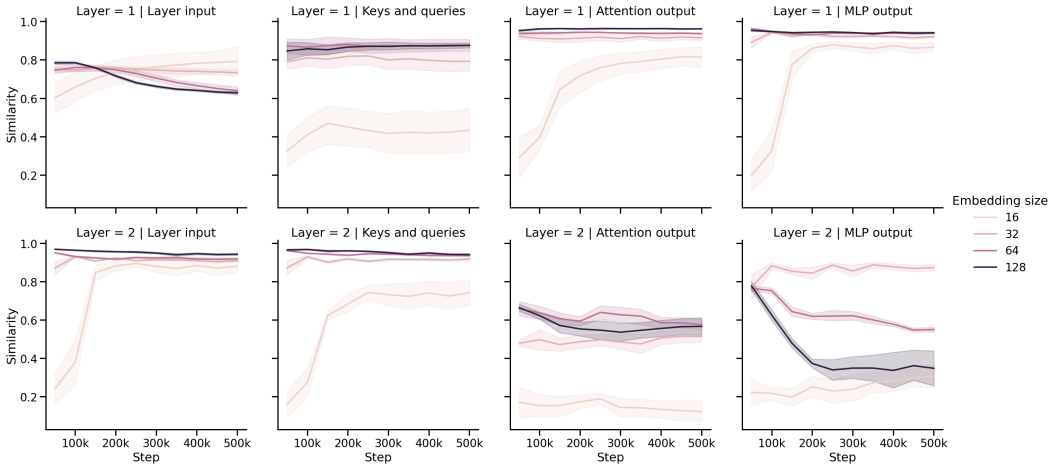

Figure 4: Average similarity (CKA) between models with the same width, over the course of training, broken down by representation, and evaluated on the depth generalization split.

remains the same. In terms of prediction similarity, the similarity scores diverge over time, with the largest models becoming the least similar to each other by the end of training.

Comparing different model components In the previous sections, we conjectured that prediction similarity would be correlated with similarity between the second-layer key and query embeddings, because these play a central role in the bracket matching algorithm. One possible explanation for our negative finding is that another model component plays a larger role in determining what the models will predict, especially in out-of-distribution settings. In Figure 4, we plot the similarity trajectories using different choices of representation on the depth-generalization split. The representations that most resemble the trajectory of prediction similarity (plotted in Fig. 3) are from the second layer MLP output, suggesting that the MLP might be more important for explaining prediction differences in this setting, and underscoring the limitations of analyzing model components in isolation.

4 Conclusion

In this work, we studied the connection between representational similarity and functional similarity in Transformer language models. In particular, we focused on small Transformers trained on Dyck balanced-parenthesis languages, a setup that would allow us to largely reverse-engineer the neural algorithm and study its behavior under systematically different distribution shifts. Across different model sizes and initializations, we found that representational similarity is not always associated with model behavior, especially when evaluated out of distribution. As model size increases, representational similarity can even diverge from behavioral similarity over the course of training. Finally, the role that the representation plays in the neural algorithm dictates the extent to which it would be associated with the model's behavior.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for BERT. *arXiv preprint arXiv:2104.07143*, 2021.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[4] Frances Ding, Jean-Stanislas Denain, and Jacob Steinhardt. Grounding representation similarity through statistical testing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[5] Tom Hennigan, Trevor Cai, Tamara Norman, Lena Martens, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

[6] Katherine Hermann and Andrew Lampinen. What shapes feature representations? Exploring datasets, architectures, and training. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D Manning. RNNs can generate bounded hierarchical languages with optimal memory. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, 2020.

[8] Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *arXiv preprint arXiv:2305.06329*, 2023.

[9] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning (ICML)*, 2019.

[10] Nikolaus Kriegeskorte, Marieke Mur, and Peter A Bandettini. Representational similarity analysis – connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience*, page 4, 2008.

[11] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *International Conference on Learning Representations (ICLR)*, 2016.

[12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[13] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[14] Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. Grokking of hierarchical structure in vanilla transformers. In *Association for Computational Linguistics (ACL)*, pages 439–448, 2023.

[15] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? Uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations (ICLR)*, 2021.

[16] Jason Phang, Haokun Liu, and Samuel Bowman. Fine-tuned Transformers show clusters of similar representations across layers. In *BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 529–538, 2021.

[17] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *European Chapter of the Association for Computational Linguistics (EACL)*, pages 157–163, 2017.

[18] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, 2017.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NIPS)*, 30, 2017.

[20] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[21] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning (ICML)*, pages 5247–5256. PMLR, 2018.

[22] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like Transformers. In *International Conference on Machine Learning (ICML)*, pages 11080–11090. PMLR, 2021.

[23] Kaiyue Wen, Yuchen Li, Bingbin Liu, and Andrej Risteski. (Un)interpretability of Transformers: A case study with Dyck grammars. In *ICML 2023 Workshop on Challenges in Deploying Generative AI*, 2023.

[24] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 3770–3785, 2021.

A Implementation Details

A.1 Dataset Details

As described in Section 2, we sample sentences from Dyck-$(10, 20)$, the language of balanced brackets with 20 bracket types and a maximum nesting depth of 10. We use the sampling distribution described and implemented by Hewitt et al. [7],[2] following existing work [24, 14]. We insert a special beginning-of-sequence token to the begin of each sequence, and append an end-of-sequence token to the end. Note that we discard sentences with lengths greater than 512. The training set contains 200k sentences and all the generalization sets contain 20k sentences. In all cases, we sample sentences, discarding sentences according to the rules described in Section 2, until the dataset has the desired size.

A.2 Model Details

For our Dyck experiments, we use a two-layer Transformer, with each layer consisting of one attention head, one MLP, and one layer normalization layer. The model has an embedding size in $\{16, 32, 64, 128, 256\}$, and the attention key and query embeddings have the same dimension. Each MLP has one hidden layer with a dimension four times larger than the embedding size, followed by a ReLU activation. The input token embeddings are tied to the output token embeddings [17], and we use absolute, learned position embeddings. The model is implemented in JAX [3] and adapted from the Haiku Transformer [5].

[2] https://github.com/john-hewitt/dyckkm-learning

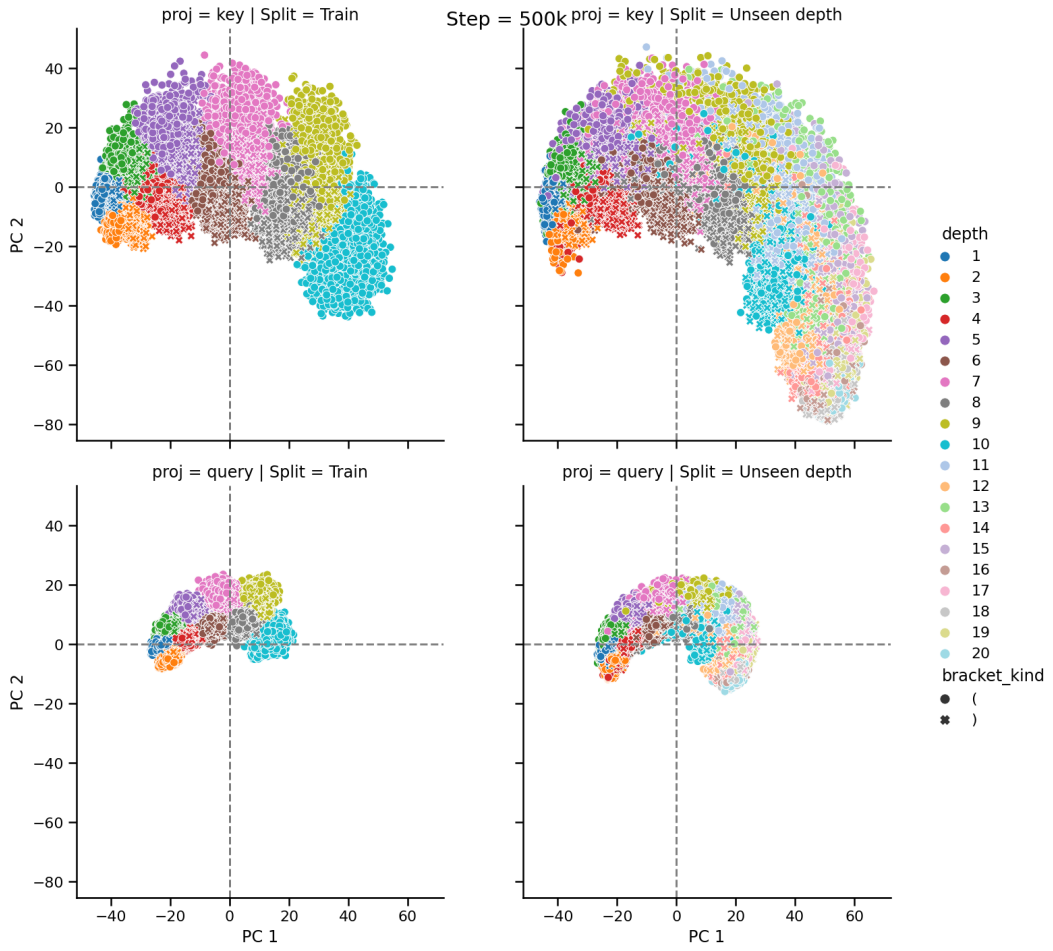

Figure 5: Key and query embeddings for inputs from the training set and out-of-distribution depth generalization set, projected onto the first and second singular vectors. The embeddings are from the second layer of a model with an embedding size of 256 after training for 500,000 steps. Visually, the representations for out-of-distribution depths support some form of depth generalization, but the deeper depths are not separated as effectively, leading to prediction errors.

A.3 Training Details

We train the models to minimize the cross entropy loss:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{w_{1:n} \in \mathcal{D}} \frac{1}{n} \sum_{i=2}^{n} p(w_i \mid w_{1:i-1}),$$

where $\mathcal{D}$ is the training set, $p(w_i \mid w_{1:i-1})$ is defined according to Section 2, and w_1 is always a special beginning-of-sequence token. We train the model for 500,000 steps with a batch size of 128 and use the final model for further analysis. We use the AdamW optimizer [12] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-7, and a weight decay factor of 1e-4. We set the learning rate to follow a linear warmup for the first 10,000 steps followed by a square root decay, with a maximum learning rate of 5e-3. We do not use dropout.

B Additional Results

Figure 5 plots the key and query embeddings from a Dyck language model. These embeddings resemble human-written constructions for modeling Dyck languages with Transformers [24, 21]: the

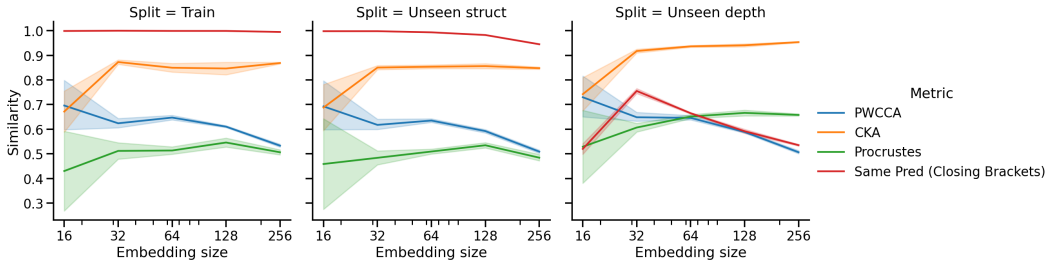

Figure 6: For each embedding size, we compare the similarity between models with different random initializations and the same size. The representations are the key and query embeddings from the second attention layer. We compare Projection-weighted Canonical Correlation Analysis (PWCCA), Centered Kernel Analysis, and Orthogonal Proctrustes analysis, following Ding et al. [4]. The representation similarity metrics paint different pictures of the relationships between these models.

first attention layer calculates the nesting depth at each position, and the second attention matches brackets by matching depths.

Figure 6 plots the similarities between models with the same width from the final training checkpoint, using additional representation similarity metrics. The metrics are calculated using the code from Ding et al. [4],[3] (adjusted to give similarity scores rather than dissimilarity scores).

[3] https://github.com/js-d/sim_metric