A Minimal Model for Compositional Generalization on gSCAN

Alice Hein **Klaus Diepold**
TUM School of Computation, Information and Technology
Department of Computer Engineering
Technical University of Munich, Munich, Germany
{alice.hein, kldi}@tum.de

Abstract

Whether neural networks are capable of compositional generalization has been a topic of much debate. Most previous studies on this subject investigate the generalization capabilities of state-of-the-art deep learning architectures. We here take a more bottom-up approach and design a minimal model that displays generalization on a compositional benchmark, namely, the gSCAN dataset. The model is a hybrid architecture that combines layers trained with gradient descent and a selective attention mechanism optimized with an evolutionary strategy. The architecture has around 60 times fewer trainable parameters than models previously tested on gSCAN, and achieves comparable accuracies on most test splits, even when trained only on a fraction of the dataset. On adverb to verb generalization accuracy, it outperforms previous approaches by 65 to 86%. Through ablation studies, neuron pruning, and error analyses, we show that weight decay and attention mechanisms facilitate compositional generalization by encouraging sparse representations divorced from irrelevant context. We find that the model's sample efficiency can mainly be attributed to its selective attention mechanism.

1 Introduction

Compositionality is a core aspect of human cognition. It is what allows us to produce and understand infinite combinations of known concepts, be it in the realm of language, vision, or motor skills. Regarding artificial intelligence (AI) systems, compositionality holds the promise of more human-like, robust generalization on out-of-distribution data, as well as increased sample efficiency. Compositionality in neural networks has thus been the subject of numerous empirical investigations – with mixed results. Several studies using a variety of deep neural network architectures have found that models either failed on compositional tasks or succeeded given enough data, but could do so without relying on systematic compositional rules (Baroni, 2020; Lake and Baroni, 2018; Loula et al., 2018; Subramanian et al., 2019; Keysers et al., 2019; Hupkes et al., 2020; Andreas et al., 2019; Chaabouni et al., 2020). Others found that such architectures could reach compositional solutions without being explicitly constrained to do so, but that this ability varied dramatically across random initializations of the same model (Liška et al., 2018; McCoy et al., 2020; Weber et al., 2018).

The main focus of these studies has been on testing whether state-of-the-art deep learning architectures are able to learn compositionally. We here take a different approach, namely that of specifically building a minimal model that is able to solve a set of compositional generalization tasks, then using this model as a tool for analyzing when and how generalization occurs. Our dataset of choice for this investigation is gSCAN, a challenge benchmark for systematic generalization in grounded language understanding.

The model we use is a hybrid architecture, containing some weights that are trained with gradient descent, some that are optimized with an evolutionary strategy, and some that are initialized randomly and left frozen. A detailed justification of these design choices is given in Section 4.2. The architecture has around 60 times fewer trainable parameters than models previously tested on gSCAN, which allows us to run extensive ablation studies and error analyses to investigate factors contributing to generalization performance. We find that our best-performing model breaks down the gSCAN tasks into simpler, reusable parts and combines them using only 13 neurons in its final decision layer. It achieves accuracies comparable with previously proposed models on most test splits and outperforms them on adverb to verb generalization by 65 to 86%, even when trained on as little as 2% of the full dataset.

Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 1 - 15

2 Related Work

2.1 Compositional Generalization

A number of works have addressed the challenge of building AI systems that generalize compositionally. Neural Module Networks were designed for visual question answering and achieve systematicity by dynamically assembling question-specific models out of trainable reusable components (Andreas et al., 2016a,b). Other approaches explore ways of encouraging compositional representations in commonly used state-of-the-art models without major architectural changes. In this vein, Hupkes et al. (2018) and Baan et al. (2019) find that attentive guidance during training helps develop small functional groups of neurons that yield more compositional solutions by seq2seq models on lookup table tasks. Andreas (2020) and Akyürek et al. (2020) propose data augmentation schemes that promote compositional learning in instruction following and morphological analysis. Ontanon et al. (2022) focus on the effect that design decisions such as position encodings, weight sharing, or model hyper-parameters can have on the compositional generalization abilities of Transformer models. Finally Power et al. (2021) identify weight decay as being particularly effective at improving generalization on a binary operation table task.

2.2 Grounded instruction following

Several datasets have been proposed in recent years for training embodied agents to follow instructions in simulated 2D or 3D environments (Hermann et al., 2017; Yu et al., 2018a; Misra et al., 2018; Chaplot et al., 2018; Yu et al., 2018b; Deruyttere et al., 2019; Chevalier-Boisvert et al., 2019; Shridhar et al., 2020). One such task is gSCAN, which was specifically introduced as a benchmark for compositionality in grounded language understanding and contains 8 test splits for assessing different kinds of out-of-distribution generalization (Ruis et al., 2020). Previous approaches to solving gSCAN include language-conditioned message passing (Gao et al., 2020), compositional networks (Kuo et al., 2021), neuro-symbolic, dual-system models (Nye et al., 2021), and the introduction of auxiliary tasks (Jiang and Bansal, 2021; Heinze-Deml and Bouchacourt, 2020). The most successful model to date uses a general-purpose Transformer architecture with cross-modal attention and solves 5 out of 8 tasks (Qiu et al., 2021).

As outlined in the introduction, our goal is not necessarily to compete with these previous approaches. Instead we aim to devise a parameter-efficient model that can serve as a tool for a more in-depth investigation of the factors influencing performance on the different gSCAN test splits, and to contextualise the results with previous findings on out-of-distribution generalisation.

2.3 Neuroevolution

Evolutionary algorithms (EA) are stochastic, gradient-free methods that explore multiple areas of a search space in parallel. This work was particularly inspired by Tang et al. (2020), who combine neuroevolution techniques with self-attention to solve vision-based RL tasks. Their model extracts relevant patches from input images through a hard (non-differentiable) attention mechanism, optimized via an EA rather than more commonly used techniques like RL. The most attended-to patches are then passed on to an LSTM controller which determines the agent's action. The authors find that this approach significantly reduces the number of model parameters needed compared to previous methods, as well as offering increased interpretability and higher robustness to out-of-distribution modifications (Tang et al., 2020).

3 Background

Our architecture makes use of an Echo-State Network (ESN) and the covariance matrix adaptation evolution strategy (CMA-ES) to reduce the number of learnable parameters needed (see Section 4.2). As both are not commonly used in NLP, we here provide some background on these techniques.

3.1 Echo-State Networks

A basic ESN consists of an input layer W_i^r, a recurrent neural network (RNN) or so-called reservoir, and an output layer W_o. The reservoir's state is updated at each discrete time step as follows:

$$\begin{aligned}\mathbf{x}[n+1] =&(1-\alpha)\mathbf{x}[n] + \alpha f\big(W_i^r \mathbf{u}[n] \\ &+ W_r^r \mathbf{x}[n]\big),\end{aligned} \tag{1}$$

where α is a leak rate, $\mathbf{x}[n]$ is the current reservoir activation state, f is a the hyperbolic tangent function, $\mathbf{u}[n]$ is the external input, and W_r^r is the reservoir's internal weight matrix. The ESN's output is computed as

$$\mathbf{y}[n+1] = g(W_o \mathbf{x}[n+1]), \tag{2}$$

where g is an activation function. Crucially, W_i^r and W_r^r are randomly initialized and left untrained. Only W_o is optimized. This leads to considerably faster training times than for conventional RNNs where all weights are learned (Gauthier et al., 2021). ESNs' main areas of application therefore include resource-constrained contexts like robotics and edge computing (Nakajima, 2020).

3.2 CMA-ES

CMA-ES is a black-box optimization algorithm. It has been empirically shown to perform robustly on a range of tasks and requires very little parameter tuning (Hansen et al., 2010), making it the EA of choice for optimizing the model in Tang et al. (2020) which inspired our architecture. CMA-ES works by iteratively sampling λ candidate solutions from a multivariate normal distribution $\mathcal{N}(m, \sigma^2, C)$ with mean m, step size σ and covariance matrix C. At each generation, the candidate solutions' fitness is evaluated according to some function f, and m, σ, and C are adjusted to increase the probability of success. As the CMA-ES algorithm is not a main focus of this work, we relegate details on how the parameters are updated to Appendix A and refer the interested reader to Hansen and Ostermeier (2001) for a more in-depth description of the method.

4 Experiment setup

4.1 gSCAN Benchmark

The gSCAN environment is a grid with objects of various shapes, sizes, and colors. It is represented as a $16 \times 6 \times 6$ array, where 6 is the grid size and 16 is the dimension of the binary feature encoding for each grid cell. The agent receives synthetically generated English language instructions which it must carry out using 6 output actions, such as walking or turning. Some combinations are held out of the training set. Out-of-distribution generalization is then assessed on nine separate test splits, listed in Table 1, measured using exact match accuracy of predicted action sequences. The full dataset has $\approx 370,000$ training and $\approx 20,000$ test sequences. Hupkes et al. (2020) propose to distinguish between five interpretations of model compositionality, namely, the systematic recombination of known parts and rules (*systematicity*), the extension of predictions beyond lengths seen during training (*productivity*), robustness to synonym substitutions (*substitutivity*), dependence on local vs global structures (*localism*), and the preference for rules vs exceptions (*overgeneralization*). Following this taxonomy, split G tests the model's one-shot learning capabilities, or overgeneralization. Split I tests for productivity. We mainly consider splits B, C, D, E, F, and H, which focus on systematic generalization and substitutivity.

Table 1: Overview of gSCAN's compositional test splits

Test Split	**Held-out Examples**
A: Random	Random (in-distribution)
B: Yellow Squares	Yellow squares as targets if referred to as *yellow*
C: Red Squares	Red squares as targets
D: Novel Direction	Targets south-west of the agent
E: Relativity	Circles of size 2 referred to as *small* (references are relative to other grid objects, not tied to absolute sizes)
F: Class inference	Pushing squares of size 3 (*heavy* objects are pushed/pulled twice)
G: Adverb k = 1	All except *k* mentions of *cautiously* (looking both ways before each step)
H: Adverb to verb	Commands containing both *pull* and *while spinning* (turning 4 times)
I: Length	Action sequences of length ≥ 15

4.2 Model

To solve a gSCAN task, the agent requires knowledge of the command to carry out, the grid state, and its own past actions. The latter is needed to keep track of e.g. the number of turns completed when "spinning". In the following, we describe how these inputs are represented and processed.

Reservoir To create the representation of the language command we chose an ESN, due to its ability to capture information about all input words and their order in a single vector, without requiring any weight updates. This fit our goal of keeping the number of trainable parameters low. The instruction to the agent is tokenized, one-hot encoded, and input sequentially to a reservoir with 400 hidden neurons, which is updated after each token according to Equation 1. All reservoir neurons are randomly connected to an output layer W_o of size 64, yielding a 64-dimensional command embedding.

Selective attention The selective attention part of the model is responsible for extracting task-relevant information from the input grid. The command embedding $\mathbf{x}_{\text{lang}} \in \mathbb{R}^{1\times 64}$ is passed through a layer $W_{\text{lang}} \in \mathbb{R}^{64\times 16}$. The resulting vector is convolved with the input grid at each position to obtain a heatmap over grid $G \in \mathbb{R}^{16\times 6\times 6}$. The x-

and y-coordinates and the 16-dimensional feature vector for the most-attended grid cell $\mathbf{g}^*$ are then extracted:

$$\mathbf{g}^* = \arg\max\left((\mathbf{x}_{\text{lang}} \cdot W_{\text{lang}}) * G\right) \quad (3)$$

Because this $\arg\max$ operation is non-differentiable, we follow Tang et al. (2020)'s approach of using CMA-ES to optimize W_{lang}. However, in contrast to Tang et al., we apply the attention matrix to feature vectors rather than image patches, and we do not evolve all learnable parameters in our model. This is because our model has significantly more parameters than that of Tang et al. and the time and space complexity of CMA-ES is quadratic in the dimensionality of its objective function – restricting its application to problems with no more than a few hundred variables (Varelas et al., 2018). Therefore, only this selective attention part of the model is optimized using CMA-ES. The rest is trained using gradient descent. Inspired by joint attention mechanisms and parental guidance during child learning, the CMA-ES receives auxiliary feedback on whether the correct target object was most attended to. We also test and report the results for a version where the CMA-ES receives as feedback the cross-entropy loss produced by the agent's final prediction outputs (see Section 5.1).

Action attention The action attention part of the model serves as the agent's "memory" of past outputs. The command embedding undergoes self-attention, yielding a weighted embedding $\mathbf{a}_{\text{lang}} \in \mathbb{R}^{1\times 64}$. This is then passed through another attention layer $W_{\text{act}} \in \mathbb{R}^{64\times 200}$ and multiplied element-wise with a vector $\mathbf{x}_{\text{act}} \in \mathbb{R}^{200\times 1}$ containing the agent's one-hot encoded past 20 actions and orientations:

$$\mathbf{a}_{\text{act}} = (\mathbf{a}_{\text{lang}} \cdot W_{\text{act}}) \odot \mathbf{x}_{\text{act}} \quad (4)$$

As there is no $\arg\max$ operation involved, W_a is trained with conventional gradient descent.

Controller Finally, the outputs of the selective and action attention modules are concatenated with the agent's current x- and y-coordinates and orientation, as well as the unweighted command embedding and input to the agent's controller to predict the agent's next step. The controller consists of a layer normalization layer, a layer with 100 hidden ReLU units, and an output layer of size 6.

In total, the model has a little under $5 \cdot 10^4$ trainable parameters, compared to around $3 \cdot 10^6$ for models previously tested on gSCAN (Qiu et al., 2021). A schematic overview is shown in Figure 1.

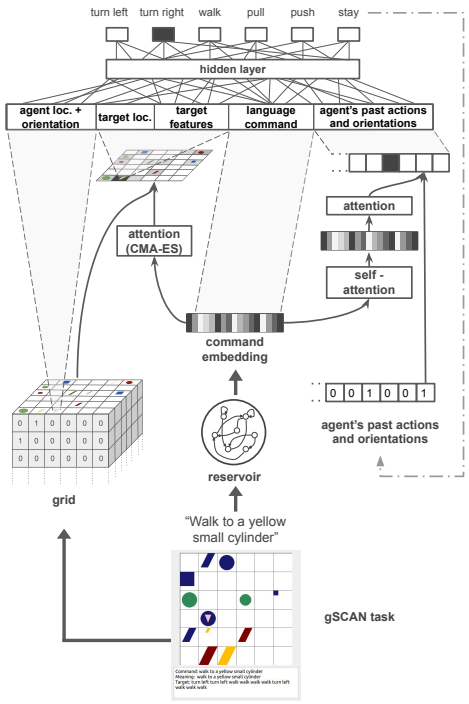


Figure 1: Schematic visualization of the proposed model

4.3 Training details

The weights of the ESN were initialized with a spectral radius of 0.99 and a density of $1e-2$. The leaking rate was set to $1e-1$. For the CMA-ES, we used a population size of 8 and an initial normal distribution with standard deviation $1e-1$. Optimization was implemented with the pycma library[1]. For the part of the model trained via gradient descent, we used the Adamax optimizer and a learning rate cycle with an upper boundary of $1e-2$. Weight decay was set to $1e-4$ and models were trained with batch size 4,096 for 100 epochs unless otherwise specified. All performance results are based on 10 runs. Each run used a different random seed for model weight initialization. However, the same 10 seeds were used for all tested modified or ablated architectures, so that all compared models started with the same 10 sets of weights. Experiments were implemented in Pytorch[2] and run on a server with 4 NVIDIA RTX 3090 GPUs and a 24 core Epyc CPU. The training time for one model was approximately 1.3 hours on the full

[1] pypi.org/project/cma/
[2] pytorch.org

dataset, 16 minutes on the 10% subset, and 9 minutes on the 2% subset. Code is publicly available at https://github.com/lemonk6/minmodgscan.

5 Results

5.1 Performance

As shown in Table 2, the model with auxiliary attention feedback reaches competitive accuracy on splits A, C, E, and F. On split H, it outperforms previous proposals by 65 to 86%. To see if generalization extended to other combinations, we also tested two custom splits. The first is a variation of task C, where not only red squares, but also yellow squares, green cylinders, and blue circles never appear as targets during training. The second is an extension of split H, where in addition to "pull while spinning", the agent is never told to "push while zigzagging" or to "walk hesitantly" during training. The model generalized to test sets containing only held-out shape-color and verb-adverb combinations, reaching 98.7% $\pm$ 1.5 and 98.9% $\pm$ 0.5 accuracy, respectively.

Table 3 compares the performance of models trained with and without an auxiliary feedback signal as well as models receiving perfect target location inputs, for reference. As can be seen, the model without an auxiliary signal does learn to focus on the target in some cases, but performance across the 10 runs exhibits a high variation. We also test a model which instead of absolute locations receives agent-centric row- and column-wise distances as input, which is sometimes used in RL goal navigation tasks. This stronger inductive bias seems to force the agent to more reliably employ the selective attention mechanism for target location, even when it only receives indirect feedback in the form of cross-entropy loss. Detailed evaluation results are given in B.

5.2 Sample Efficiency

One of the main advantages of our model is its sample efficiency. As shown in Figure 2, it achieves around 90% accuracy on splits A and C when trained on only 1% of the dataset, and 90 - 97% accuracy on splits A, C, E, and F with 2% of the data. This is well below the 40% data requirement threshold identified by Qiu et al. (2021) for their cross-modal transformer model. Interestingly, the exact match accuracy on splits B and C peaks at the 10% subset and declines slightly when given more data – something we take a closer look at in Section 5.3. Performance on task H increases more slowly than on other splits and requires at least 10% of the dataset to surpass 90% accuracy.

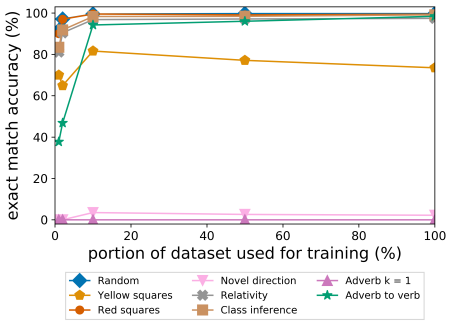


Figure 2: Sample efficiency on test splits for models with selective attention and auxiliary feedback

5.3 Error Analyses

Attention: We first analyze the mistakes made by the models trained without auxiliary feedback by treating the task of focusing on the correct target as a classification, and analyzing the feature-wise confusion matrices of the models. This reveals an accumulated false discovery rate of 66.5% for the "agent" dimension of the grid cell feature vectors, compared to 0% for the models trained with feedback. This means the models without attentive guidance tend to overly focus on the agent. The location of the agent does coincide with the target object's location around 18% of the time, which might lead to an overreliance on this dimension. We also find that the models trained without attention supervision struggle more with under-specified commands. For example, the models focus on an object of the correct color in ca. 96% of cases when the color is explicitly mentioned in the command. When the target object is only referred to by its shape or size, the accuracy drops to about 90%. Detailed confusion matrices can be found in D.1.

Yellow squares: In the case of split B, performance exhibits a large variation across instantiations of the same model. Out of 10 runs, approximately half always achieve accuracies in the range of 90 - 99% while the others only reach 35 - 55%. The best performance is achieved with a 10% subset of the training set, where all ten models reach at least 60% accuracy. A look at the confusion matrices shows that, on average, models correctly identify a square as their target object in 97% of test cases. However, their color accuracy is only around 75%. Taken together, this suggests that the

	Seq2Seq (2020)	GECA (2020)	Heinze (2020)	Gao (2020)	Kuo (2020)	Qiu (2021)	Jiang (2021)	Nye (2021)	Ours (100%)	Ours (10%)
A	97.69 ± 0.2	87.60 ± 1.2	94.19 ± 0.7	98.60 ± 1.0	96.73 ± 0.6	**99.95 ± 0.0**	-	74.7	99.7 ± 0.1	99.5 ± 0.1
B	54.96 ± 39.4	34.92 ± 39.3	86.45 ± 6.3	99.08 ± 0.7	94.91 ± 1.3	**99.90 ± 0.1**	-	81.3	73.5 ± 25.4	81.6 ± 14.3
C	23.51 ± 21.8	78.77 ± 6.6	81.07 ± 10.1	80.31 ± 24.5	67.72 ± 10.8	99.25 ± 0.9	-	78.1	99.4 ± 0.4	**99.5 ± 0.2**
D	0.00 ± 0.0	0.00 ± 0.0	-	0.16 ± 0.1	**11.52 ± 8.2**	0.00 ± 0.0	-	0.0	2.2 ± 1.5	3.5 ± 2.7
E	35.02 ± 2.4	33.19 ± 3.7	43.43 ± 7.0	87.32 ± 27.4	76.83 ± 2.3	**99.02 ± 1.2**	-	53.6	97.4 ± 2.0	96.8 ± 1.9
F	92.52 ± 6.8	85.99 ± 0.9	-	99.33 ± 0.5	98.67 ± 0.1	**99.98 ± 0.0**	-	76.2	99.1 ± 0.6	98.3 ± 1.7
G	0.00 ± 0.0	0.00 ± 0.0	-	-	1.14 ± 0.3	0.00 ± 0.0	**4.9**	0.00	0.00 ± 0.0	0.0 ± 0.1
H	22.70 ± 4.6	11.83 ± 0.3	-	33.6 ± 20.8	20.98 ± 1.4	22.2 ± 0.01	28.0	21.8	**98.4 ± 1.1**	94.2 ± 3.7

Table 2: Exact match accuracy on gSCAN compositional splits. For our proposed model, we report both the performance of models trained on the full dataset and of those trained on a 10% subset.

Table 3: Exact match accuracy and attention match accuracy on gSCAN compositional splits for models with selective attention, optimized with and without auxiliary feedback.

	perfect att.		**w/o aux. signal abs. loc.**				**w/o aux. signal rel. dist.**			
	seq. match		seq. match		att. match		seq. match		att. match	
A	100.0	± 0.0	59.3	± 29.1	74.2	± 21.4	83.0	± 3.4	92.8	± 2.2
B	100.0	± 0.0	50.8	± 21.1	61.6	± 17.0	59.5	± 15.7	70.0	± 16.7
C	100.0	± 0.0	70.0	± 29.5	73.8	± 24.8	89.7	± 9.3	91.1	± 8.4
D	1.9	± 1.7	0.1	± 0.2	66.6	± 28.9	0.8	± 0.9	91.3	± 2.6
E	100.0	± 0.0	50.3	± 20.4	62.1	± 17.9	74.1	± 6.2	84.1	± 7.6
F	100.0	± 0.0	52.6	± 25.0	70.4	± 20.7	67.5	± 9.3	84.4	± 8.0
G	0.0	± 0.0	0.0	± 0.0	63.0	± 15.7	0.0	± 0.0	73.0	± 5.8
H	99.3	± 1.0	37.5	± 20.2	74.4	± 14.9	56.4	± 6.2	89.9	± 3.9

models overfit to the absence of yellow squares. Depending on the random initialization of its selective attention matrix, a model may be more or less predisposed to generalization on this task. In the absence of any samples with yellow squares that could cause a course correction, this predisposition may be exacerbated with each update and thus deteriorate performance in the higher-data regimes.

Novel direction: Similar to previous architectures tested on gSCAN, our model has no trouble identifying the correct targets in split D (Ruis et al., 2020; Qiu et al., 2021). Its attention match accuracy is 100%. However, it cannot navigate to the identified target successfully. On average, it ends up in the correct row in 44% of cases, in the right column in 23% of cases, and never both.

5.4 Ablations

Weight Decay and Action Attention: As shown in Table 4, ablating weight decay or attention over past steps causes the most pronounced performance drops in splits E, F, and H. To compare structural differences between the ablated models, we perform a neuron pruning experiment (detailed results in C). For every neuron in the trained models' final hidden layer, we record the product of its activation and outgoing weights at each step when processing a 2% subset of the training set. We then disable neurons in ascending order of contribution to the models' outputs and assess the pruned model's exact match accuracy. All full models require only 13 hidden neurons to solve all tasks. Without attention over past actions, 16 neurons are needed to reach the final accuracy. Models without weight decay rely almost equally on all 100 neurons. Pruning any of them leads to decreased performance.

This difference in learned representations is also illustrated in Figure 3, which shows the weights between the agent's past actions and the hidden layer of three identically initialized models with different ablations applied. The model with weight decay and action attention learns the most sparse weights and focuses on recent steps. The hidden model without action attention has a similarly sparse hidden layer, but a longer "memory", i.e., it takes into account past actions from further back in the step sequence. The model without weight decay is very densely connected.

Selective Attention: To investigate the effect of selective attention, we train a soft attention version of the model. Instead of the isolated feature vector of the most attended grid cell, this model receives the attention-weighted whole grid as input, similar to the action attention mechanism. To account for the higher dimensionality of the input, we increase the number of neurons in the hidden units to 500. The relative amount of neurons needed to reach

	full model	**w/o weight decay**	**w/o action attention**	**w/o selective attention**
A	99.7 ± 0.1	92.5 ± 1.8	92.2 ± 2.5	89.6 ± 3.3
B	73.5 ± 25.4	74.2 ± 12.9	73.0 ± 21.1	69.5 ± 21.8
C	99.4 ± 0.4	95.9 ± 3.0	92.9 ± 7.6	78.6 ± 17.1
D	2.2 ± 1.5	0.1 ± 0.1	0.0 ± 0.0	0.3 ± 0.6
E	97.4 ± 2.0	73.9 ± 8.2	85.7 ± 6.6	72.1 ± 2.3
F	99.1 ± 0.6	73.7 ± 7.8	80.6 ± 9.3	81.6 ± 9.9
G	0.0 ± 0.0	0.4 ± 0.2	0.0 ± 0.0	0.0 ± 0.0
H	98.4 ± 1.1	39.5 ± 14.5	23.8 ± 3.7	65.5 ± 13.1

Table 4: Exact match accuracy on gSCAN compositional splits for ablated models

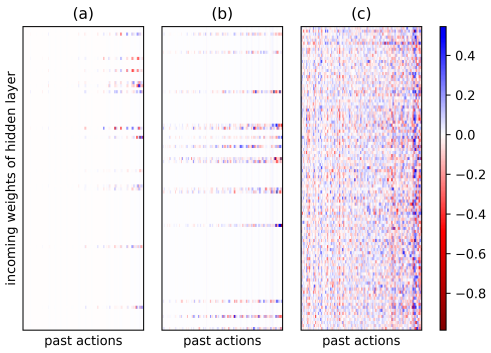


Figure 3: Weights between the agent's past actions and the model's hidden layer, as learned by (a) the full model, (b) the model with weight decay but no action attention, and (c) the model with action attention but no weight decay

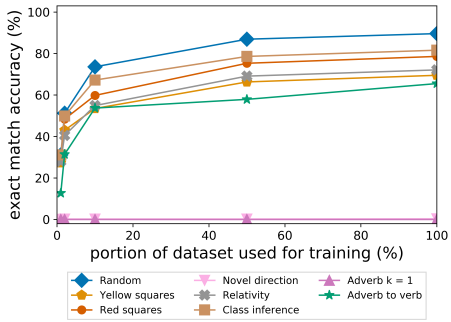


Figure 4: Sample efficiency on test splits for models without selective attention

full accuracy is similar to the model without action attention – around 18%. Performance-wise, the ablation causes a drop-off across the board but still achieves around 90% accuracy on in-distribution data when trained on the full dataset. However, the sample efficiency is greatly reduced (see Figure 4). I.e., models need to have seen a greater number of input combinations to start generalizing. This is also supported by a comparison of the confusion matrices for models with and without selective attention via a χ^2-test on split A (details in D.2). By far the most over-represented feature among misclassifications by the soft-attention model, as measured by standardized residuals, is the "square" dimension. Since squares are held out for splits B, C, and F, this shape is underrepresented in the training set. The model thus sees fewer examples during training, which seems to affect its ability to generalize to new combinations involving squares even for in-distribution data.

5.5 "Spontaneous" Generalization

During our ablation studies, we observed that generalization to the "adverb to verb" split did occur frequently in models without weight decay and action attention, but not in a linear fashion. As shown in Figure 5, performance on split H would spike on one training batch, then fall again. Higher systematic generalization ability is not necessarily evident from looking at the performance on in-distribution data – two models may have the same train loss or test accuracy, but very different out-of-distribution accuracies. Such spurious generalization behavior may also explain the variation in performance on split H observed by Gao et al. (2020) and Jiang and Bansal (2021).

One reason often cited for unstable generalization is sharp local minima (Keskar et al., 2017). However, a visualization of the loss landscape of the models at various points during training shows relatively flat planes. The landscapes for training and "adverb to verb" data are simply well aligned for some model-batch combinations, and less so for others (see Figure 6). We also investigated whether the batches used to update the models immediately before out-of-distribution performance spikes had any special properties that would facilitate generalization. We saved batches that preceded an increase on split H accuracy of at least 5%, injected them randomly into the training of other models, and recorded the difference in performance caused. However, we found no statistically significant improvement over random batches, and no statistically significant differences in feature or label distributions of such "spike" batches.

We did find that batch size had an impact on the likelihood of generalization spikes. We trained 10

[2] github.com/marcellodebernardi/loss-landscapes

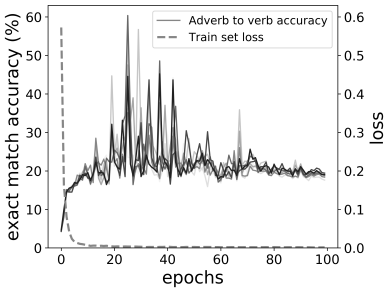


Figure 5: Accuracy on split H over the course of training for a model without action attention

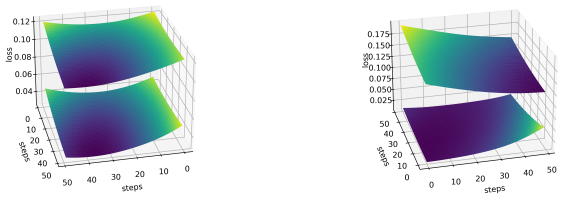


Figure 6: Examples of loss landscapes for models trained without weight decay, visualized with the loss-landscapes library [3]. Lower planes show the landscapes for a random training batch of size 256. Upper planes show the landscapes for the entire "adverb to verb" split. For some model-batch combinations, the two align well (left). For others, less so (right).

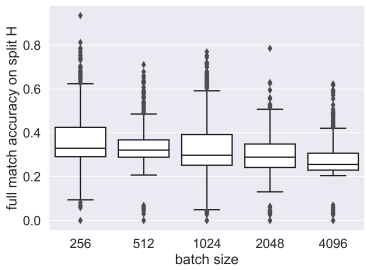


Figure 7: Distributions of split H accuracy sampled during training, for 5 different batch sizes

models without weight decay on 5 different batch sizes using a 2% subset of the training data. All models were trained for the same number of absolute updates. For all batch sizes, the random initialization of the ten models used the same random seeds. We then sampled the models' performance on split H at 50 points in regular intervals during training. As shown in Figure 7, generalization performance with smaller batches was higher but more volatile. Comparing the distribution of sampled "adverb to verb" accuracies across batch sizes yielded statistically significant Z-scores > 2 between batch sizes ≤ 512 and ≥ 2048. This is consistent with previous findings that smaller batch sizes facilitate better generalization (Smith and Le, 2018; Keskar et al., 2017; Smith et al., 2018; Hoffer et al., 2017; Masters and Luschi, 2018). Details on statistical tests are given in E.

6 Discussion

The core of systematic generalization, namely, the ability to flexibly compose known parts, is not something neural networks seem incapable of – as long as they receive atomic units as inputs that are separated from irrelevant context. Otherwise, they may overfit and learn solutions that only perform well on in-distribution data. Seen from this perspective, factors identified as helpful to generalization, both in the literature and in this study, are all mechanisms that can contribute to learning atomic input units. Weight decay facilitates this by serving as a kind of inductive simplicity bias (Power et al., 2021; Kirk et al., 2021). So do soft attention mechanisms, which filter out irrelevant inputs. So does the hard attention bottleneck employed in this paper, by decoupling content, which is only relevant for target identification, from location, which is only relevant for navigation (Heinze-Deml and Bouchacourt, 2020; Dubois et al., 2020).

7 Conclusion

In summary, we build on Tang et al.'s neuroevolution approach to selective attention and embed it in a hybrid model. We apply this model to the task of systematic generalization in grounded instruction following and explore the effect of various design decisions on out-of-distribution performance. We find that weight decay and attention mechanisms facilitate compositional generalization by encouraging sparse representations divorced from irrelevant context, and that selective attention dramatically improves the model's sample efficiency. We also find that, even without weight decay and attention, generalization performance may improve sporadically during training independent of in-distribution accuracy, especially with smaller batch sizes. Studies on out-of-distribution generalization should therefore employ a sufficiently high number of training runs to obtain a reliable estimate of a models' generalization robustness.

Although our architecture is specific to the dataset at hand, the factors contributing to its performance are consistent with related work on systematic generalization and likely to apply to other situations as well. However, compositional generalization encompasses a wide range of skills and even within systematic generalization, solving one task, e.g., recombining shapes and colors, may not translate to another, e.g. recombining directions. Several gSCAN tasks remain unsolved and likely require different inductive biases than the ones presented here. We hope that this closer look at the minimal requirements for generalization on the various gSCAN test splits can inform future work on this benchmark going forward.

References

Ekin Akyürek, Afra Feyza Akyürek, and Jacob Andreas. 2020. Learning to Recombine and Resample Data For Compositional Generalization. In *Proceedings of ICLR*.

Jacob Andreas. 2020. Good-Enough Compositional Data Augmentation. In *ACL*, pages 7556–7566.

Jacob Andreas, Marco Baroni, Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, Antoine Bordes, Jacob Devlin, Alona Fyshe, Leila Wehbe, et al. 2019. Measuring Compositionality in Representation Learning. In *Proceedings of ICLR*, volume 375, pages 2227–2237.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016a. Learning to Compose Neural Networks for Question Answering. In *Proceedings of NAACL-HTL*, pages 1545–1554.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. Neural Module Networks. In *Proceedings of CVPR*, pages 39–48.

Joris Baan, Jana Leible, Mitja Nikolaus, David Rau, Dennis Ulmer, Tim Baumgärtner, Dieuwke Hupkes, and Elia Bruni. 2019. On the Realization of Compositionality in Neural Networks. In *BlackboxNLP@ ACL*, pages 127–137.

Marco Baroni. 2020. Linguistic generalization and compositionality in modern artificial neural networks. *Philosophical Transactions of the Royal Society B*, 375(1791):20190307.

Rahma Chaabouni, Eugene Kharitonov, Diane Bouchacourt, Emmanuel Dupoux, and Marco Baroni. 2020. Compositionality and Generalization in Emergent Languages. In *Proceedings of ACL*, pages 4427–4442.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. 2018. Gated-Attention Architectures for Task-Oriented Language Grounding. In *Proceedings of AAAI*, pages 2819–2826.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *Proceedings of ICML*.

Thierry Deruyttere, Simon Vandenhende, Dusan Grujicic, Luc Van Gool, and Marie-Francine Moens. 2019. Talk2Car: Taking Control of Your Self-Driving Car. In *Proceedings of EMNLP/IJCNLP (1)*, pages 2088–2098.

Yann Dubois, Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. 2020. Location Attention for Extrapolation to Longer Sequences. In *Proceedings of ACL*, pages 403–413.

Tong Gao, Qi Huang, and Raymond Mooney. 2020. Systematic Generalization on gSCAN with Language Conditioned Embedding. In *Proceedings of AACL-IJCNLP*, pages 491–503.

Daniel J. Gauthier, Erik Bollt, Aaron Griffith, and Wendson A. S. Barbosa. 2021. Next Generation Reservoir Computing. *Nature Communications*, 12(1):5564.

Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Posík. 2010. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *GECCO (Companion)*, pages 1689–1696. ACM.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1):1–18.

Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evol. Comput.*, 9(2):159–195.

Christina Heinze-Deml and Diane Bouchacourt. 2020. Think before you act: A simple baseline for compositional generalization. *arXiv preprint arXiv:2009.13962*.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. 2017. Grounded Language Learning in a Simulated 3D World. *arXiv preprint arXiv:1706.06551*.

Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Proceedings of NeurIPS*, pages 1731–1741.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality Decomposed: How do Neural Networks Generalise? *Journal of Artificial Intelligence Research*, 67:757–795.

Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. 2018. Learning compositionally through attentive guidance. *arXiv preprint arXiv:1805.09657*.

Yichen Jiang and Mohit Bansal. 2021. Inducing Transformer's Compositional Generalization Ability via Auxiliary Sequence Prediction Tasks. In *Proceedings of EMNLP*, pages 6253–6265.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *Proceedings of ICLR*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring Compositional Generalization: A Comprehensive Method on Realistic Data. In *Proceedings of ICLR*.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. 2021. A Survey of Generalisation in Deep Reinforcement Learning. *arXiv preprint arXiv:2111.09794*.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. 2021. Compositional Networks Enable Systematic Generalization for Grounded Language Understanding. In *Findings of EMNLP*, pages 216–226.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of ICLR*, pages 2873–2882. PMLR.

Adam Liška, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? Searching for a compositional RNN in a haystack. In *AEGAP Workshop@ ICML*.

João Loula, Marco Baroni, and Brenden M. Lake. 2018. Rearranging the Familiar: Testing Compositional Generalization in Recurrent Networks. In *BlackboxNLP@ EMNLP*.

Dominic Masters and Carlo Luschi. 2018. Revisiting Small Batch Training for Deep Neural Networks. *arXiv preprint arXiv:1804.07612*.

R. Thomas McCoy, Junghyun Min, and Tal Linzen. 2020. BERTs of a feather do not generalize together: Large variability in generalization across models with similar test set performance. In *BlackboxNLP@ EMNLP*, pages 217–227.

Dipendra Kumar Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. 2018. Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction. In *Proceedings of EMNLP*, pages 2667–2678.

Kohei Nakajima. 2020. Physical reservoir computing—An introductory perspective. *Japanese Journal of Applied Physics*, 59(6):060501.

Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M. Lake. 2021. Improving Coherence and Consistency in Neural Sequence Models with Dual-System, Neuro-Symbolic Reasoning. *Proceedings of NeurIPS*, 34.

Santiago Ontanon, Joshua Ainslie, Zachary Fisher, and Vaclav Cvicek. 2022. Making Transformers Solve Compositional Tasks. In *Proceedings of ACL*, pages 3591–3607.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. 2021. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets. In *MATH-AI@ ICLR*.

Linlu Qiu, Hexiang Hu, Bowen Zhang, Peter Shaw, and Fei Sha. 2021. Systematic Generalization on gSCAN: What is Nearly Solved and What is Next? In *Proceedings of EMNLP*, pages 2180–2188.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. 2020. A Benchmark for Systematic Generalization in Grounded Language Understanding. *Proceedings of NeurIPS*, 33:19861–19872.

Gresa Shala, André Biedenkapp, Noor Awad, Steven Adriaensen, Marius Lindauer, and Frank Hutter. 2020. Learning step-size adaptation in CMA-ES. In *International Conference on Parallel Problem Solving from Nature*, pages 691–706. Springer.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *Proceedings of CVPR*, pages 10737–10746. Computer Vision Foundation / IEEE.

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2018. Don't Decay the Learning Rate, Increase the Batch Size. In *Proceedings of ICLR*.

Samuel L. Smith and Quoc V. Le. 2018. A Bayesian Perspective on Generalization and Stochastic Gradient Descent. In *Proceedings of ICLR*.

Sanjay Subramanian, Sameer Singh, and Matt Gardner. 2019. Analyzing Compositionality in Visual Question Answering. *ViGIL@ NeurIPS*, 7.

Yujin Tang, Duong Nguyen, and David Ha. 2020. Neuroevolution of Self-Interpretable Agents. In *Proceedings of GECCO*, pages 414–424.

Konstantinos Varelas, Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Ouassim Ait ElHara, Yann Semet, Rami Kassab, and Frédéric Barbaresco. 2018. A Comparative Study of Large-Scale Variants of CMA-ES. In *PPSN (1)*, volume 11101 of *Lecture Notes in Computer Science*, pages 3–15. Springer.

Noah Weber, Leena Shekhar, and Niranjan Balasubramanian. 2018. The Fine Line between Linguistic Generalization and Failure in Seq2Seq-Attention Models. In *Gen-Deep@ NAACL*, pages 24–27.

Haonan Yu, Xiaochen Lian, Haichao Zhang, and Wei Xu. 2018a. Guided Feature Transformation (GFT): A Neural Language Grounding Module for Embodied Agents. In *CoRL*, volume 87 of *PMLR*, pages 81–98.

Haonan Yu, Haichao Zhang, and Wei Xu. 2018b. Interactive Grounded Language Acquisition and Generalization in a 2D World. In *Proceedings of ICLR*.

A Background on CMA-ES

CMA-ES begins by sampling λ individual solutions $x_1^{(g+1)}, ..., x_\lambda^{(g+1)}$ from a multivariate Gaussian distribution $\mathcal{N}\big(m^{(g)}, {\sigma^{(g)}}^2 C^{(g)}\big)$ with mean $m^{(g)}$, step size $\sigma^{(g)}$ and covariance matrix $C^{(g)}$. The initial mean, step size and covariance matrix are then adapted iteratively to increase the likelihood of successful solutions as evaluated by some function f. Mean adaptation is done by shifting m by the weighted average of the μ best solutions of generation g (Shala et al., 2020):

$$m^{(g+1)} = m^{(g)} + c_m \sum_{i=1}^{\mu} w_i \big(x_{i:\sigma}^{(g+1)} - m^{(g)}\big), \quad (5)$$

where c_m is a learning rate. The new step size σ is determined as follows (Shala et al., 2020):

$$\sigma^{(g+1)} = \sigma^{(g)} exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma^{(g+1)}\|}{E\|\mathcal{N}(0,I)\|} - 1\right)\right), \quad (6)$$

where c_σ is a separate learning rate, d_σ is a damping parameter, and $\mathbf{p}_\sigma^{(g+1)}$ is the next generation's conjugate evolution path computed as (Hansen et al., 2003):

$$\mathbf{p}_\sigma^{(g+1)} = (1 - c_\sigma) \cdot \mathbf{p}_\sigma^{(g)} + \sqrt{c_\sigma \cdot (2 - c_\sigma)} \cdot \frac{\sqrt{\mu}}{\sigma^{(g)}} \big(x_\mu^{(g+1)} - x_\mu^{(g)}\big). \quad (7)$$

Finally, the covariance matrix is updated (Hansen et al., 2003):

$$C^{(g+1)} = (1 - c_{cov}) \cdot C^{(g)} + c_{cov} \cdot \mathbf{p}_c^{(g+1)} \big(\mathbf{p}_c^{(g+1)}\big)^T, \quad (8)$$

where c_{cov} is another learning rate.

B Detailed Evaluation Results

Parameter	**Size**
Hidden layer	28,800
Layer normalization weights	100
Layer normalization biases	100
Output layer	600
Selective attention key matrix	1,024
Self-attention key matrix	4,096
Action attention key matrix	12,800
Total	**47,520**

Table 5: Overview of our model's trainable parameters

	0.01	0.02	0.1	0.5	1.0
A	0.996± 0.002	1.000± 0.000	1.000± 0.000	1.000± 0.000	1.000± 0.000
B	N/A	N/A	N/A	N/A	N/A
C	N/A	N/A	N/A	N/A	N/A
D	0.000± 0.000	0.000± 0.000	0.034± 0.032	0.021± 0.025	0.019 ± 0.017
E	0.997± 0.001	1.000± 0.000	1.000± 0.000	1.000± 0.000	1.000± 0.000
F	0.995± 0.002	1.000± 0.000	1.000± 0.000	1.000± 0.000	1.000± 0.000
G	0.000± 0.000	0.000± 0.000	0.000± 0.000	0.000± 0.000	0.000± 0.000
H	0.610± 0.182	0.790± 0.165	0.999± 0.001	0.988± 0.028	0.993± 0.01

Table 6: Sequence match accuracies on gSCAN compositional splits with perfect selective attention trained on 1%, 2%, 10%, 50%, and 100% of the dataset

	Att. Match	Exact Match	Exact Match if Att. Match
A	0.951± 0.015	0.925± 0.018	0.988± 0.006
B	0.786± 0.128	0.742± 0.129	0.988± 0.012
C	0.965± 0.028	0.959± 0.03	1.000± 0.000
D	0.934± 0.021	0.001± 0.001	0.001± 0.002
E	0.839± 0.109	0.739± 0.082	0.909± 0.066
F	0.878± 0.054	0.737± 0.078	0.886± 0.049
G	0.718± 0.07	0.004± 0.002	0.006± 0.003
H	0.918± 0.033	0.395± 0.145	0.441± 0.171

Table 7: Sequence and attention match accuracies on gSCAN compositional splits with selective attention but without weight decay (trained on the full dataset)

	Att. Match	Exact Match	Exact Match if Att. Match
A	0.947± 0.020	0.922± 0.025	0.996± 0.002
B	0.781± 0.188	0.730± 0.211	1.000± 0.000
C	0.947± 0.066	0.929± 0.076	1.000± 0.000
D	0.931± 0.027	0.000± 0.000	0.000± 0.000
E	0.901± 0.058	0.857± 0.066	0.996± 0.003
F	0.863± 0.073	0.806± 0.093	0.994± 0.005
G	0.772± 0.072	0.000± 0.000	0.000± 0.000
H	0.919 ± 0.032	0.238 ± 0.037	0.272 ± 0.034

Table 8: Sequence and attention match accuracies on gSCAN compositional splits with selective attention but without action attention (trained on the full dataset)

Table 9: Sequence and attention match accuracies with selective attention and auxiliary feedback, trained on 1%, 2%, 10%, 50%, and 100% of the dataset

	0.01			0.02			0.1			0.5			1.0		
	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match
A	0.974 ± 0.005	0.916 ± 0.011	0.969 ± 0.004	0.990 ± 0.003	0.971 ± 0.006	0.995 ± 0.001	0.999 ± 0.000	0.995 ± 0.001	1.000 ± 0.000	0.999 ± 0.000	0.997 ± 0.001	1.000 ± 0.000	0.999 ± 0.000	0.997 ± 0.001	0.999 ± 0.000
B	0.772 ± 0.079	0.700 ± 0.086	0.978 ± 0.007	0.702 ± 0.175	0.649 ± 0.195	0.997 ± 0.004	0.846 ± 0.127	0.816 ± 0.143	1.000 ± 0.000	0.804 ± 0.212	0.771 ± 0.232	1.000 ± 0.000	0.781 ± 0.212	0.735 ± 0.254	0.999 ± 0.000
C	0.948 ± 0.030	0.900 ± 0.036	0.974 ± 0.011	0.985 ± 0.007	0.970 ± 0.009	0.997 ± 0.001	0.998 ± 0.001	0.995 ± 0.002	1.000 ± 0.000	0.996 ± 0.005	0.991 ± 0.008	1.000 ± 0.000	0.998 ± 0.003	0.994 ± 0.004	0.999 ± 0.000
D	0.977 ± 0.004	0.000 ± 0.000	0.000 ± 0.000	0.991 ± 0.003	0.000 ± 0.000	0.000 ± 0.000	1.000 ± 0.000	0.035 ± 0.027	0.035 ± 0.027	1.000 ± 0.000	0.026 ± 0.020	0.026 ± 0.020	1.000 ± 0.000	0.022 ± 0.015	0.022 ± 0.015
E	0.892 ± 0.062	0.811 ± 0.071	0.965 ± 0.006	0.941 ± 0.047	0.904 ± 0.054	0.997 ± 0.002	0.985 ± 0.014	0.968 ± 0.019	1.000 ± 0.001	0.985 ± 0.017	0.971 ± 0.031	1.000 ± 0.000	0.985 ± 0.016	0.974 ± 0.020	1.000 ± 0.000
F	0.930 ± 0.032	0.834 ± 0.042	0.946 ± 0.015	0.958 ± 0.026	0.918 ± 0.030	0.990 ± 0.007	0.992 ± 0.011	0.983 ± 0.017	1.000 ± 0.000	0.992 ± 0.013	0.984 ± 0.020	1.000 ± 0.000	0.997 ± 0.004	0.991 ± 0.006	1.000 ± 0.000
G	0.780 ± 0.050	0.000 ± 0.000	0.000 ± 0.000	0.805 ± 0.064	0.000 ± 0.000	0.000 ± 0.000	0.852 ± 0.050	0.000 ± 0.001	0.000 ± 0.001	0.804 ± 0.052	0.000 ± 0.000	0.000 ± 0.000	0.766 ± 0.101	0.000 ± 0.000	0.000 ± 0.000
H	0.957 ± 0.017	0.377 ± 0.082	0.406 ± 0.119	0.987 ± 0.005	0.468 ± 0.119	0.472 ± 0.130	0.994 ± 0.003	0.942 ± 0.037	0.956 ± 0.035	0.995 ± 0.005	0.960 ± 0.060	0.972 ± 0.062	0.998 ± 0.002	0.984 ± 0.011	0.992 ± 0.007

Table 10: Sequence and attention match accuracies with selective attention but without auxiliary feedback, trained on 1%, 2%, 10%, 50%, and 100% of the dataset, using relative target distances

	0.01			0.02			0.1			0.5			1.0		
	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match
A	0.765 ± 0.154	0.604 ± 0.189	0.815 ± 0.17	0.946 ± 0.034	0.906 ± 0.044	0.984 ± 0.011	0.975 ± 0.008	0.976 ± 0.003	0.999 ± 0.001	0.978 ± 0.01	0.986 ± 0.003	1.000 ± 0.000	0.928 ± 0.022	0.83 ± 0.034	0.919 ± 0.017
B	0.502 ± 0.134	0.372 ± 0.154	0.791 ± 0.272	0.693 ± 0.195	0.618 ± 0.217	0.978 ± 0.013	0.73 ± 0.108	0.697 ± 0.118	0.999 ± 0.001	0.64 ± 0.198	0.603 ± 0.223	1.000 ± 0.000	0.7 ± 0.167	0.595 ± 0.157	0.928 ± 0.051
C	0.649 ± 0.205	0.507 ± 0.218	0.768 ± 0.264	0.939 ± 0.022	0.906 ± 0.033	0.985 ± 0.012	0.973 ± 0.012	0.967 ± 0.015	0.999 ± 0.003	0.975 ± 0.012	0.971 ± 0.018	0.999 ± 0.001	0.911 ± 0.084	0.897 ± 0.093	1.000 ± 0.000
D	0.749 ± 0.169	0.000 ± 0.000	0.000 ± 0.000	0.946 ± 0.039	0.000 ± 0.000	0.000 ± 0.000	0.983 ± 0.006	0.002 ± 0.002	0.002 ± 0.002	0.984 ± 0. 008	0.004 ± 0.006	0.005 ± 0.006	0.913 ± 0.026	0.008 ± 0.009	0.009 ± 0.010
E	0.615 ± 0.147	0.461 ± 0.144	0.736 ± 0.258	0.802 ± 0.074	0.731 ± 0.083	0.981 ± 0.01	0.938 ± 0.032	0.918 ± 0.028	0.994 ± 0.01	0.961 ± 0.024	0.954 ± 0.029	0.999 ± 0.003	0.841 ± 0.076	0.741 ± 0.062	0.91 ± 0.028
F	0.666 ± 0.175	0.502 ± 0.191	0.772 ± 0.267	0.911 ± 0.048	0.826 ± 0.058	0.942 ± 0.031	0.96 ± 0.022	0.944 ± 0.024	0.998 ± 0.003	0.971 ± 0.008	0.967 ± 0.016	0.999 ± 0.002	0.844 ± 0.08	0.675 ± 0.093	0.874 ± 0.038
G	0.624 ± 0.11	0.000 ± 0.000	0.000 ± 0.000	0.764 ± 0.083	0.000 ± 0.000	0.000 ± 0.000	0.800 ± 0.069	0.000 ± 0.000	0.000 ± 0.000	0.772 ± 0.045	0.000 ± 0.000	0.000 ± 0.000	0.73 ± 0.058	0.000 ± 0.000	0.000 ± 0.000
H	0.76 ± 0.15	0.311 ± 0.139	0.302 ± 0.17	0.922 ± 0.036	0.576 ± 0.056	0.615 ± 0.072	0.96 ± 0.023	0.744 ± 0.086	0.822 ± 0.058	0.962 ± 0.013	0.771 ± 0.097	0.822 ± 0.095	0.899 ± 0.039	0.564 ± 0.062	0.667 ± 0.055

Table 11: Sequence and attention match accuracies with selective attention but without auxiliary feedback, trained on 1%, 2%, 10%, 50%, and 100% of the dataset, using absolute target locations

	0.01			0.02			0.1			0.5			1.0		
	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match
A	0.476 ± 0.005	0.228 ± 0.003	0.145 ± 0.226	0.541 ± 0.136	0.333 ± 0.182	0.203 ± 0.338	0.58 ± 0.201	0.41 ± 0.28	0.299 ± 0.457	0.68 ± 0.246	0.555 ± 0.347	0.399 ± 0.489	0.742 ± 0.214	0.593 ± 0.291	0.612 ± 0.415
B	0.435 ± 0.009	0.251 ± 0.006	0.025 ± 0.074	0.499 ± 0.145	0.353 ± 0.168	0.169 ± 0.342	0.493 ± 0.115	0.375 ± 0.155	0.245 ± 0.4	0.562 ± 0.193	0.469 ± 0.235	0.399 ± 0.489	0.616 ± 0.17	0.508 ± 0.211	0.591 ± 0.449
C	0.438 ± 0.018	0.266 ± 0.008	0.028 ± 0.085	0.481 ± 0.11	0.364 ± 0.128	0.161 ± 0.331	0.534 ± 0.203	0.454 ± 0.239	0.254 ± 0.406	0.647 ± 0.265	0.59 ± 0.31	0.400 ± 0.49	0.738 ± 0.248	0.700 ± 0.295	0.700 ± 0.458
D	0.313 ± 0.002	0.000 ± 0.000	0.000 ± 0.000	0.401 ± 0.192	0.000 ± 0.000	0.000 ± 0.000	0.446 ± 0.267	0.000 ± 0.000	0.000 ± 0.000	0.582 ± 0.330	0.013 ± 0.033	0.014 ± 0.036	0.666 ± 0.289	0.001 ± 0.002	0.001 ± 0.002
E	0.412 ± 0.003	0.249 ± 0.004	0.000 ± 0.000	0.464 ± 0.132	0.338 ± 0.155	0.163 ± 0.334	0.524 ± 0.22	0.407 ± 0.261	0.197 ± 0.395	0.623 ± 0.256	0.529 ± 0.317	0.399 ± 0.488	0.621 ± 0.179	0.503 ± 0.204	0.552 ± 0.451
F	0.462 ± 0.003	0.227 ± 0.006	0.032 ± 0.097	0.515 ± 0.117	0.332 ± 0.148	0.167 ± 0.339	0.554 ± 0.188	0.417 ± 0.245	0.199 ± 0.399	0.662 ± 0.247	0.544 ± 0.335	0.398 ± 0.488	0.704 ± 0.207	0.526 ± 0.25	0.53 ± 0.434
G	0.448 ± 0.01	0.000 ± 0.000	0.000 ± 0.000	0.493 ± 0.093	0.000 ± 0.000	0.000 ± 0.000	0.535 ± 0.167	0.000 ± 0.000	0.000 ± 0.000	0.596 ± 0.183	0.000 ± 0.000	0.000 ± 0.000	0.63 ± 0.157	0.000 ± 0.000	0.000 ± 0.000
H	0.557 ± 0.021	0.122 ± 0.012	0.000 ± 0.000	0.61 ± 0.109	0.248 ± 0.111	0.103 ± 0.206	0.648 ± 0.16	0.365 ± 0.224	0.173 ± 0.346	0.715 ± 0.183	0.37 ± 0.195	0.262 ± 0.334	0.744 ± 0.149	0.375 ± 0.202	0.394 ± 0.328

Table 12: Sequence and attention match accuracies without selective attention, trained on 1%, 2%, 10%, 50%, and 100% of the dataset

	0.01			0.02			0.1			0.5			1.0		
	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match	Att. Match	Exact Match	Exact Match if Att. Match
A	0.498 ± 0.136	0.311 ± 0.12	0.282 ± 0.162	0.570 ± 0.222	0.512 ± 0.126	0.407 ± 0.271	0.578 ± 0.243	0.736 ± 0.028	0.617 ± 0.311	0.545 ± 0.216	0.869 ± 0.033	0.739 ± 0.372	0.548 ± 0.204	0.896 ± 0.033	0.652 ± 0.428
B	0.44 ± 0.1	0.273 ± 0.083	0.195 ± 0.167	0.566 ± 0.21	0.43 ± 0.127	0.352 ± 0.235	0.614 ± 0.229	0.535 ± 0.139	0.476 ± 0.24	0.579 ± 0.204	0.663 ± 0.199	0.572 ± 0.324	0.571 ± 0.195	0.695 ± 0.218	0.587 ± 0.343
C	0.492 ± 0.201	0.312 ± 0.105	0.256 ± 0.178	0.604 ± 0.274	0.484 ± 0.143	0.368 ± 0.249	0.631 ± 0.31	0.598 ± 0.175	0.457 ± 0.305	0.59 ± 0.278	0.753 ± 0.168	0.661 ± 0.335	0.557 ± 0.251	0.786 ± 0.171	0.696 ± 0.356
D	0.435 ± 0.174	0.001 ± 0.001	0.000 ± 0.000	0.548 ± 0.273	0.002 ± 0.004	0.002 ± 0.004	0.544 ± 0.284	0.002 ± 0.002	0.002± 0.004	0.504 ± 0.262	0.002 ± 0.005	0.002 ± 0.006	0.512 ± 0.245	0.003 ± 0.006	0.004 ± 0.012
E	0.334 ± 0.1	0.28 ± 0.077	0.199 ± 0.217	0.417 ± 0.133	0.405 ± 0.085	0.212 ± 0.26	0.422 ± 0.14	0.549 ± 0.019	0.255 ± 0.312	0.552 ± 0.255	0.691 ± 0.02	0.286 ± 0.351	0.561 ± 0.25	0.721 ± 0.023	0.296 ± 0.363
F	0.474 ± 0.101	0.311 ± 0.133	0.228 ± 0.234	0.522 ± 0.132	0.499 ± 0.126	0.447 ± 0.3	0.578 ± 0.203	0.672 ± 0.116	0.652 ± 0.334	0.575 ± 0.199	0.786 ± 0.099	0.534 ± 0.439	0.576 ± 0.195	0.816 ± 0.099	0.546 ± 0.448
G	0.487 ± 0.14	0.000 ± 0.000	0.000 ± 0.000	0.561 ± 0.21	0.000 ± 0.000	0.000 ± 0.000	0.571 ± 0.236	0.000 ± 0.000	0.000 ± 0.000	0.558 ± 0.214	0.000 ± 0.000	0.000 ± 0.000	0.562 ± 0.206	0.000 ± 0.000	0.000 ± 0.000
H	0.491 ± 0.117	0.127 ± 0.073	0.158 ± 0.167	0.535 ± 0.189	0.314 ± 0.139	0.337 ± 0.23	0.547 ± 0.209	0.537 ± 0.067	0.503 ± 0.331	0.528 ± 0.177	0.579 ± 0.163	0.636 ± 0.421	0.528 ± 0.164	0.655 ± 0.131	0.628 ± 0.427

	Att. Match	Exact Match	Exact Match if Att. Match
Pull while spinning, Push while zigzagging, Walk hesitantly	0.982± 0.008	0.989± 0.005	0.996± 0.002
H:Adverb to verb	0.996± 0.003	0.93± 0.059	0.943± 0.055

Table 13: Sequence and attention match accuracies on additional held-out verb-adverb combinations and split H with selective attention and auxiliary feedback (trained on the full dataset)

	Att. Match	Exact Match	Exact Match if Att. Match
Red squares, Yellow squares, Green cylinders, Blue circles	0.991± 0.013	0.987± 0.015	1.000± 0.000
B:Yellow squares	0.855± 0.144	0.829± 0.165	1.000± 0.000
C:Red squares	0.996± 0.006	0.992± 0.007	1.000± 0.000

Table 14: Sequence and attention match accuracies on additional held-out shape-color target combinations and splits B and C with selective attention and auxiliary feedback (trained on the full dataset)

C Neuron pruning

For each neuron in the final hidden layer of the model, we recorded its activation, multiplied by its outgoing weight (no biases were used in the model, except in the layer normalization layer). We then sorted neurons based on their accumulated contribution to the final model output and tested exact sequence accuracy on the gSCAN dev set with the top X% of neurons active. The rest were disabled by setting outgoing weights to 0. Detailed results are shown in Table 15.

% of top hidden neurons active	unablated model	w/o action attention	w/o selective attention	w/o weight decay
10%	0.538 ± 0.054	0.354 ± 0.096	0.576 ± 0.054	0.042 ± 0.023
11%	0.664 ± 0.111	0.442 ± 0.117	0.627 ± 0.057	0.044 ± 0.026
12%	0.855 ± 0.108	0.522 ± 0.158	0.671 ± 0.051	0.068 ± 0.027
13%	0.998 ± 0.001	0.649 ± 0.164	0.715 ± 0.045	0.073 ± 0.021
14%	-	0.824 ± 0.104	0.782 ± 0.034	0.079 ± 0.025
15%	-	0.876 ± 0.090	0.823 ± 0.033	0.083 ± 0.033
16%	-	0.904 ± 0.029	0.867 ± 0.034	0.093 ± 0.032
17%	-	-	0.902 ± 0.024	0.087 ± 0.031
18%	-	-	0.916 ± 0.025	0.097 ± 0.053
20%	-	-	-	0.126 ± 0.092
30%	-	-	-	0.119 ± 0.069
40%	-	-	-	0.263 ± 0.149
50%	-	-	-	0.486 ± 0.231
60%	-	-	-	0.741 ± 0.171
70%	-	-	-	0.810 ± 0.114
80%	-	-	-	0.874 ± 0.045
90%	-	-	-	0.880 ± 0.048
95%	-	-	-	0.885 ± 0.049
100%	-	-	-	0.906 ± 0.025

Table 15: Exact match accuracy on in-distribution data for ablated and unablated models with different percentages of disabled top contributing hidden neurons

D Error analyses

D.1 Confusion matrices

We collected the feature vectors for the grid cells that were most attended to by the models trained with selective attention, but without auxiliary feedback. We also collected the feature vectors of the actual target objects. We then created confusion matrices for the parts of the feature vector relating to the agent, to color, to size, and to shape (shown in Figures 8 - 13). For color and size, we distinguish between situations where the attribute is mentioned in the command and those where it is not.

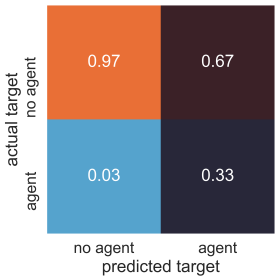


Figure 8: Confusion matrix for the agent dimension

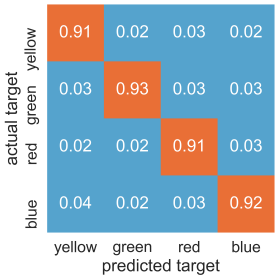


Figure 9: Confusion matrix for the color dimensions when color is specified in the command

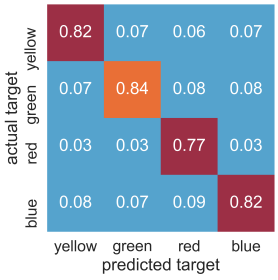


Figure 10: Confusion matrix for the color dimensions when color is not specified in the command

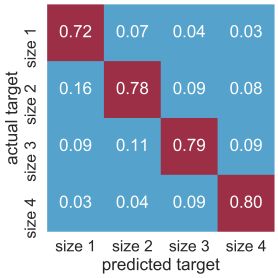


Figure 11: Confusion matrix for the color dimensions when size is specified in the command

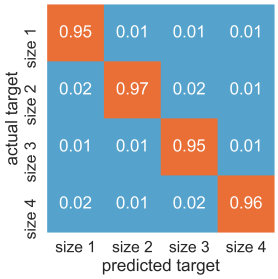


Figure 12: Confusion matrix for the color dimensions when size is not specified in the command

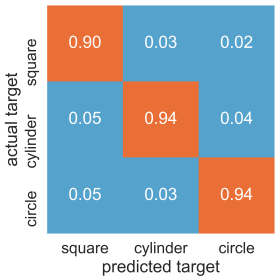
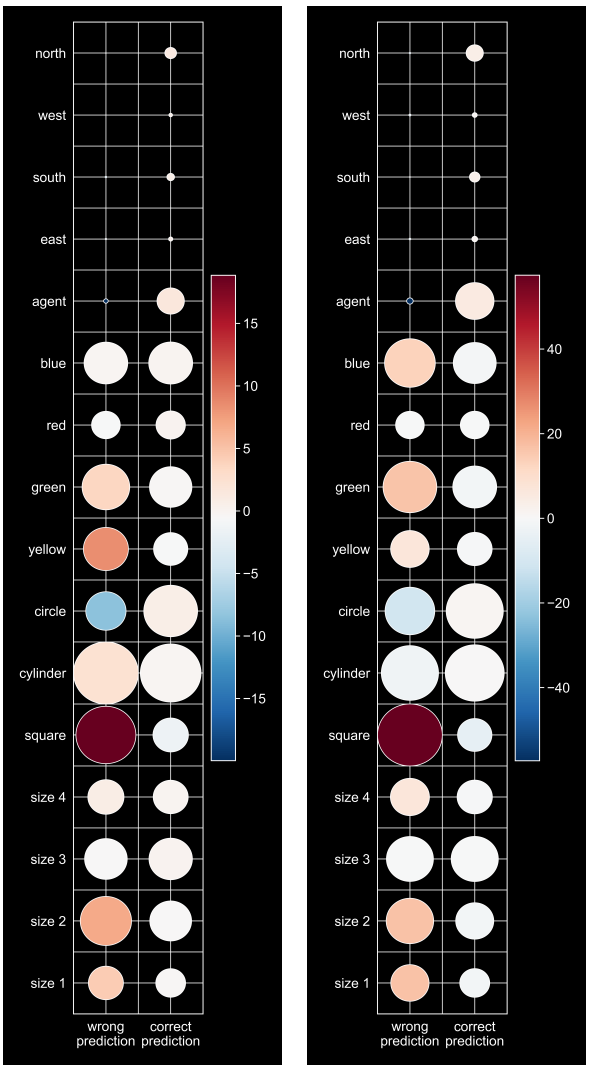


Figure 13: Confusion matrix for the shape dimensions (always specified in the command)

D.2 Ablated selective attention

We use a chi-squared test to compare the kind of target features that models tend to mis-identify when they are trained with vs. without selective attention. Figure 14 shows the test's standardized residuals for the model trained without selective attention, i.e., the strength of the difference between observed and expected values. Squares, the color yellow, and small object sizes are especially over-represented in the model's incorrect target predictions.



(a) dev set (b) test set

Figure 14: Plots of the standardized residuals of a Chi-square test comparing the wrong predictions of models trained with vs. without selective attention, on in-distribution data. We ran this test both on the dev set (14a) and the test set (14b) with similar results. Circle color represents absolute value of the residuals. Red indicates that a feature is over-represented, blue indicates a feature is under-represented. Circle size represents the number of occurrences in the tested set.

E “Spontaneous" generalization

E.1 “Spike" batches

To test if the batches used to update the model before a spike in performance on split H had any special properties, we trained a model with batch size 256 without action attention for 50 epochs and saved any batches that preceded at least a 5% increase in exact match accuracy on a 2% subset of split H. We then trained 10 additional models (with the same random seeds as used in the batch size experiments) and injected one of the “good" batches during training with a chance of 10%. We recorded the difference to the performance on the split H dev set before the batch update. A comparison of the distributions of split H performance differences after an update with “good" batch vs. a normal batch yields a Z-statistic of 0.665, which is not significant at the 0.05 level.

Injecting “good" batches also does not seem to increase the overall likelihood of higher performance on split H during training. We compared the distributions of split H accuracies sampled after each epoch for the models trained with and without “good" batch injections in the course of training. A two-sample Kolmogorov-Smirnov test yielded a p-value of 0.413, which is well above the threshold of 0.05 and indicates there is no difference between the distributions. Finally, we compare the distribution of labels in the “good" batches vs. the normal batches with a chi-squared test that yields a p-value of 0.445 – again, indicating little to no difference between the distributions.

E.2 Effect of batch size

We trained 10 models without weight decay on a 2% subset of the training data with batch sizes 256, 512, 1024, 2048, and 4096. The number of epochs was adjusted for each batch size so that all models were trained for the same number of absolute updates. For all batch sizes, the random initialization of the ten models used the same random seeds. We then sampled the models' performance on split H at 50 points in regular intervals during training and compared Z-scores for the resulting distributions. Results are given in Table 16

Batch size 1	Batch size 2	Z-score
256	512	1.35
256	1024	1.03
256	2048	**3**
256	4096	**4.09**
512	256	-1.35
512	1024	-0.09
512	2048	**2.33**
512	4096	**3.95**
1024	256	-1.03
1024	512	0.09
1024	1024	1.68
1024	4096	**2.74**
2048	256	**-3**
2048	512	**-2.33**
2048	1024	-1.68
2048	4096	1.77
4096	256	**-4.09**
4096	512	**-3.95**
4096	1024	**-2.74**
4096	2048	-1.77

Table 16: Pairwise comparison of distributions of split H performance sampled during training, for 5 different batch sizes. Statistically significant scores ($\geq |2|$) marked in bold.