The Devil is in the Detail: Simple Tricks Improve Systematic Generalization of Transformers

Róbert Csordás Kazuki Irie Jürgen Schmidhuber
The Swiss AI Lab IDSIA, USI & SUPSI, Lugano, Switzerland
{robert, kazuki, juergen}@idsia.ch

Abstract

Recently, many datasets have been proposed to test the systematic generalization ability of neural networks. The companion baseline Transformers, typically trained with default hyper-parameters from standard tasks, are shown to fail dramatically. Here we demonstrate that by revisiting model configurations as basic as scaling of embeddings, early stopping, relative positional embedding, and Universal Transformer variants, we can drastically improve the performance of Transformers on systematic generalization. We report improvements on five popular datasets: SCAN, CFQ, PCFG, COGS, and Mathematics dataset. Our models improve accuracy from 50% to 85% on the PCFG productivity split, and from 35% to 81% on COGS. On SCAN, relative positional embedding largely mitigates the EOS decision problem (Newman et al., 2020), yielding 100% accuracy on the length split with a cutoff at 26. Importantly, performance differences between these models are typically invisible on the IID data split. This calls for proper generalization validation sets for developing neural networks that generalize systematically. We publicly release the code to reproduce our results[1].

1 Introduction

Systematic generalization (Fodor et al., 1988) is a desired property for neural networks to extrapolate compositional rules seen during training beyond training distribution: for example, performing different combinations of known rules or applying them to longer problems. Despite the progress of artificial neural networks in recent years, the problem of systematic generalization still remains unsolved (Fodor and McLaughlin, 1990; Lake and Baroni, 2018; Liska et al., 2018; Greff et al., 2020; Hupkes et al., 2020). While there has been much progress in the past years (Bahdanau et al., 2019; Korrel et al., 2019; Lake, 2019; Li et al., 2019; Russin et al., 2019), in particular on the popular SCAN dataset (Lake and Baroni, 2018) where some methods even achieve 100% accuracy by introducing some non-trivial symbolic components into the system (Chen et al., 2020; Liu et al., 2020), the flexibility of such solutions is questionable. In fact, the existing SCAN-inspired solutions have limited performance gains on other datasets (Furrer et al., 2020; Shaw et al., 2020). It is thus not enough to solely focus on the SCAN dataset to progress research on systematic generalization.

Recently, many datasets have been proposed for testing systematic generalization, including PCFG (Hupkes et al., 2020) and COGS (Kim and Linzen, 2020). The baseline Transformer models which are released together with the dataset are typically shown to dramatically fail at the task. However, the configurations of these baseline models are questionable. In most cases, some standard practices from machine translation are applied without modification. Also, some existing techniques such as relative positional embedding (Shaw et al., 2018; Dai et al., 2019), which are relevant for the problem, are not part of the baseline.

In order to develop and evaluate methods to improve systematic generalization, it is necessary to have not only good datasets but also strong baselines to correctly evaluate the limits of existing architectures and to avoid false sense of progress over bad baselines. In this work, we demonstrate that the capability of Transformers (Vaswani et al., 2017) and in particular its universal variants (Dehghani et al., 2019) on these tasks are largely underestimated. We show that careful designs of model and training configurations are particularly important for these reasoning tasks testing systematic generalization. By revisiting configurations such as basic scaling of word and positional embeddings, early stopping strategy, and relative positional em-

[1] https://github.com/robertcsordas/transformer_generalization

bedding, we dramatically improve the performance of the baseline Transformers. We conduct experiments on five datasets: SCAN (Lake and Baroni, 2018), CFQ (Keysers et al., 2020), PCFG (Hupkes et al., 2020), COGS (Kim and Linzen, 2020), and Mathematic dataset (Saxton et al., 2019). In particular, our new models improve the accuracy on the PCFG productivity split from 50% to 85%, on the systematicity split from 72% to 96%, and on COGS from 35% to 81% over the existing baselines. On the SCAN dataset, we show that our models with relative positional embedding largely mitigates the so-called end-of-sentence (EOS) decision problem (Newman et al., 2020), achieving 100% accuracy on the length split with a cutoff at 26.

Also importantly, we show that despite these dramatic performance gaps, all these models perform equally well on IID validation datasets. The consequence of this observation is the need for proper generalization validation sets for developing neural networks for systematic generalization.

We thoroughly discuss guidelines that empirically yield good performance across various datasets, and we will publicly release the code to make our results reproducible.

2 Datasets and Model Architectures for Systematic Generalization

Here we describe the five datasets, and specify the Transformer model variants we use in our experiments. The selected datasets include both already popular ones and recently proposed ones. Statistics of the datasets can be found in Table 10 in the appendix.

2.1 Datasets

Many datasets in the language domain have been proposed to test systematic generalization. All datasets we consider here can be formulated as a sequence-to-sequence mapping task (Sutskever et al., 2014; Graves, 2012). Common to all these datasets, the test set is sampled from a distribution which is systematically different from the one for training: for example, the test set might systematically contain longer sequences, new combinations or deeper compositions of known rules. We call this split the *generalization split*. Most of the datasets also come with a conventional split, where the train and test (and validation, if available) sets are independently and identically distributed samples. We call this the *IID split*. In this paper, we consider the following five datasets:

SCAN (Lake and Baroni, 2018). The task consists of mapping a sentence in natural language into a sequence of commands simulating navigation in a grid world. The commands are compositional: e.g. an input `jump twice` should be translated to `JUMP JUMP`. It comes with multiple data splits: in addition to the "simple" IID split, in the "length" split, the training sequences are shorter than test ones, and in the "add primitive" splits, some commands are presented in the training set only in isolation, without being composed with others. The test set focuses on these excluded combinations.

CFQ (Keysers et al., 2020). The task consists of translating a natural language question to a Freebase SPARQL query. For example `Was M0 a director and producer of M1` should be translated to `SELECT count(*) WHERE {M0 ns:film.director.film M1 . M0 ns:film.producer.film | ns:film.production_company.films M1}`. The authors introduce splits based on "compound divergence" which measures the difference between the parse trees in the different data splits. The authors experimentally show that it is well correlated with generalization difficulty. It also comes with a length-based split.

PCFG (Hupkes et al., 2020). The task consists of list manipulations and operations that should be executed. For example, `reverse copy O14 O4 C12 J14 W3` should be translated to `W3 J14 C12 O4 O14`. It comes with different splits for testing different aspects of generalization. In this work, we focus on the "productivity" split, which focuses on generalization to longer sequences, and on the "systematicity" split, which is about recombining constituents in novel ways.

COGS (Kim and Linzen, 2020). The task consists of semantic parsing which maps an English sentence to a logical form. For example, `The puppy slept.` should be translated to `* puppy ( x _ 1 ) ; sleep . agent ( x _ 2, x _ 1 )`. It comes with a single split, with a training, IID validation and OOD generalization testing set.

Mathematics Dataset (Saxton et al., 2019). The task consists of high school level textual math questions, e.g. `What is -5 - 110911?` should be translated to `-110916`. The data is

split into different subsets by the problem category, called modules. Some of them come with an extrapolation set, designed to measure generalization. The amount of total data is very large and thus expensive to train on, but different modules can be studied individually. We focus on “add_or_sub" and “place_value" modules.

2.2 Model Architectures

We focus our analysis on two Transformer architectures: standard Transformers (Vaswani et al., 2017) and Universal Transformers (Dehghani et al., 2019), and in both cases with absolute or relative positional embedding (Dai et al., 2019). Our Universal Transformer variants are simply Transformers with shared weights between layers, without adaptive computation time (Schmidhuber, 2012; Graves, 2016) and timestep embedding. Positional embedding are only added to the first layer.

Universal Transformers are particularly relevant for reasoning and algorithmic tasks. For example, if we assume a task which consists in executing a sequence of operations, a regular Transformer will learn successive operations in successive layers with separate weights. In consequence, if only some particular orderings of the operations are seen during training, each layer will only learn a subset of the operations, and thus, it will be impossible for them to recombine operations in an arbitrary order. Moreover, if the same operation has to be reused multiple times, the network has to re-learn it, which is harmful for systematic generalization and reduces the data efficiency of the model (Csordás et al., 2021). Universal Transformers have the potential to overcome this limitation: sharing the weights between each layer makes it possible to reuse the existing knowledge from different compositions. On the downside, the Universal Transformer’s capacity can be limited because of the weight sharing.

3 Improving Transformers on Systematic Generalization

In this section, we present methods which greatly improve Transformers on systematic generalization tasks, while they could be considered as details in standard tasks. For each method, we provide experimental evidences on a few representative datasets. In Section 4, we apply these findings to all datasets.

3.1 Addressing the EOS Decision Problem with Relative Positional Embedding

The EOS decision problem. A thorough analysis by Newman et al. (2020) highlights that LSTMs and Transformers struggle to generalize to longer output lengths than they are trained for. Specifically, it is shown that the decision when to end the sequence (the EOS decision) often overfits to the specific positions observed in the train set. To measure whether the models are otherwise able to solve the task, they conduct a so-called *oracle evaluation*: they ignore the EOS token during evaluation, and use the ground-truth sequence length to stop decoding. The performance with this evaluation mode is much better, which illustrates that the problem is indeed the EOS decision. More surprisingly, if the model is trained without EOS token as part of output vocabulary (thus it can only be evaluated in oracle mode), the performance is further improved. It is concluded that teaching the model when to end the sequence has undesirable side effects on the model’s length generalization ability.

We show that the main cause of this EOS decision problem in the case of Transformers is the absolute positional embedding. Generally speaking, the meaning of a word is rarely dependent on the word’s absolute position in a document but depends on its neighbors. Motivated by this assumption, various relative positional embedding methods (Shaw et al., 2018; Dai et al., 2019) have been proposed. Unfortunately, they have not been considered for systematic generalization in prior work (however, see Sec. 5), even though they are particularly relevant for that.

We test Transformers with relative positional embedding in the form used in Transformer XL (Dai et al., 2019). Since it is designed for auto-regressive models, we directly apply it in the decoder of our model, while for the encoder, we use a symmetrical variant of it (see Appendix C). The interface between encoder and decoder uses the standard attention without any positional embedding.

Our experimental setting is similar to Newman et al. (2020). The length split in SCAN dataset restricts the length of the train samples to 22 tokens (the test set consists of samples with an output of more than 22 tokens). This removes some compositions from the train set entirely, which introduces additional difficulty to the task. 80% of the test set consists of these missing compositions. In order to mitigate the issue of unknown composition

Table 1: Exact match accuracies on length splits with different cutoffs. Reported results are the median of 5 runs. Trafo denotes Transformers. The numbers in the rows `+EOS+Oracle` and `−EOS+Oracle` are taken from Newman et al. (2020) as reference numbers but they can not be compared to others as they are evaluated with oracle length. Our models use different hyperparameters compared to theirs. We refer to Section 3.1 for details.

	ℓ *(length cutoff)*	22	24	25	26	27	28	30	32	33	36	40
Reference	`+EOS`	0.00	0.05	0.04	0.00	0.09	0.00	0.09	0.35	0.00	*0.00*	0.00
	`+EOS+Oracle`	*0.53*	*0.51*	*0.69*	*0.76*	*0.74*	*0.57*	*0.78*	*0.66*	*0.77*	***1.00***	*0.97*
	`−EOS+Oracle`	***0.58***	***0.54***	*0.67*	*0.82*	*0.88*	*0.85*	*0.89*	*0.82*	***1.00***	***1.00***	***1.00***
Ours (+EOS)	Trafo	0.00	0.04	0.19	0.29	0.30	0.08	0.24	0.36	0.00	0.00	0.00
	+ Relative PE	0.20	0.12	0.31	0.61	**1.00**	**1.00**	**1.00**	0.94	**1.00**	**1.00**	**1.00**
	Universal Trafo	0.02	0.05	0.14	0.21	0.26	0.00	0.06	0.35	0.00	0.00	0.00
	+ Relative PE	0.20	0.12	**0.71**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

and focus purely on the length problem, Newman et al. (2020) re-split SCAN by introducing different length cutoffs and report the performance of each split. We test our models similarly. However, our preliminary experiments showed the performance of the original model is additionally limited by being too shallow: it uses only 2 layers for both the encoder and decoder. We increased the number of layers to 3. To compensate for the increased number of parameters, we decrease the size of the feed-forward layers from 1024 to 256. In total, this reduces the number of parameters by 30%. We train our models with Adam optimizer, a learning rate of 10^{-4}, batch size of 128 for 50k steps.

The results are shown in Table 1. In order to show that our changes of hyperparameters are not the main reason for the improved performance, we report the performance of our modified model without relative positional embedding (row Trafo). We also include the results from Newman et al. (2020) for reference. We report the performance of Universal Transformer models trained with identical hyperparameters. All our models are trained *to predict* the EOS token and are evaluated *without* oracle (`+EOS` configuration). It can be seen that both our standard and Universal Transformers with absolute positional embedding have near-zero accuracy for all length cutoffs, whereas models with relative positional embedding excel: they even outperform the models trained without EOS prediction and evaluated with the ground-truth length.

Although Table 1 highlights the advantages of relative positional embedding and shows that they can largely mitigate the EOS-overfitting issue, this does not mean that the problem of generalizing to longer sequences is fully solved. The sub-optimal performance on short length cutoffs (22-25) indicates that the model finds it hard to zero-shot generalize to unseen compositions of specific rules. To improve these results further, research on models which assume analogies between rules and compositions are necessary, such that they can recombine known constituents without any training example.

Further benefits of relative positional embedding. In addition to the benefit highlighted in the previous paragraph, we found that models with relative positional embedding are easier to train in general. They converge faster (Figure 6 in the appendix) and are less sensitive to batch size (Table 9 in the appendix). As another empirical finding, we note that relative Transformers without shared layers sometimes catastrophically fail before reaching their final accuracy: the accuracy drops to 0, and it never recovers. We observed this with PCFG productivity split and the "Math: place_value" task. Reducing the number of parameters (either using Universal Transformers or reducing the state size) usually stabilizes the network.

3.2 Model Selection Should Be Done Carefully

The danger of early stopping. Another crucial aspect greatly influencing the generalization performance of Transformers is model selection, in particular early stopping. In fact, on these datasets, it is a common practice to use only the IID split to tune hyperparameters or select models with early stopping (e.g. Kim and Linzen (2020)). However, since any reasonable models achieve nearly 100% accuracy on the IID validation set, there is no good reason to believe this to be a good practice for selecting models for generalization splits. To test this hypothesis, we train models on COGS dataset without early stopping, but with a fixed number of 50k training steps. The best model achieved a test accuracy of 81%, while the original performance

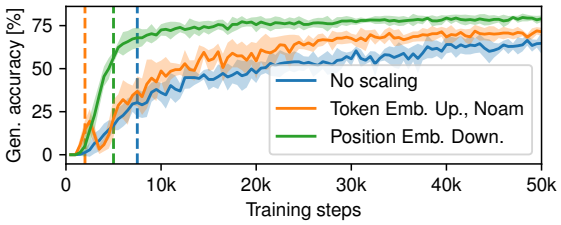


Figure 1: Generalization accuracy on COGS as a function of training steps for standard Transformers with different embedding scaling schemes. The vertical lines show the median of the early stopping points for the five runs. Early stopping parameters are from Kim and Linzen (2020). "Token Emb. Up., Noam" corresponds to the baseline configuration (Kim and Linzen, 2020). See Sec. 3.3 for details on scaling.

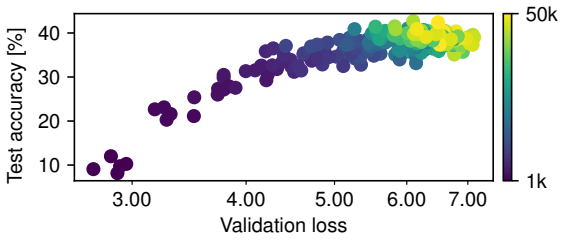


Figure 2: Relationship between validation loss and test accuracy (same distribution) on CFQ MCD 1 split for a relative Transformer. The color shows the training step. Five runs are shown. The loss has a logarithmic scale. High accuracy corresponds to higher loss, which is unexpected. For detailed analysis, see Figure 5.

Table 2: Final IID validation and generalizations accuracy for COGS (50k steps) and PCFG Productivity set (300k steps) with different scaling (Section 3.3). Token Embedding Upscaling (TEU) is unstable on PCFG with our hyperparameters. Position Embedding Downscaling (PED) performs the best on both datasets.

		IID Validation	Gen. Test
COGS	TEU	**1.00 ± 0.00**	0.78 ± 0.03
	No scaling	**1.00 ± 0.00**	0.62 ± 0.06
	PED	**1.00 ± 0.00**	**0.80 ± 0.00**
PCFG	TEU	0.92 ± 0.07	0.47 ± 0.27
	No scaling	**0.97 ± 0.01**	0.63 ± 0.02
	PED	0.96 ± 0.01	**0.65 ± 0.03**

by Kim and Linzen (2020) is 35%. Motivated by this huge performance gap, we had no other choice but to conduct an analysis on the generalization split to demonstrate the danger of early stopping and discrepancies between the performance on the IID and generalization split. The corresponding results are shown in Figure 1 (further effect of embedding scaling is discussed in next Sec. 3.3) and Table 2. Following Kim and Linzen (2020), we measure the model's performance every 500 steps, and mark the point where early stopping with patience of 5 would pick the best performing model. It can be seen that in some cases the model chosen by early stopping is not even reaching half of the final generalization accuracy.

To confirm this observation in the exact setting of Kim and Linzen (2020), we also disabled the early stopping in the original codebase [2], and observed that the accuracy improved to 65% without any other tricks. We discuss further performance improvements on COGS dataset in Section 4.4.

[2] https://github.com/najoungkim/COGS

The lack of validation set for the generalization split. A general problem raised in the previous paragraph is the lack of validation set for evaluating models for generalization. Most of the datasets come without a validation set for the generalization split (SCAN, COGS, and PCFG). Although CFQ comes with such a set, the authors argue that only the IID split should be used for hyperparameter search, and it is not clear what should be used for model development.

In order to test novel ideas, a way to gradually measure progress is necessary, such that the effect of changes can be evaluated. If the test set is used for developing the model, it implicitly risks overfitting to this test set. On the other hand, measuring performance on the IID split does not necessarily provide any valuable information about the generalization performance on the systematically different test set (see Table 2). The IID accuracy of all the considered datasets is 100% (except on PCFG where it's also almost 100%); thus, no further improvement, nor potential difference between generalization performance of models can be measured (see also Table 8 in the appendix).

It would be beneficial if future datasets would have a validation and test set for both the IID *and* the generalization split. For the generalization split, the test set could be designed to be more difficult than the validation set. This way, the validation set can be used to measure progress during development, but overfitting to it would prevent the model to generalize well to the test set. Such a division can be easily done on the splits for testing productivity. For other types of generalization, we could use multiple datasets sharing the same generalization problem. Some of them could be dedicated for development and others for testing.

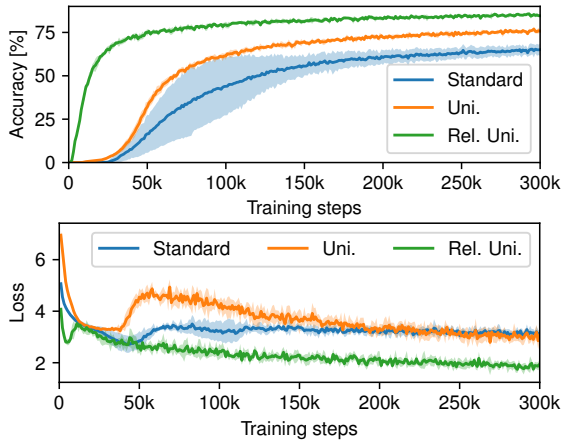


Figure 3: Test loss and accuracy on PCFG during training. The loss exhibits an epoch-wise double descent phenomenon (Nakkiran et al., 2019), while the accuracy increases monotonically. Standard Transformer with PED (Sec. 3.3), Universal Transformer with absolute, and relative positional embeddings are shown.

Intriguing relationship between generalization accuracy and loss. Finally, we also note the importance of using accuracy (instead of loss) as the model selection criterion. We find that the generalization accuracy and loss do not necessarily correlate, while sometimes, model selection based on the loss is reported in practice e.g. in Kim and Linzen (2020). Examples of this undesirable behavior are shown on Figure 2 for CFQ and on Figure 4 in the appendix for COGS dataset. On these datasets, the loss and accuracy on the generalization split both grows during training. We conducted an analysis to understand the cause of this surprising phenomenon, we find that the total loss grows because the loss of the samples with incorrect outputs increases more than it improves on the correct ones. For the corresponding experimental results, we refer to Figure 5 in the appendix. We conclude that even if a validation set is available for the generalization split, it would be crucial to use the *accuracy instead of the loss* for early stopping and hyperparameter tuning.

Finally, on PCFG dataset, we observed epoch-wise double descent phenomenon (Nakkiran et al., 2019), as shown in Figure 3. This can lead to equally problematic results if the loss is used for model selection or tuning.

3.3 Large Impacts of Embedding Scaling

The last surprising detail which greatly influences generalization performance of Transformers is the choice of embedding scaling scheme. This is especially important for Transformers with absolute positional embedding, where the word and positional embedding have to be combined. We experimented with the following scaling schemes:

1. Token Embedding Upscaling (TEU). This is the standard scaling used by Vaswani et al. (2017). It uses Glorot initialization (Glorot and Bengio, 2010) for the word embeddings. However, the range of the sinusoidal positional embedding is always in $[-1, 1]$. Since the positional embedding is directly added to the word embeddings, this discrepancy can make the model untrainable. Thus, the authors upscale the word embeddings by $\sqrt{d_{\text{model}}}$ where d_{model} is the embedding size. OpenNMT[3], the framework used for the baseline models for PCFG and COGS datasets respectively by Hupkes et al. (2020) and Kim and Linzen (2020), also uses this scaling scheme.

2. No scaling. It initializes the word embedding with $\mathcal{N}(0, 1)$ (normal distribution with mean 0 and standard deviation of 1). Positional embeddings are added without scaling.

3. Position Embedding Downscaling (PED), which uses Kaiming initialization (He et al., 2015), and scales the positional embeddings by $\frac{1}{\sqrt{d_{\text{model}}}}$.

The PED differs from TEU used in Vaswani et al. (2017) in two ways: instead of scaling the embedding up, PED scales the positional embedding down and uses Kaiming instead of Glorot initialization. The magnitude of the embeddings should not depend on the number of words in the vocabulary but on the embedding dimension.

Table 2 shows the results. Although "no scaling" variant is better than TEU on the PCFG test set, it is worse on the COGS test set. PED performs consistently the best on both datasets. Importantly, the gap between the best and worst configurations is large on the test sets. The choice of scaling thus also contributes in the large improvements we report over the existing baselines.

4 Results Across Different Datasets

In this section, we apply the methods we illustrated in the previous section across different datasets. Table 3 provides an overview of all improvements we

[3] https://opennmt.net/

obtain on all considered datasets. Unless reported otherwise, all results are the mean and standard deviation of 5 different random seeds. If multiple embedding scaling schemes are available, we pick the best performing one for a fair comparison. Transformer variants with relative positional embedding outperform the absolute variants on almost all tested datasets. Except for COGS and CFQ MCD 1, the universal variants outperform the standard ones. In the following, we discuss and highlight the improvements we obtained for each individual dataset.

4.1 SCAN

We focused on the **length split** of the dataset. We show that it is possible to mitigate the effect of overfitting to the absolute position of the EOS token by using relative positional embedding. We already discussed the details in Sec. 3.1 and Table 1.

4.2 CFQ

On the **output length split** of CFQ, our Universal Transformer with absolute positional embedding achieves significantly better performance than the one reported in Keysers et al. (2020): 77% versus $\sim 66\%$[4]. Here, we were unable to identify the exact reason for this large improvement. The only architectural difference between the models is that ours does not make use of any timestep (i.e. layer ID) embedding. Also, the positional embedding is only injected to the first layer in case of absolute positional embeddings (Sec. 2.2). The relative positional embedding variant performs even better, achieving 81%. This confirms the importance of using relative positional embedding as a default choice for length generalization tasks, as we also demonstrated on SCAN in Sec. 3.1.

On the **MCD splits**, our results slightly outperform the baseline in Keysers et al. (2020), as shown in Table 3. Relative Universal Transformers perform marginally better than all other variants, except for MCD 1 split, where the standard Transformer wins with a slight margin. We use hyperparameters from Keysers et al. (2020). We report performance after 35k training training steps.

4.3 PCFG

The performance of different models on the PCFG dataset is shown on Table 3. First of all, simply by increasing the number of training epochs from 25, used by Hupkes et al. (2020), to $\sim$237 (300k steps), our model achieves 65% on the **productivity split** compared to the 50% reported in Hupkes et al. (2020) and 87% compared to 72% on the **systematicity split**. Furthermore, we found that Universal Transformers with relative positional embeddings further improve performance to a large extent, achieving 85% final performance on the **productivity** and 96% on the **systematicity split**. We experienced instabilities while training Transformers with relative positional embeddings on the productivity split; thus, the corresponding numbers are omitted in Table 3 and Figure 6 in the appendix.

[4] As Keysers et al. (2020) only report charts, the exact value is unknown.

4.4 COGS

On COGS, our best model achieves the generalization accuracy of 81% which greatly outperforms the 35% accuracy reported in Kim and Linzen (2020). As we discussed in Sec. 3.2, just by removing early stopping in the setting of Kim and Linzen (2020), the performance improves to 65%. Moreover, the baseline with early stopping is very sensitive to the random seed and even sensitive to the GPU type it is run on. Changing the seed in the official repository from 1 to 2 causes a dramatic performance drop with a 2.5% final accuracy. By changing the scaling of embeddings (Sec. 3.3), disabling label smoothing, fixing the learning rate to 10^{-4}, we achieved 81% generalization accuracy, which is stable over multiple random seeds.

Table 3 compares different model variants. Standard Transformers with absolute and relative positional encoding perform similarly, with the relative positional variant having a slight advantage. Here Universal Transformers perform slightly worse.

4.5 Mathematics Dataset

We also test our approaches on subsets of Mathematics Dataset (Saxton et al., 2019). Since training models on the whole dataset is too resource-demanding, we only conduct experiments on two subsets: “place_value” and “add_or_sub”.

The results are shown in Table 3. While we can not directly compare our numbers with those reported in Saxton et al. (2019) (a single model is jointly trained on the whole dataset there), our results show that relative positional embedding is advantageous for the generalization ability on both subsets.

Table 3: Test accuracy of different Transformer (Trafo) variants on the considered datasets. See Sec. 4 for details. The last column shows previously reported accuracies. References: [1] Newman et al. (2020), [2] Keysers et al. (2020), [3] https://github.com/google-research/google-research/tree/master/cfq, [4] Hupkes et al. (2020), [5] Kim and Linzen (2020), [6] Saxton et al. (2019). Results marked with $*$ cannot be directly compared because of different training setups. $\sim$ denotes approximative numbers read from charts reported in previous works.

	Trafo	Uni. Trafo	Rel. Trafo	Rel. Uni. Trafo	Prior Work
SCAN (length cutoff=26)	0.30 ± 0.02	0.21 ± 0.01	0.72 ± 0.21	$\mathbf{1.00 \pm 0.00}$	$0.00^{[1]}$
CFQ Output length	0.57 ± 0.00	0.77 ± 0.02	0.64 ± 0.06	$\mathbf{0.81 \pm 0.01}$	$\sim 0.66^{[2]}$
CFQ MCD 1	$\mathbf{0.40 \pm 0.01}$	0.39 ± 0.03	0.39 ± 0.01	0.39 ± 0.04	$0.37 \pm 0.02^{[3]}$
CFQ MCD 2	0.10 ± 0.01	0.09 ± 0.02	0.09 ± 0.01	$\mathbf{0.10 \pm 0.02}$	$0.08 \pm 0.02^{[3]}$
CFQ MCD 3	0.11 ± 0.00	0.11 ± 0.01	0.11 ± 0.01	$\mathbf{0.11 \pm 0.03}$	$0.11 \pm 0.00^{[3]}$
CFQ MCD mean	0.20 ± 0.14	0.20 ± 0.14	0.20 ± 0.14	$\mathbf{0.20 \pm 0.14}$	$0.19 \pm 0.01^{[2]}$
PCFG Productivity split	0.65 ± 0.03	0.78 ± 0.01	-	$\mathbf{0.85 \pm 0.01}$	$0.50 \pm 0.02^{[4]}$
PCFG Systematicity split	0.87 ± 0.01	0.93 ± 0.01	0.89 ± 0.02	$\mathbf{0.96 \pm 0.01}$	$0.72 \pm 0.00^{[4]}$
COGS	0.80 ± 0.00	0.78 ± 0.03	$\mathbf{0.81 \pm 0.01}$	0.77 ± 0.01	$0.35 \pm 0.06^{[5]}$
Math: add_or_sub	0.89 ± 0.01	0.94 ± 0.01	0.91 ± 0.03	$\mathbf{0.97 \pm 0.01}$	$\sim 0.91^{[6]*}$
Math: place_value	0.12 ± 0.07	0.20 ± 0.02	-	$\mathbf{0.75 \pm 0.10}$	$\sim 0.69^{[6]*}$

5 Related Work

Many recent papers focus on improving generalization on the SCAN dataset. Some of them develop specialized architectures (Korrel et al., 2019; Li et al., 2019; Russin et al., 2019; Gordon et al., 2020; Herzig and Berant, 2020) or data augmentation methods (Andreas, 2020), others apply meta-learning (Lake, 2019). As an alternative, the CFQ dataset proposed in (Keysers et al., 2020) is gaining attention recently (Guo et al., 2020; Furrer et al., 2020). Mathematical problem solving has also become a popular domain for testing generalization of neural networks (Kaiser and Sutskever, 2016; Schlag et al., 2019; Charton et al., 2021). The PCFG (Hupkes et al., 2020) and COGS (Kim and Linzen, 2020) are also datasets proposed relatively recently. Despite increasing interests in systematic generalization tasks, interestingly, no prior work has questioned the baseline configurations which could be overfitted to the machine translation tasks.

Generalizing to longer sequences have been proven to be especially difficult. Currently only hybrid task-specific neuro-symbolic approaches can solve it (Nye et al., 2020; Chen et al., 2020; Liu et al., 2020). In this work, we focus on a subproblem required for length generalization: the EOS decision problem (Newman et al., 2020), and we show that it can be mitigated by using relative positional embeddings.

The study of generalization ability of neural networks at different stages of training has been a general topic of interest (Nakkiran et al., 2019; Roelofs, 2019). Our analysis has shown that this question is particularly relevant to the problem of systematic generalization, as demonstrated by large performance gaps in our experiments, which has not been discussed in prior work.

Prior work proposed several sophisticated initialization methods for Transformers (Zhang et al., 2019; Zhu et al., 2021), e.g. with a purpose of removing the layer normalization components (Huang et al., 2020). While our work only revisited basic scaling methods, we demonstrated their particular importance for systematic generalization.

In recent work,[5] Ontañón et al. (2021) have also focused on improving the compositional generalization abilities of Transformers. In addition to relative positional encodings and Universal Transformers, novel architectural changes such as "copy decoder" as well as dataset-specific "intermediate representations" (Herzig et al., 2021) have been studied. However, other aspects we found crucial, such as early stopping, scaling of the positional embeddings, and the validation set issues have not been considered. In consequence, our models achieve substantially higher performance than the best results reported by Ontañón et al. (2021) across all standard datasets: PCFG, COGS, and CFQ (without intermediate representations).

Finally, our study focused on the basic Trans-

[5] Our work was submitted to EMNLP 2021 on May 17, 2021 and has been under the anonymity period until Aug. 25. Ontañón et al. (2021) appeared on arXiv on Aug. 9, 2021.

former architectures. However, the *details* discussed above in the context of algorithmic tasks should also be relevant for other Transformer variants and fast weight programmers (Schmidhuber, 1992; Schlag et al., 2021; Irie et al., 2021), as well as other architectures specifically designed for algorithmic reasoning (Graves et al., 2016; Kaiser and Sutskever, 2016; Csordás and Schmidhuber, 2019; Freivalds et al., 2019).

6 Conclusion

In this work we showed that the performance of Transformer architectures on many recently proposed datasets for systematic generalization can be greatly improved by revisiting basic model and training configurations. Model variants with relative positional embedding often outperform the ones with absolute positional embedding. They also mitigate the EOS decision problem, an important problem previously found by Newman et al. (2020) when considering the length generalization of neural networks. This allows us to focus on the problem of compositions in the future, which is the remaining problem for the length generalization.

We also demonstrated that reconsidering early stopping and embedding scaling can greatly improve baseline Transformers, in particular on the COGS and PCFG datasets. These results shed light on the discrepancy between the model performance on the IID validation set and the test accuracy on the systematically different generalization split. As consequence, currently common practice of validating models on the IID dataset is problematic. We conclude that the community should discuss proper ways to develop models for systematic generalization. In particular, we hope that our work clearly demonstrated the necessity of a validation set for systematic generalization in order to establish strong baselines and to avoid a false sense of progress.

Acknowledgments

We thank Aleksandar Stanić and Imanol Schlag for their helpful comments and suggestions on an earlier version of the manuscript. This research was partially funded by ERC Advanced grant no: 742870, project AlgoRNN, and by Swiss National Science Foundation grant no: 200021_192356, project NEUSYM. We thank hardware donations from NVIDIA & IBM.

References

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proc. Association for Computational Linguistics (ACL)*, pages 7556–7566, Virtual only.

Dzmitry Bahdanau, Harm de Vries, Timothy J O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron Courville. 2019. CLOSURE: Assessing systematic generalization of CLEVR models. In *ViGIL workshop, NeurIPS*, Vancouver, Canada.

Francois Charton, Amaury Hayat, and Guillaume Lample. 2021. Learning advanced mathematical computations from examples. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only.

Róbert Csordás and Jürgen Schmidhuber. 2019. Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. 2021. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proc. Association for Computational Linguistics (ACL)*, pages 2978–2988, Florence, Italy.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA.

Jerry Fodor and Brian P McLaughlin. 1990. Connectionism and the problem of systematicity: Why smolensky's solution doesn't work. *Cognition*, 35(2):183–204.

Jerry A Fodor, Zenon W Pylyshyn, et al. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71.

Karlis Freivalds, Emils Ozolins, and Agris Sostaks. 2019. Neural shuffle-exchange networks - sequence processing in o(n log n) time. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *Preprint arXiv:2007.08970*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, Sardinia, Italy.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2020. Permutation equivariant models for compositional generalization in language. In *Int. Conf. on Learning Representations (ICLR)*, Addis Ababa, Ethiopia.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. In *Workshop on Representation Learning, ICML*, Edinburgh, Scotland.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. In *Int. Conf. on Learning Representations (ICLR) Workshop Track*, Vancouver, Canada.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gomez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John P. Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. 2020. On the binding problem in artificial neural networks. *Preprint arXiv:2012.05208*.

Yinuo Guo, Zeqi Lin, Jian-Guang Lou, and Dongmei Zhang. 2020. Hierarchical poset decoding for compositional generalization in language. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, pages 1026–1034, Santiago, Chile.

Jonathan Herzig and Jonathan Berant. 2020. Span-based semantic parsing for compositional generalization. *Preprint arXiv:2009.06040*.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *Preprint arXiv:2104.07478*.

Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. 2020. Improving transformer optimization through better initialization. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 4475–4483, Virtual only.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, pages 757–795.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. 2021. Going beyond linear transformers with recurrent fast weight programmers. *Preprint arXiv:2106.06295*.

Lukasz Kaiser and Ilya Sutskever. 2016. Neural GPUs learn algorithms. In *Int. Conf. on Learning Representations (ICLR)*, San Juan, Puerto Rico.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *Int. Conf. on Learning Representations (ICLR)*, Addis Ababa, Ethiopia.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Virtual only.

Kris Korrel, Dieuwke Hupkes, Verna Dankers, and Elia Bruni. 2019. Transcoding compositionally: Using attention to find more generalizable solutions. In *Proc. BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, ACL*, pages 1–11, Florence, Italy.

Brenden M Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 9788–9798, Vancouver, Canada.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 2873–2882, Stockholm, Sweden.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. Compositional generalization for primitive substitutions. In *Proc. Conf. on Empirical Methods in Natural Language Processing and Int.Joint Conf. on Natural Language Processing (EMNLP-IJCNLP)*, pages 4292–4301, Hong Kong, China.

Adam Liska, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? searching for a compositional RNN in a haystack. In *AEGAP Workshop ICML*, Stockholm, Sweden.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2019. Deep double descent: Where bigger models and more data hurt. In *Int. Conf. on Learning Representations (ICLR)*, Addis Ababa, Ethiopia.

Benjamin Newman, John Hewitt, Percy Liang, and Christopher D Manning. 2020. The eos decision and length extrapolation. In *Proc. BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, EMNLP*, pages 276–291, Virtual only.

Maxwell I Nye, Armando Solar-Lezama, Joshua B Tenenbaum, and Brenden M Lake. 2020. Learning compositional rules via neural program synthesis. *Preprint arXiv:2003.05562*.

Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. 2021. Making transformers solve compositional tasks. *Preprint arXiv:2108.04378*.

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. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, Vancouver, Canada.

Rebecca Roelofs. 2019. *Measuring Generalization and overfitting in Machine learning*. Ph.D. thesis, UC Berkeley.

Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *Preprint arXiv:1904.09708*.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 139, pages 9355–9366, Virtual only.

Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jürgen Schmidhuber, and Jianfeng Gao. 2019. Enhancing the transformer with explicit relational encoding for math problem solving. *Preprint arXiv:1910.06611*.

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

Jürgen Schmidhuber. 2012. Self-delimiting neural networks. *Preprint arXiv:1210.0118*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *Preprint arXiv:2010.12725*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proc. North American Chapter of the Association for Computational Linguistics on Human Language Technologies (NAACL-HLT)*, pages 464–468, New Orleans, Louisiana, USA.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, Montréal, Canada.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, Long Beach, CA, USA.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2019. Improving deep transformer with depth-scaled initialization and merged attention. In *Proc. Conf. on Empirical Methods in Natural Language Processing and Int.Joint Conf. on Natural Language Processing (EMNLP-IJCNLP)*, pages 898–909, Hong Kong, China.

Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. 2021. Gradinit: Learning to initialize neural networks for stable and efficient training. *Preprint arXiv:2102.08098*.

A Evaluation Metrics

For all tasks, accuracy is computed on the sequence-level, i.e. all tokens in the sequence should be correct for the output to be counted as correct. For the losses, we always report the average token-wise cross entropy loss.

B Hyperparameters

For all of our models we use an Adam optimizer with the default hyperparameters of PyTorch (Paszke et al., 2019). We only change the learning rate. We use dropout with probability of 0.1 after each component of the transformer: both after the attention heads and linear transformations. We specify the dataset-specific hyperparameters in Table 4. For all Universal Transformer experiments, we use both the "No scaling" and the "Positional Embedding Downscaling" methods. For the standard Transformers with absolute positional embedding we test different scaling variants on different datasets shown in Table 6. When multiple scaling methods are available, we choose the best performing ones when reporting results in Table 3. We always use the same number of layers for both encoder and decoder. The embedding and the final softmax weights of the decoder are always shared (tied embeddings).

The number of parameters for different models and the corresponding to representative execution time is shown in Table 5.

C Relative Positional Embedding

We use the relative positional embedding variant of self attention from Dai et al. (2019). Here, we use a decomposed attention matrix of the following form:

$$\boldsymbol{A}^{\text{rel}}_{i,j} = \underbrace{\boldsymbol{H}_i^\top \boldsymbol{W}_q^\top \boldsymbol{W}_{k,E} \boldsymbol{H}_j}_{(a)} + \underbrace{\boldsymbol{H}_i^\top \boldsymbol{W}_q^\top \boldsymbol{W}_{k,P} \boldsymbol{P}_{i-j}}_{(b)} + \underbrace{\boldsymbol{u}^\top \boldsymbol{W}_{k,E} \boldsymbol{H}_j}_{(c)} + \underbrace{\boldsymbol{v}^\top \boldsymbol{W}_{k,P} \boldsymbol{P}_{i-j}}_{(d)}$$

where $\boldsymbol{H}_i$ is the hidden state of the i^{th} column of the Transformer, $\boldsymbol{P}_i$ is an embedding for position (or in this case distance) i. Matrix $\boldsymbol{W}_q$ maps the states to queries, $\boldsymbol{W}_{k,E}$ maps states to keys, while $\boldsymbol{W}_{k,P}$ maps positional embedding to keys. $\boldsymbol{u}$ and $\boldsymbol{v}$ are learned vectors. Component (a) corresponds to content-based addressing, (b) to content based relative positional addressing, (c) represents a global content bias, while (d) represents a global position bias.

We use sinusoidal positional embedding $\boldsymbol{P}_i \in \mathbb{R}^{d_{\text{model}}}$. The relative position, i, can be both positive and negative. Inspired by Vaswani et al. (2017), we define $\boldsymbol{P}_{i,j}$ as:

$$\boldsymbol{P}_{i,j} = \begin{cases} \sin(i/10000^{2j/d_{\text{model}}}), & \text{if } j = 2k \\ \cos(i/10000^{2j/d_{\text{model}}}) & \text{if } j = 2k+1 \end{cases} \tag{1}$$

Prior to applying the softmax, $\boldsymbol{A}^{\text{rel}}_{i,j}$ is scaled by $\frac{1}{\sqrt{d_{\text{model}}}}$, as in Vaswani et al. (2017).

We never combine absolute with relative positional embedding. In case of a relative positional variant of any Transformer model, we do not add absolute positional encoding to the word embeddigs. We use relative positional attention in every layer, except at the interface between encoder and decoder, where we use the standard formulation from Vaswani et al. (2017), without adding any positional embedding.

D Embedding Scaling

In this section, we provide full descriptions of embedding scaling strategies we investigated. In the following, w_i denotes the word index at input position i, $\boldsymbol{E}_w \in \mathbb{R}^{d_{\text{model}}}$ denotes learned word embedding for word index w. Positional embedding for position i is defined as in Eq. 1.

Position Embedding Downscaling. Vaswani et al. (2017) combine the input word and positional embeddings for each position i as $\boldsymbol{H}_i = \sqrt{d_{\text{model}}}\boldsymbol{E}_{w_i} + \boldsymbol{P}_i$. Although in the original paper, the initialization of $\boldsymbol{E}$ is not discussed, most implementations use Glorot initialization (Glorot and Bengio, 2010), which in this case means that each component of $\boldsymbol{E}$ is drawn from $\mathcal{U}(-\sqrt{\frac{6}{d_{\text{model}}+N_{\text{words}}}}, \sqrt{\frac{6}{d_{\text{model}}+N_{\text{words}}}})$ where $\mathcal{U}(a,b)$ represents the uniform distribution in range $[a,b]$.

No scaling. This corresponds to how PyTorch initializes embedding layers by default: each element of $\boldsymbol{E}$ is drawn from $\mathcal{N}(0,1)$. $\mathcal{N}(\mu,\sigma)$ is the normal distribution with mean μ and standard deviation of σ. The word embeddings are combined with the positional embeddings without any scaling: $\boldsymbol{H}_i = \boldsymbol{E}_{w_i} + \boldsymbol{P}_i$

Table 4: Hyperparameters used for different tasks. We denote the feedforward size as d_{FF}. For the learning rate of CFQ (denoted by *), the learning rate seemingly differs from Keysers et al. (2020). In fact, although Keysers et al. (2020) use Noam learning rate scheduling, scaling by $\frac{1}{\sqrt{d_{\text{model}}}}$ is not used, so we had to compensate for this to make them functionally equivalent.

	d_{model}	d_{FF}	n_{head}	n_{layers}	batch size	learning rate	warmup	scheduler
SCAN	128	256	8	3	256	10^{-3}	-	-
CFQ - Non-universal	128	256	16	2	4096	0.9*	4000	Noam
CFQ - Universal	256	512	4	6	2048	2.24*	8000	Noam
PCFG	512	2048	8	6	64	10^{-4}	-	-
COGS	512	512	8	2	128	10^{-4}	-	-
COGS Noam	512	512	8	2	128	2	4000	Noam
Mathematics	512	2048	8	6	256	10^{-4}	-	-

Table 5: Model sizes and execution times. One representative split is shown per dataset. Other splits have the same number of parameters, and their execution time is in the same order of magnitude.

Dataset	Model	No. of params	Execution time	GPU type
SCAN	Standard	992k	1:30	Titan X Maxwell
	Universal	333k	1:15	
	Relative Pos.	1.1M	1:45	
	Universal, Relative Pos.	366k	1:30	
CFQ MCD 2	Standard	685k	10:00	Tesla V100-SXM2-32GB-LS
	Universal	1.4M	12:00	
	Relative Pos.	751k	14:15	
	Universal, Relative Pos.	1.5M	14:00	
PCFG Systematicity	Standard	44.7M	20:30	Tesla V100-PCIE-16GB
	Universal	7.9M	17:00	
	Relative Pos.	47.8M	21:30	
	Universal, Relative Pos.	8.4M	21:30	
COGS	Standard	9.3M	17:30	Tesla V100-SXM2-32GB-LS
	Universal	5.1M	17:15	
	Relative Pos.	10.3M	21:00	
	Universal, Relative Pos.	5.6M	20:00	
Math: add_or_sub	Standard	4.4M	8:00	Tesla P100-SXM2-16GB
	Universal	7.4M	7:30	
	Relative Pos.	4.7M	8:30	
	Universal, Relative Pos.	7.9M	8:00	

Table 6: Scaling types used for standard transformers with absolute positional embedding on different datasets. TEU denotes Token Embedding Upscaling, PED denotes Position Embedding Downscaling.

	TEU	No scaling	PED
SCAN		✓	✓
CFQ MCD		✓	✓
CFQ Length	✓	✓	✓
PCFG Productivity	✓	✓	✓
PCFG Systematicity	✓	✓	✓
COGS	✓	✓	✓
Mathematics		✓	✓

Token Embedding Upscaling. We propose to use Kaiming initialization (He et al., 2015) for the word embeddings: each element of $\boldsymbol{E} \sim \mathcal{N}(0, \frac{1}{\sqrt{d_{\text{model}}}})$. Instead of scaling up the word embeddings, the positional embeddings are scaled down: $\boldsymbol{H}_i = \boldsymbol{E}_{w_i} + \frac{1}{\sqrt{d_{\text{model}}}}\boldsymbol{P}_i$

E Analyzing the Positively Correlated Loss and Accuracy

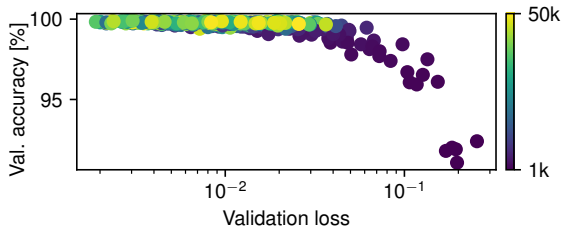
In Sec. 3.2, we reported that on the generalization splits of some datasets both the accuracy and the loss grows together during training. Here we further analyze this behavior in Figure 5 (see the caption).

F Accuracies on the IID Split

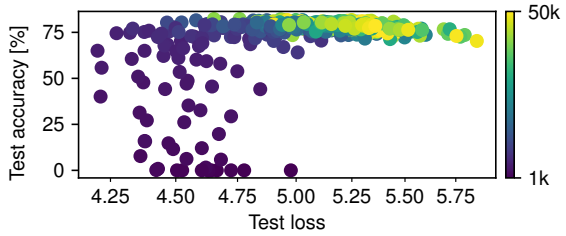
To show that the IID accuracy does not provide any useful signal for assessing the quality of the final model, we report IID accuracies of the models from Table 3 in Table 8. We only show datasets for which an IID validation set is available in the same split

Table 7: Test accuracy of different Transformer (Trafo) variants and different initializations on the considered datasets. This is a more detailed version of Table 3, with detailed scores for all initialization variants. The last column shows previously reported accuracies. References: [1] Newman et al. (2020), [2] Keysers et al. (2020), [3] https://github.com/google-research/google-research/tree/master/cfq, [4] Hupkes et al. (2020), [5] Kim and Linzen (2020), [6] Saxton et al. (2019). Results marked with $*$ cannot be directly compared because of different training setups. $\sim$ denotes imprecise numbers read from charts in prior works. For the configuration marked by †, the results are obtained by running 8 seeds from which 3 crashed, resulting in 5 useful runs reported below. Crashed runs suddenly drop their accuracy to 0, which never recovers during the training. The reason for the crashing is the overly big learning rate (2.24, from the baseline). We run another 10 seeds with learning rate of 2.0, obtaining similar final accuracy of 0.75 ± 0.02, but without any crashed runs.

	Init	Trafo	Uni. Trafo	Rel. Trafo	Rel. Uni. Trafo	Reported
SCAN (length cutoff=26)	PED	**0.30 ± 0.02**	**0.21 ± 0.01**	-	-	0.00[1]
	No scaling	0.15 ± 0.07	0.14 ± 0.05	**0.72 ± 0.21**	**1.00 ± 0.00**	
CFQ Output length	PED	0.56 ± 0.02	0.60 ± 0.34	-	-	$\sim$ 0.66[2]
	TEU	**0.57 ± 0.00**	0.74 ± 0.02 †	-	-	
	No scaling	0.53 ± 0.04	**0.77 ± 0.02**	**0.64 ± 0.06**	**0.81 ± 0.01**	
CFQ MCD 1	PED	0.36 ± 0.02	0.37 ± 0.05	-	-	0.37 ± 0.02[3]
	No scaling	**0.40 ± 0.01**	**0.39 ± 0.03**	**0.39 ± 0.01**	**0.39 ± 0.04**	
CFQ MCD 2	PED	0.08 ± 0.01	**0.09 ± 0.01**	-	-	0.08 ± 0.02[3]
	No scaling	**0.10 ± 0.01**	**0.09 ± 0.02**	**0.09 ± 0.01**	**0.10 ± 0.02**	
CFQ MCD 3	PED	0.10 ± 0.00	**0.11 ± 0.00**	-	-	0.11 ± 0.00[3]
	No scaling	**0.11 ± 0.00**	**0.11 ± 0.01**	**0.11 ± 0.01**	**0.11 ± 0.03**	
CFQ MCD mean	PED	0.18 ± 0.13	0.19 ± 0.14	-	-	0.19 ± 0.01[2]
	No scaling	**0.20 ± 0.14**	**0.20 ± 0.14**	**0.20 ± 0.14**	**0.20 ± 0.14**	
PCFG Productivity split	PED	**0.65 ± 0.03**	**0.78 ± 0.01**	-	-	0.50 ± 0.02[4]
	TEU	0.47 ± 0.27	**0.78 ± 0.01**	-	-	
	No scaling	0.63 ± 0.02	0.76 ± 0.01	-	**0.85 ± 0.01**	
PCFG Systematicity split	PED	**0.87 ± 0.01**	**0.93 ± 0.01**	-	-	0.72 ± 0.00[4]
	TEU	0.75 ± 0.08	0.92 ± 0.01	-	-	
	No scaling	0.86 ± 0.02	0.92 ± 0.00	**0.89 ± 0.02**	**0.96 ± 0.01**	
COGS	PED	**0.80 ± 0.00**	0.77 ± 0.02	-	-	0.35 ± 0.06[5]
	TEU	0.78 ± 0.03	**0.78 ± 0.03**	-	-	
	No scaling	0.62 ± 0.06	0.51 ± 0.07	**0.81 ± 0.01**	**0.77 ± 0.01**	
Math: add_or_sub	PED	0.80 ± 0.01	0.92 ± 0.02	-	-	$\sim$ 0.91[6]$*$
	No scaling	**0.89 ± 0.01**	**0.94 ± 0.01**	**0.91 ± 0.03**	**0.97 ± 0.01**	
Math: place_value	PED	0.00 ± 0.00	**0.20 ± 0.02**	-	-	$\sim$ 0.69[6]$*$
	No scaling	**0.12 ± 0.07**	0.12 ± 0.01	-	**0.75 ± 0.10**	



(a) COGS: IID Validation set



(b) COGS: Generalization test set

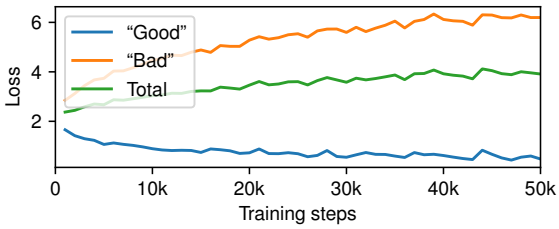
Figure 4: Relationship between the loss and accuracy on (a) IID validation set and (b) the generalization test set on COGS (it comes without a validation set for the generalization splits). Standard Transformers are used. The color shows the training step. Five runs are shown. The loss is shown on a logarithmic scale. On the IID validation set (a), the accuracy increases when the loss decreases, as expected. In contrast, on the generalization split (b), high accuracy corresponds to higher loss. For generalization validation loss versus generalization accuracy on CFQ MCD 1, see Figure 2. For the analysis of the underlying reason, see Figure 5.

as the one reported in Table 3. This complements the IID and generalization accuracies on COGS and PCFG with different embedding scalings we reported in Table 2. With the exception of standard Transformer on PCFG and the "place_value" module of the Mathematics dataset, all other validation accuracies are 100%, while their generalization accuracy vary wildly.

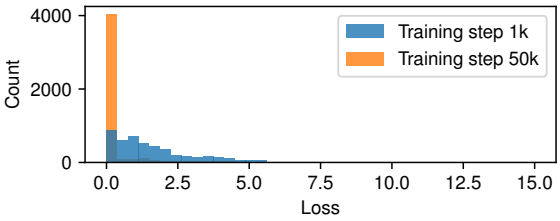
G Additional Results

Figure 4 shows that both the test loss and accuracy grows on COGS dataset during training. Additionally, it shows the expected, IID behavior on the same dataset for contrast.

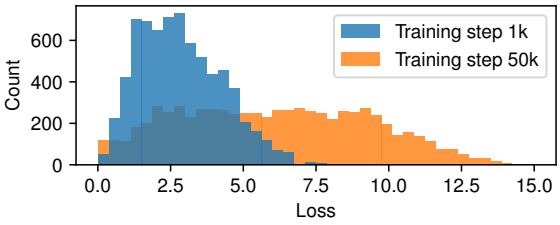
Figure 6 shows the relative change in convergence speed when using relative positional embeddings.



(a) Decomposed loss



(b) Histogram of "good" loss (first and last measurement)



(c) Histogram of "bad" loss (first and last measurement)

Figure 5: Analysis of the growing test loss on the systematically different test set on CFQ MCD 1 split. We measure the loss individually for each sample in the test set. We categorize samples as "good" if the network output on the corresponding input matched the target exactly any point during the training, and as "bad" otherwise. (a) The total loss (increasing) can be decomposed to the loss of the "good" samples (decreasing), and the loss of the "bad" samples (increasing). (b, c) The histogram of the loss for the "good" and "bad" samples at the beginning and end of the training. The loss of the "good" samples concentrates near zero, while the "bad" samples spread out and the corresponding loss can be very high. The net effect is a growing total loss.

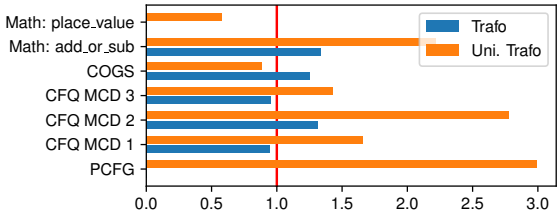


Figure 6: Relative change in convergence speed by using relative positional embeddings instead of absolute. Convergence speed is measured as the mean number of steps needed to achieve 80% of the final performance of the model. Relative variants usually converge faster. Universal Transformers benefit more than the non-universal ones. The non-universal variants are not shown for PCFG and "Math: place_value", because the relative variants do not converge (see Sec. 3.1).

Table 8: IID validation accuracy for datasets where IID test set is available. CFQ and PCFG are not shown because they require the model to be trained on a separate, IID split. The other settings correspond to Table 3 in the main text. Generalization split test accuracies are shown in parenthesis for easy comparison.

	Transformer	Uni. Transformer	Rel. Transformer	Rel. Uni. Transformer
SCAN (length cutoff=26)	**1.00 ± 0.00** (0.30)	**1.00 ± 0.00** (0.21)	**1.00 ± 0.00** (0.72)	**1.00 ± 0.00** (1.00)
COGS	**1.00 ± 0.00** (0.80)	**1.00 ± 0.00** (0.78)	**1.00 ± 0.00** (0.81)	**1.00 ± 0.00** (0.77)
Math: add_or_sub	**1.00 ± 0.00** (0.89)	**1.00 ± 0.00** (0.94)	**1.00 ± 0.00** (0.91)	**1.00 ± 0.00** (0.97)
Math: place_value	0.80 ± 0.45 (0.12)	**1.00 ± 0.00** (0.20)	-	**1.00 ± 0.00** (0.75)

Table 9: Accuracy of different Transformer variants on CFQ. "Big" variant has a batch size of 4096, and is trained with Noam scheduler (learning rate 0.9). "Small" variant has a batch size of 512 and a fixed learning rate of 10^{-4}. The ratio of accuracies of "small" and "big" variants are also shown in the "Ratio" column, indicating the relative performance drop caused by decreasing the batch size. Relative variants experience less accuracy drop.

	Variant	Transformer	Rel. Transformer	Uni. Transformer	Rel. Uni. Transformer
CFQ MCD 1	Big	0.40 ± 0.01	0.39 ± 0.02	0.41 ± 0.03	0.42 ± 0.02
	Small	0.26 ± 0.02	0.32 ± 0.01	0.28 ± 0.00	0.36 ± 0.01
	Ratio	0.65	0.80	0.68	**0.85**
CFQ MCD 2	Big	0.10 ± 0.01	0.09 ± 0.01	0.09 ± 0.00	0.09 ± 0.02
	Small	0.05 ± 0.01	0.07 ± 0.01	0.04 ± 0.01	0.10 ± 0.01
	Ratio	0.51	0.76	0.50	**1.05**
CFQ MCD 3	Big	0.11 ± 0.00	0.11 ± 0.01	0.11 ± 0.01	0.12 ± 0.02
	Small	0.09 ± 0.00	0.09 ± 0.00	0.09 ± 0.01	0.11 ± 0.01
	Ratio	0.80	0.85	0.85	**0.98**
CFQ Out. len.	Big	0.57 ± 0.02	0.64 ± 0.04	0.76 ± 0.03	0.81 ± 0.02
	Small	0.41 ± 0.03	0.51 ± 0.02	0.55 ± 0.02	0.70 ± 0.03
	Ratio	0.72	0.80	0.73	**0.87**

Table 10: Dataset statistics. "#" denotes number of samples. Vocabulary size shows the union of input and output vocabularies. Train and test length denotes the maximum input/output length in the train and test set, respectively.

Dataset	# train	# IID valid.	# gen. test	# gen. valid.	Voc. size	Train len.	Test len.
Scan (length cutoff=26)	16458	1828	2624	-	19	9/26	9/48
CFQ MCD 1	95743	-	11968	11968	181	29/95	30/103
CFQ MCD 2	95743	-	11968	11968	181	29/107	30/91
CFQ MCD 3	95743	-	11968	11968	181	29/107	30/103
CFQ Output Length	100654	-	9512	9512	181	29/77	29/107
PCFG Productivity	81010	-	11333	-	535	53/200	71/736
PCFG Systematicity	82168	-	10175	-	535	71/736	71/496
COGS	24155	3000	21000	-	871	22/153	61/480
Math: add_or_sub	1969029	10000	10000	-	69	60/19	62/23
Math: place_value	1492268	9988	10000	-	69	50/1	52/1