Compositional Generalization in Grounded Language Learning via Induced Model Sparsity

Sam Spilsbury and **Alexander Ilin**
Department of Computer Science
Aalto University
Espoo, Finland
{first.last}@aalto.fi

Abstract

We provide a study of how induced model sparsity can help achieve compositional generalization and better sample efficiency in grounded language learning problems. We consider simple language-conditioned navigation problems in a grid world environment with disentangled observations. We show that standard neural architectures do not always yield compositional generalization. To address this, we design an agent that contains a goal identification module that encourages sparse correlations between words in the instruction and attributes of objects, composing them together to find the goal.[1] The output of the goal identification module is the input to a value iteration network planner. Our agent maintains a high level of performance on goals containing novel combinations of properties even when learning from a handful of demonstrations. We examine the internal representations of our agent and find the correct correspondences between words in its dictionary and attributes in the environment.

1 Introduction

Ideally, when training an agent that acts upon natural language instructions, we want the agent to understand the meaning of the words, rather than overfitting to the training instructions. We expect that when an agent encounters an unfamiliar instruction made up of familiar terms, it should be able to complete the task. In this sense, the agent learns to leverage both *groundedness* of language; for example in English, tokens in the language map to observed attributes of objects or phenomena in its environment, as well as its *compositionality*; which enables the description of potentially infinite numbers of new phenomena from known components (Chomsky, 1965). Using language to express goals is potentially a way to approach task distribution shift and sample efficiency, key problems in reinforcement learning (Sodhani et al., 2021; Jang et al., 2021).

However, compositional generalization does not come automatically with standard architectures when using language combined with multi-modal inputs, as indicated by the mixed results of generalization performance in Goyal et al. (2021); Sodhani et al. (2021). Concurrently with Qiu et al. (2021), we show that the Transformer architecture can demonstrate generalization, but requires large amounts of data for training. In this work, we tackle sample inefficiency and retain generalization.

Our contributions are as follows. We propose a model and a training method that utilizes the inductive biases of *sparse interactions* and *factor compositionality* when finding relationships between words and disentangled attributes. We hypothesize that such sparsity in the *interactions* between object attributes and words (as opposed to just their representations) leads to a correct identification of what attributes the words actually correspond to, instead of what they are merely correlated with. We show in both quantitative and qualitative experiments that such sparsity and factor compositionality enable compositional generalization. To improve sample efficiency, we decouple the goal identification task (which requires language understanding) from the planning process (implemented with an extension of Value Iteration Networks).

2 Related Work

Compositional Generalization and Language Grounding There is a long line of work on learning to achieve language encoded instructions within interactive environments. Vision-Language Navigation environments typically require an agent to navigate to a requested goal object (for example, DeepMind Lab (Beattie et al., 2016), R2R (Anderson et al., 2018) and ALFRED (Shridhar et al., 2020)). Algorithmic and deep imitation learning approaches for autonomous agents in these environ-

[1] github.com/aalto-ai/sparse-compgen

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop, pages 143 - 155
July 10-15, 2022 ©2022 Association for Computational Linguistics

ments have been proposed, but room for improvement in both generalization performance and sample efficiency remains (Chen and Mooney, 2011; Bisk et al., 2016; Shridhar et al., 2021).

The generalization issue arises because there are many possible instructions or goals that could be expressed with language and a learner may not necessarily observe each one within its training distribution. Some are “out of distribution" and maintaining performance on them is not guaranteed; a problem well known the within reinforcement learning community (Kirk et al., 2021). However, a peculiar feature of language instructions is that language is *compositional* in nature. This has led to an interest in whether this aspect can be leveraged to get better generalization on unseen goals made up of familiar terms (Oh et al., 2017; Hermann et al., 2017). However, even in simple environments such as BabyAI (Chevalier-Boisvert et al., 2019), and gSCAN (Ruis et al., 2020) this can still be difficult problem.

Various approaches to leveraging compositionality have been proposed, including gated word-channel attention (Chaplot et al., 2018), hierarchical processing guided by parse-trees (Kuo et al., 2021), graph neural networks (Gao et al., 2020), neural module networks (Andreas et al., 2016), and extending agents with a boolean task algebra solver (Tasse et al., 2022). Closest to our approach are Heinze-Deml and Bouchacourt (2020); Hanjie et al. (2021) which use attention to identify goal states, Narasimhan et al. (2018); Ruis and Lake (2022), which decompose goal identification and planning modules, Bahdanau et al. (2019) which uses a discriminator to model reward for instructions and Buch et al. (2021) which factorizes object classification over components. We contribute a new approach of learning sparse attention over factored observations, then attaching that attention module to a learned planning module. This can be shown to solve the compositional generalization problem by learning the correct correspondences between words and factors without spurious correlation.

Representation Sparsity We hypothesize that sparsity is an important factor in the design of a compositional system because it can bias the optimization procedure towards solutions where relationships exist only between things that are actually related and not just weakly correlated. Previous work has shown that induced sparsity can improve both generalization (Zhao et al., 2021) and model interpretability (Wong et al., 2021). Induced sparsity has been applied both within the model weights (Jayakumar et al., 2020) and also within the attention computation (Zhang et al., 2019). In our work, we apply it in the space of all possible interactions between words in the language and attributes of objects in the environment.

Sample Efficiency In grounded language learning, improved sample efficiency may enable new use-cases, for example, the training of intelligent assistants by users who would not have the patience to give many demonstrations of a desired behavior (Tucker et al., 2020). Various tricks have been proposed to improve sample efficiency in reinforcement learning in general (Yu, 2018), including prioritized replay (Hessel et al., 2018), data augmentation (Laskin et al., 2020) and model based learning or replay buffers (van Hasselt et al., 2019; Kaiser et al., 2020). Limited work exists on explicitly addressing sample efficiency in the grounded language learning context (Chevalier-Boisvert et al., 2019; Hui et al., 2020; Qiu et al., 2021). In this work, sample efficiency is one of our primary objectives and we claim to achieve it using a functionally decomposed architecture and offline learning.

3 Experimental Setup

We study the performance of our proposed approach on the `GoToLocal` task of the BabyAI environment. A detailed description of the environment is given in Appendix A. The environment can be seen as a Goal-Conditioned Markov Decision Process, (formally defined in Kaelbling (1993)). Each episode is generated by a seed i and has an initial state $s_0^{(i)}$. To obtain a reward during an episode, the agent must successfully complete the language-encoded instruction (denoted g) that it is given. The language is simple and generated by the use of a templating system. `GoToLocal` consists only of statements “go to (a|the) (color) (object)". Each state is a fully observable 8-by-8 grid world and each cell (denoted c_{ij}) may contain an object, the agent, or nothing.

The information in each cell is *disentangled*; the object’s color is in a separate channel to the object’s type. We work with disentangled observations because they have been shown to improve the performance and sample-efficiency of attention-based models (see, e.g., Loynd et al., 2020). This disentanglement is preserved by embedding each component separately as factored embeddings q_a.

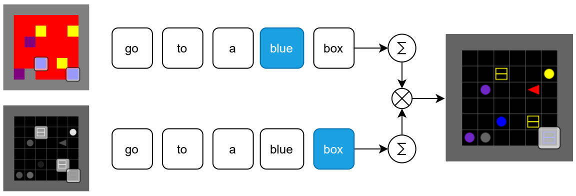


Figure 1: Attention over separate components of the input representation. The model is a single layer of query-key attention applied to each component individually, where queries are image attribute values for a given component, the keys are the words and values are a one-tensor. Performing an AND relation on the components means taking the product of each attention operation.

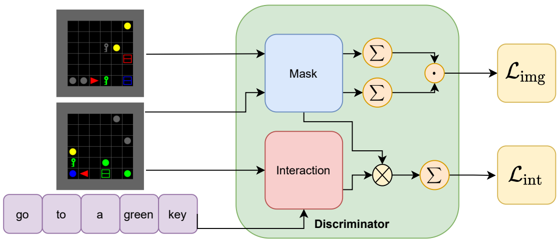


Figure 2: Discriminator training method. $\mathcal{L}_{\text{img}}$ is used to train the "mask" module. Because true examples are those where the agent is situated next to the same goal, an optimal mask module should select states the agent is facing. This can help with learning $S(s, g)$.

The environment also comes with an expert agent which can produce an optimal trajectory for a given initial state and goal $\tau^{(i)}|s_0, g$.

The key performance metric is *success rate*. A *success* happens if the agent completes the instruction within 64 steps. We study compositional generalization and sample efficiency.

By *compositional generalization* we mean maintaining performance when navigating to objects with attribute combinations not seen during training. To study this, we separate goals into $\mathcal{G}_{\text{ID}}$ and $\mathcal{G}_{\text{OOD}}$ following the principle of leaving one attribute combination out (shown in Table 1 and similar to the "visual" split in Ruis et al. (2020)). Then we create corresponding training and validation datasets, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{v\_ID}}$ and $\mathcal{D}_{\text{v\_OOD}}$ each containing the same number of trajectories (10,000) per goal. Trajectories for each goal are generated in the same way, so we expect that a different split of $\mathcal{G}_{\text{ID}}$ and $\mathcal{G}_{\text{OOD}}$ following the same principle will cause similar behavior in both the baselines and our models. Finer details about the dataset construction are given in Appendix B.

	blue	red	green	yellow	purple	grey
box	🟦	🟦	🟥	🟦	🟦	🟥
ball	🟦	🟥	🟦	🟦	🟥	🟦
key	🟥	🟦	🟦	🟥	🟦	🟦

Table 1: Split between $\mathcal{G}_{\text{ID}}$ and $\mathcal{G}_{\text{OOD}}$. Blue cells are object attributes appearing in the goals for $\mathcal{G}_{\text{ID}}$ and red cells correspond to those in $\mathcal{G}_{\text{OOD}}$.

By *sample efficiency* we mean achieving a high level of performance given a smaller number of samples than conventional methods might require. We denote N as the number of trajectories per goal that an agent has access to and study performance at different levels of N. We train various models using $\mathcal{D}_{\text{train}}$ and describe the training methodology and results in Section 5.1.

4 Designing a learning method

We now design a learning agent with Section 2 in mind. To complete an instruction, the agent needs to identify the goal and plan actions to reach it. The learning problem is decomposed into separate modules with separate training processes. Subsections 4.1 and 4.2 describe a sparse vision-language architecture and training process for identifying goal cells ($S(s, g) \in \mathbb{R}^{H \times W}$). Subsection 4.3 shows how to plan given that identification $\pi(a_t|S(s, g))$.

4.1 Sparse Factored Attention for Goal Identification

We hypothesize that learning to to match objects to descriptions by matching their factors to words individually is a process that generalizes more strongly than matching all at once. For example, the agent should match "red ball" to `red ball` because "red" matches factor `red` and "ball" matches `ball`. If the agent only learns that "red ball" means `red ball`, as a whole, then it may not learn what the meaning of the parts are. Standard architectures, which can mix information between all the words or factors of the observation might fall into the trap of doing the latter over the former. We propose two inductive biases to learn the former. The first bias is *factor compositionality*. As language is a descriptive tool, words should operate at the level of object properties and not entire objects. The second bias is *sparsity* in word/attribute relationships. A particular word should only match as many attributes as necessary.

From this intuition, we propose a "Sparse Factored Attention" architecture, pictured in Fig. 1. The words are the keys and attributes are the queries. However, a critical difference is that the attribute embeddings for each c_{jk} remain partitioned into separate components q_a corresponding to each factor. The normalized dot product $(\hat{c}_{jkq_a} \cdot \hat{g}_w)$ is computed separately between the instruction and

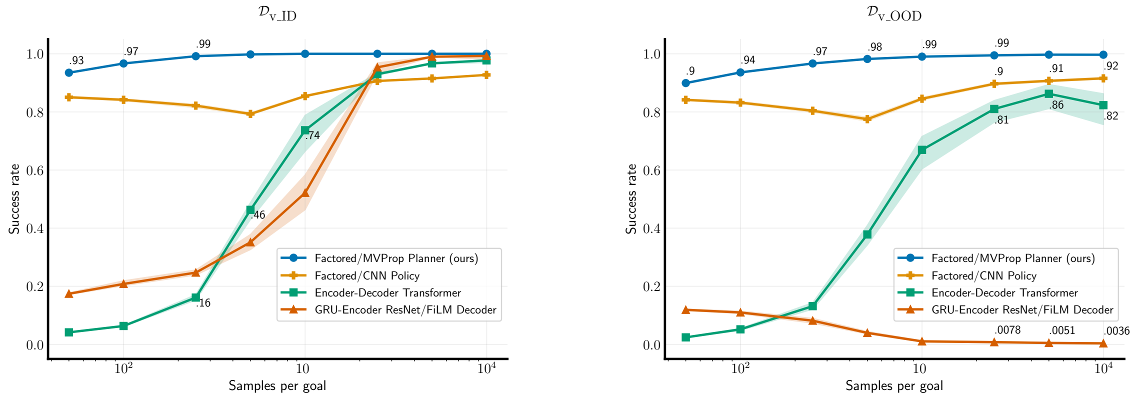


Figure 3: Success Rates on validation seeds. The x-axis is the log-scale number of samples per goal statement. Since there are 18 different goals in the training set, the total number of samples is $18 \times N$. Peak performance on within-distribution goals for prior methods in the same environment is typically reached at 2500 samples per goal, or 45,000 total samples. However, in the compositional generalization case ($\mathcal{D}_{\text{v\_OOD}}$), both baselines fail to maintain the same level of performance, although the Transformer baseline can provide a good amount of performance at a high number of samples. In comparison, Factored/MVProp (ours) reaches a comparable level of performance to peak performance of the baselines at 50 samples per goal, or 900 total samples, and maintains a consistent level of performance on the out-of-distribution validation set. Without a differentiable planner, Factored/CNN is still efficient but does not perform quite as well as Factored/MVProp.

the flattened observation cells for each factor, then the elementwise product is taken over each q_a:

$$S(s_t, g)_{jk} = \prod_{q_a} \sigma(\alpha(\sum_{w} \hat{c}_{jkq_a} \cdot \hat{g}_w) + \beta) \quad (1)$$

where α and β are a single weight and bias applied to all dot product scores and σ is the sigmoid activation function. In practice, exp-sum-log is used in place of $\prod_{q_a}$ for training stability. To encourage sparsity within the outer product, we add an L1 regularization penalty to the outer product of the normalized embedding spaces ($\lambda||\hat{E_c} \cdot \hat{E_w}^T||_1$) to the loss. This goes beyond just penalizing $S(s, g)$; it ensures that the system's entire knowledge base is sparse, which in turn assumes that no relationship exists between unseen pairs and is also not sensitive to imbalances in the dataset regarding how often different objects appear in the observations.

4.2 Training with a Discriminator

We found that performance of end-to-end learning by differentiating through the planner to our model was highly initialization sensitive. Instead we propose to learn goal-identification and planning separately. However, $\mathcal{D}$ does not have labels of which cells are goal cells, but only full observations of the environment at each step. To learn to identify the goals, we propose a self-supervised objective in the form of a state-goal discriminator architecture $\hat{D}(s, g)$ shown in Fig. 2, which is trained to match end-states to their corresponding goals.

The discriminator is defined as:

$$\hat{D}(s, g) = \sum_{HW} M(s) \cdot S(s, g) \qquad (2)$$

where $S(s, g)$ is the trainable goal identification module and $M(s) \rightarrow \mathbb{R}^{H \times W}$, $\sum_{\text{HW}} M(s) = 1$ is a "Mask Module". The "Mask Module" is a convolutional neural network with no downsampling or pooling and returns a single-channel "spatial softmax" with the same spatial dimensions as s. Ideally the mask module should learn to identify the cell that the agent is facing. When $M(s)$ and $S(s, g)$ are correctly learned, then $\hat{D}(s, g)$ answers whether the agent is at the goal state. The training process for the discriminator uses a loss function similar to a triplet loss between positive, negative, and anchor samples. Positive and negative goals are sampled from the set of goals, then corresponding positive, anchor, and negative end-states. Finer details of this process are given in Appendix D.

4.3 Planning Module

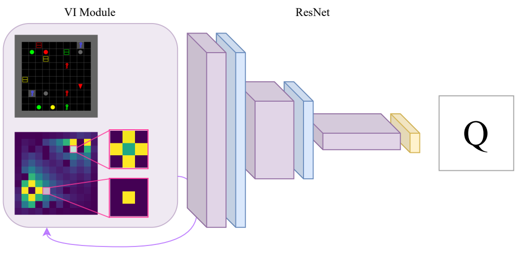
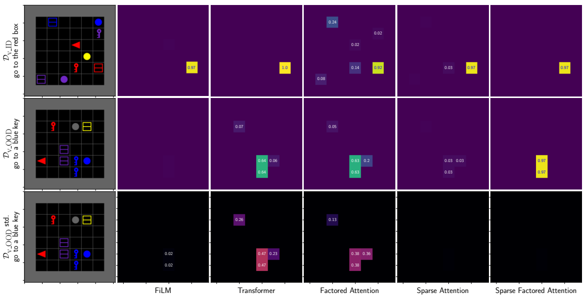
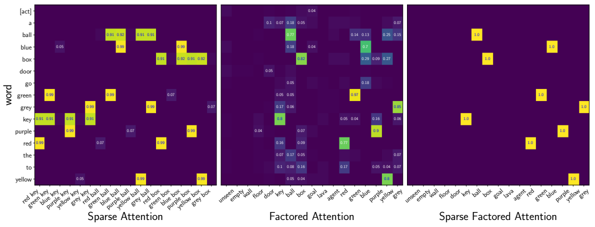


Figure 5: Using a Value Propagation Network (Nardelli et al., 2019) (VPN) to estimate the Q function. VPN is an extension of the Value Iteration Network (Tamar et al., 2017) which makes the convolutional filter propagating value from one cell to its neighbors conditional on its inputs. The Q function is estimated by concatenating the output of the VPN with the estimated rewards, visual features, and agent state, then processing it with a ResNet.

Once $S(s, g)$ is learned, with a knowledge of the connectivity between cells, full observability of the environment, and the assumption that each action



(4a) Qualitative evaluation of interaction networks on environment samples. The **top** row contains the mean activations a $\mathcal{D}_{\text{v\_ID}}$ sample, the **middle** and **bottom** rows are means and standard deviations on a $\mathcal{D}_{\text{v\_OOD}}$ sample. Other models either suffer from overfitting or high variance when predicting OOD goals.



(4b) IQM of Embedding Internal Correlations for our method, showing the effect of applying L1 regularization to the embedding outer product. The horizontal axes correspond to factors and the vertical axes correspond to words. **Left:** when concatenating factor embeddings and applying sparse attention, unseen combinations such as `key/blue key` and `blue/blue key` are given little weight. **Middle:** without sparsity regularization, unrelated factors such as `box/yellow` are confused and less weight is given to the true correspondences. **Right:** ours, where the correspondences between words and factors are learned exactly and others are zero.

moves the agent to a either the same cell or an adjacent, learning to plan to reach a goal state becomes trivial. We extend Value Propagation Networks (Nardelli et al., 2019) for this purpose. Details of our implementation are given in Appendix E.

5 Experimental results

5.1 End-to-End performance on the benchmark task

We first examine performance and sample efficiency on both $\mathcal{D}_{\text{v\_ID}}$ and $\mathcal{D}_{\text{v\_OOD}}$ using the experimental setup described in Section 3. We train our approach and several baseline models for the same number (70,000) of training steps over many values of N and 10 random intializations. The models are briefly described as follows:

Factored/MVProp (blue circles, ours) Sparse Factored Attention is pre-trained with the Discriminator in Section 4.2 and frozen, then we only learn the planning and value networks in Section 4.3.

Factored/CNN (light orange plus marks) Ablation of our model with a skipped planning step; detected goals and observations are processed directly into a policy using a convolutional network.

Transformer (green squares) Standard encoder-decoder transformer, encoder inputs are position-encoded instruction word embeddings, decoder inputs are position-encoded flattened cells and a [CLS] token used to predict the policy.

GRU-Encoder ResNet/FiLM Decoder (red triangles) Process visual observation into policy with interleaved FiLM conditioning on the GRU-encoded instruction, similar to Hui et al. (2020).

The training objective is behavioral cloning of the expert policy. The model is evaluated is every 500 steps. Evaluation is performed in a running copy of the environment seeded using each of the stored seeds in the validation sets. To succeed the agent must solve the task - it is not enough to copy what the expert does on most steps. Further details are given in Appendices C and H.

In contrast to both baselines, our method in Fig. 3 attains a high level of performance on both $\mathcal{D}_{\text{v\_ID}}$ and $\mathcal{D}_{\text{v\_OOD}}$, even with a small number of samples, significantly outperforming both baselines even when those models have a greater number of samples available to learn from.

5.2 Examination of Interaction Module Architectures

We also examine what it is about our model architecture that explains its performance on the benchmark task. We perform an ablation study to examine the effectiveness of different architectures for $S(s, g)$. Performance is measured using a "soft F_1 score" against a ground truth on goal locations, as this is essentially an imbalanced classification problem. The metric is described in more detail in Appendix G

	$\mathcal{D}_{\text{v\_ID}}$	$\mathcal{D}_{\text{v\_OOD}}$
FiLM (Perez et al., 2018)	0.983 ± 0.000	0.015 ± 0.004
Transformer (Vaswani et al., 2017)	1.000 ± 0.000	0.799 ± 0.028
Sparse Attention	0.974 ± 0.000	0.069 ± 0.001
Factored Attention	0.891 ± 0.015	0.739 ± 0.028
Sparse Factored Attention	0.951 ± 0.000	0.951 ± 0.000

Table 2: Inter-quartile mean (IQM) of soft F1 scores (predicted goal location versus ground truth goal location) across seeds, dataset sizes, and checkpoints, with added 95% confidence intervals. Sparse Factored Attention scores consistently well on both datasets.

Each architecture for $S(s, g)$ was trained using $\mathcal{D}_{\text{train}}$ for 200,000 iterations with the parameters in Appendix F. The IQM and 95% confidence interval across seeds and top-10 checkpoints are reported in Table 2 using the package and method provided by (Agarwal et al., 2021). While not perfect, our Sparse Factored Attention model achieves high F_1 scores both $\mathcal{D}_{\text{v\_ID}}$ and $\mathcal{D}_{\text{v\_OOD}}$.

We also visualize mean model predictions and their variance across initializations on sample datapoints from both $\mathcal{D}_{\text{v\_ID}}$ and $\mathcal{D}_{\text{v\_OOD}}$ in Fig. 4a. The average is over instances with F_1 scores in the upper 75% range for their class. FiLM and Sparse Attention fail to identify the test-set goal, and the Transformer and Factored Attention models exhibit high variance on $\mathcal{D}_{\text{v\_OOD}}$ between initializations. Only our Sparse Factored Attention model reliably identifies the goal on both datasets.

5.3 Qualitative Evaluation of Model Weights

Since the Factored Attention model is very simple and its only parameters are the embeddings and single weight and bias, we can also visualize “what the model has learned" by taking the mean normalized outer product of both attribute E_c and word E_w embeddings for models shown in Fig. 4b. A perfect learner should learn a sparse correspondence between each attribute and its corresponding word; it should not confound attributes of different types. The heatmaps show the importance of sparsity regularization on the outer product of the embeddings. Without sparsity regularization, the mean correlation between a word and its correct attribute is weaker and not consistent across all initializations. There are also other “unwanted" confounding correlations, for example, between “box" and `blue`, which also appear more strongly in some initialization and data limit combinations as indicated by its high standard deviation. In contrast, the Sparse Factored Attention model displays an almost perfect correlation between each word and the corresponding attribute and very little variance between checkpoints (not pictured). In this sense, we can be much more confident that the Sparse Factored Attention model has *actually learned the symbol grounding* and the meaning of the words as they relate to cell attributes in the environment.

6 Conclusion

We studied the problem of compositional generalization and sample efficient grounded language learning for a vision-language navigation agent. We showed that even under strong assumptions on environment conditions such as full observability and disentanglement of inputs, compositional generalization and sample efficiency do not arise automatically with standard learning approaches. We demonstrate how such conditions can be leveraged by our Sparse Factored Attention model presented in Section 4.1. We demonstrate a method to learn goal identification without labels in Section 4.2 and planning Section 4.3 using a small number of offline trajectories. We further showed superior sample efficiency and generalization performance in Section 5.1 and perform a model analysis and ablation study in Section 5.2 to show how our proposed approach works the way we intended.

7 Limitations of this Work

Goal identification and planning The goal identification and planning methods proposed in Section 4.3 do not work over compound goals. The discriminator training method in Section 4.2 requires that $\mathcal{D}_{\text{train}}$ can be partitioned into subsets corresponding to each goal and that there is at most a many-to-one relationship between goal cell configurations and language statements.

Measuring sample efficiency Testing sample efficiency of gradient-based methods learned from off-policy datasets is not a well specified problem, since each training step could be used to improve the model performance by a small amount an arbitrary number of times. It was a qualitative judgment of the researchers of when to stop training, and we used the same upper bound on training steps for all models to ensure a fair comparison.

Further limitations of this work are discussed in Appendix I.

8 Responsible Research Statement

We also provide details regarding code and reproducibility in Appendix J and computational resource usage in Appendix K. We do not anticipate any special ethical issues to arise from this work as it is foundational in nature and uses a synthetically generated dataset. However, the methods presented in this work may be more amenable to analytic languages as opposed to synthetic ones.

9 Acknowledgements

We thank Yonatan Bisk for his valuable feedback and suggestions on this work. We also acknowledge the computational resources provided by the Aalto Science-IT project and the support within the Academy of Finland Flagship programme: Finnish Center for Artificial Intelligence (FCAI).

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. 2021. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, pages 29304–29320.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Seyed Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. 2019. Learning to understand goal specifications by modelling reward. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. 2016. Deepmind lab. *arXiv:1612.03801*.

Richard Bellman. 1957. A Markovian Decision Process. In *Journal of Mathematics and Mechanics*, volume 6, pages 679–684.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016. Natural language communication with robots. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 751–761. The Association for Computational Linguistics.

Shyamal Buch, Li Fei-Fei, and Noah D. Goodman. 2021. Neural event semantics for grounded language understanding. *Transactions of the Association for Computational Linguistics*, 9:875–890.

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 the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2819–2826. AAAI Press.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press.

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 *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. Minimalistic Gridworld Environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`.

N. Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Tong Gao, Qi Huang, and Raymond J. Mooney. 2020. Systematic generalization on gSCAN with language conditioned embedding. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2020, Suzhou, China, December 4-7, 2020*, pages 491–503. Association for Computational Linguistics.

Prasoon Goyal, Raymond J. Mooney, and Scott Niekum. 2021. Zero-shot task adaptation using natural language. *arXiv:2106.02972*.

Austin W. Hanjie, Victor Zhong, and Karthik Narasimhan. 2021. Grounding language to entities and dynamics for generalization in reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139, pages 4051–4062.

Christina Heinze-Deml and Diane Bouchacourt. 2020. Think before you act: A simple baseline for compositional generalization. *arXiv:2009.13962*, 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:1706.06551*.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the*

Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 3215–3222. AAAI Press.

David Yu-Tung Hui, Maxime Chevalier-Boisvert, Dzmitry Bahdanau, and Yoshua Bengio. 2020. BabyAI 1.1. *arXiv:2007.12770*.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. 2021. BC-Z: zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning, 8-11 November 2021, London, UK*, pages 991–1002.

Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. 2020. Top-KAST: Top-K always sparse training. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual Event*.

Leslie Pack Kaelbling. 1993. Learning to achieve goals. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 1094–1099.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. 2020. Model based reinforcement learning for atari. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. 2021. A survey of generalisation in deep reinforcement learning. *arXiv:2111.09794*.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual Event*.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. 2021. Compositional networks enable systematic generalization for grounded language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 216–226. Association for Computational Linguistics.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. 2020. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual Event*.

Ricky Loynd, Roland Fernandez, Asli Celikyilmaz, Adith Swaminathan, and Matthew J. Hausknecht. 2020. Working memory graphs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 6404–6414.

Karthik Narasimhan, Regina Barzilay, and Tommi S. Jaakkola. 2018. Grounding language for transfer in deep reinforcement learning. *J. Artif. Intell. Res.*, 63:849–874.

Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip H. S. Torr, and Nicolas Usunier. 2019. Value propagation networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Junhyuk Oh, Satinder P. Singh, Honglak Lee, and Pushmeet Kohli. 2017. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2661–2670.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. 2018. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3942–3951. AAAI Press.

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 the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 2180–2188. Association for Computational Linguistics.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. 2020. A benchmark for systematic generalization in grounded language understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual Event*.

Laura Ruis and Brenden M. Lake. 2022. Improving systematic generalization through modularity and augmentation. *arXiv:2202.10745*.

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 *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10737–10746. Computer Vision Foundation / IEEE.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. 2021. Multi-task reinforcement learning with context-based representations. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 9767–9779.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. 2017. Value iteration networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4949–4953.

Geraud Nangue Tasse, Steven James, and Benjamin Rosman. 2022. Generalisation in lifelong reinforcement learning through logical composition. In *10th International Conference on Learning Representations, ICLR 2022, Virtual Event*.

Aaron D. Tucker, Markus Anderljung, and Allan Dafoe. 2020. Social and governance implications of improved data efficiency. In *AIES '20: AAAI/ACM Conference on AI, Ethics, and Society, New York, NY, USA, February 7-8, 2020*, pages 378–384. ACM.

Hado van Hasselt, Matteo Hessel, and John Aslanides. 2019. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14322–14333.

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 *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Eric Wong, Shibani Santurkar, and Aleksander Madry. 2021. Leveraging sparse linear layers for debuggable deep networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 11205–11216.

Yang Yu. 2018. Towards sample efficient reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5739–5743.

Jiajun Zhang, Yang Zhao, Haoran Li, and Chengqing Zong. 2019. Attention with sparsity regularization for neural machine translation and summarization. *IEEE ACM Trans. Audio Speech Lang. Process.*, 27(3):507–518.

Mingde Zhao, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup, and Yoshua Bengio. 2021. A consciousness-inspired planning agent for model-based reinforcement learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event*.

A Details of the BabyAI Environment

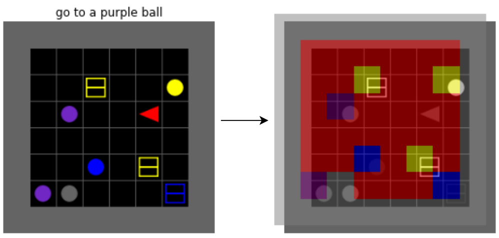


Figure 6: An illustration of the integer-encoded inputs provided by the BabyAI environment. Color and shape information are encoded in separate channels and are independent from each other.

BabyAI is a simple grid world-like environment based on Minigrid (Chevalier-Boisvert et al., 2018). chose to use this environment for this project due to its simplicity, ease of generating expert trajectories, and input representation characteristics. In the environment, the agent is given instructions to complete in a sythetically generated language that is a subset of English. The seed for the environment, i, determines its initial state s_0 and goal g, which comes from the set $\mathcal{G}$ for a given level. Within the environment, there are a few different object types (`ball`, `box`, `key`) each of which may be one of six different colors (`red`, `blue`, `green`, `grey`, `purple`, `yellow`). The agent can face one of four different directions. There are seven actions available to the agent: `turn left`, `turn right`, `go forward`, `open`, `pick up`, `put down` and `signal done`. The original implementation provides partial observations, however we modify the environment to make the state space fully observable due to the inherent difficulty planning over unobservable states.[2] The observations are subdivided into cells as explained in Section 3. Each cell is a disentangled vector of integers of comprised of three components, the first corresponding to the object type, the second corresponding to the color and the third corresponding to the object that the agent is holding.

The goals g come in the form of simple language statements such as “go to a red box". BabyAI comes in several “levels". Each level requires the agent to demonstrate competency at a certain subset of “skills", summarized in Table 1 of the original by Chevalier-Boisvert et al. (2019).

In this work, we focus on the `GoToLocal` task, where the agent must learn to reach the goal object indicated in the language-encoded instruction by navigating to the correct location in an 8×8 grid world and then performing the `signal done` action within a fixed number of steps. Performing `signal done` facing the wrong cell terminates the episode with a reward of zero. Requiring the `signal done` action precludes the trivial solution of ignoring g and visiting every object until successful. Other objects may exist in the grid as *distractors*; non-goal objects that the agent must learn to ignore and navigate around depending on the goal.

B Collecting Trajectories for the Dataset

In the `GoToLocal` task there are 36 possible goal statements. Each statement begins with “go to", followed by “the" or “a", then color and object terms. To collect the seeds to generate each environment and their corresponding solutions $\tau_i | s_0, g$, we iterate consecutively through random seeds starting from zero and reset the environment using each seed. The environment is “solved" using the provided `BotAgent`, which implements an optimal policy. We do not want our measurements or training to be biased by imbalances in the dataset, so we want to ensure that each goal has the same number of samples in $\mathcal{D}$. 10,000 state-action trajectories with a length of at least 7 are stored for each goal g. A trajectory τ is a tuple $(x, (s_0, ..., s_t), (a_0, ..., a_t), (r_0, ..., r_t), g)$, consisting of (respectively), the seed, state trajectory, action trajectory, rewards and goal.

We split the data into training, “in-distribution" and “combinatorial generalization" (out of distribution) validation sets. To make these splits, we first split the goals into “in-distribution" goals $\mathcal{G}_{\text{ID}}$ and “combinatorial generalization" goals $\mathcal{G}_{\text{OOD}}$. One color and object combination is omitted from $\mathcal{G}_{\text{ID}}$ for each color and placed in $\mathcal{G}_{\text{OOD}}$, specifically, goals containing `red ball`, `green box`, `blue key`, `purple ball`, `grey box` and `yellow key`. The “in-distribution" validation set $\mathcal{D}_{\text{v\_ID}}$ consists of the last 20 trajectories in $\mathcal{D}$ corresponding to each $g \in \mathcal{G}_{\text{ID}}$. The “combinatorial generalization" set $\mathcal{D}_{\text{v\_OOD}}$ is defined similarly with the last 40 trajectories in $\mathcal{G}_{\text{OOD}}$.[3] The training

[2] We also reproduce the relevant experiments in (Chevalier-Boisvert et al., 2019) using this fully-observable state space for fair comparison in Section 5.1 of this work.

[3] The reason for using the last 40 trajectories is to ensure that both validation datasets have the same number trajectories in total; since there are twice as many goals covered in $\mathcal{D}_{\text{v\_ID}}$

set $\mathcal{D}$ consists of all trajectories corresponding to $g \in \mathcal{G}_{\text{ID}}$, excluding those in $\mathcal{D}_{\text{v\_ID}}$.

C Details of the Baselines

The first baseline is similar to the architecture used in (Hui et al., 2020); featuring a GRU to encode g, a ResNet to encode s and the use of FiLM layers (Perez et al., 2018) to modulate feature maps according to the encoded g, which in turn is flattened and concatenated with an embedding corresponding to the agent's current direction to produce a hidden representation z. The policy π is estimated using an MLP from z. The only difference to (Hui et al., 2020) is that the memory module used to handle partial observability and exploration is removed, since the environment is fully observable.

The second baseline is an encoder-decoder Transformer model (Vaswani et al., 2017), where the input sequence is the individual words in g added with their 1D positional encodings, and the output sequence is the 2D encoded observation s added with their 2D positional encodings. A classification token is appended to the end of the output sequence, which uses a linear prediction head to estimate π in the same way as above. 10000 steps of learning rate warmup followed by subsequent logarithmic decay in the learning rate are used when training the Transformer.

For all models, an embedding dimension of 32 is used for both the words in g and each attribute in c_{jk}, implying that the total embedding dimension is 96 after each embedded attribute is concatenated together. The batch size and learning rate for Adam used during training are 32 and 10^{-4} respectively.

D Training the Discriminator

Two goals, g_+, g_- are sampled without replacement uniformly from the set of all known goals $\mathcal{G}_{\text{v\_ID}}$. Two trajectories are sampled without replacement from $\{\mathcal{D}_{\text{train}} | g = g_+\}$, τ_1^{g+}, τ_2^{g+} and one trajectory is sampled from $\{\mathcal{D}_{\text{train}} | g = g_-\}$, τ^{g-}. s_r is assumed to be the rewarding states for all three trajectories and are denoted $(s_r^{g+})_1, (s_r^{g+})_2, (s_r^{g-})_1$. With probability $\frac{1}{|\mathcal{G}|}$, $(s_r^{g-})_1$ is replaced with a random state in $\tau_{0:T-1}^{g-}$, so that the discriminator also sees states that are not rewarding for any goal. The discriminator's inputs and labels are tuples (s_1, s_2, g, y). In this tuple, s_1 is an "anchor" state, s_2 is a comparison state, g is the goal and y is the label. The tuple $((s_r^{g+})_1, (s_r^{g+})_2, g_+, 1)$ is a "true" example and the tuple $((s_r^{g+})_1, (s_r^{g-})_1, g_+, 0)$ is a "false" example. True and false examples are sampled consecutively.

We define the loss for the discriminator as:

$$\mathcal{L}_D(s_1, s_2, g, y) = \mathcal{L}_{\text{int}}(s_2, g, y) + \mathcal{L}_{\text{img}}(s_1, s_2, y) \quad (3)$$

The "interaction loss" $\mathcal{L}_{\text{int}}$ is used to optimize $S(s, g)$. As S classifies whether a given s is a rewarding state for g, the loss is a binary-cross-entropy loss, where the outputs of S are logits:

$$\mathcal{L}_{\text{int}}(s_2, g, y) = y \log D(s_2, g) + (1 - y) \log(1 - D(s_2, g)) \quad (4)$$

The image-matching loss $\mathcal{L}_{\text{img}}$ is used to resolve the ambiguity of whether a high loss value in $\mathcal{L}_{\text{int}}$ was caused by an incorrect parameterization of $M(s)$ or $S(s, g)$. Define the *mask-weighted image* as $I(s) = \sum_{\text{HW}} M(s) \odot s$ and the *normalized mask-weighted image* as $\hat{I}(s) = \frac{I(s)}{||I(s)||_2^2}$ Then the normalized image-matching loss L_{img} is given by:[4]

$$\mathcal{L}_{\text{img}}(s_1, s_2, y) = ||(\hat{I}(s_1) \cdot \hat{I}(s_2)) - y||_2^2 \quad (5)$$

E Planning with Value Iteration

Value-based differentiable planning networks assume the existence of a function $r(s, g)$: $\mathbb{R}^{H \times W \times A}$ which returns the cell-action combinations in s that give a reward for being reached by an agent. In this case, r is modelling a reward function for goal g in terms of c_{jk}. Knowing both this function and the dynamics $p(s_{t+1} | s, a_t)$ with a discrete state space enables using *Value Iteration* (Bellman, 1957) to solve for the *optimal value function* V^*, which induces an *optimal policy*:

$$\pi^* = \max_a Q(s, a) = \max_a \textstyle\sum_{a \in | \mathcal{A}} r(s, a) + \gamma p(s_{t+1} | s, a_t) V(s_{t+1}) \quad (6)$$

In this case, we do not know the dynamics exactly, but we have a prior that we can start from, which is that all neighboring cells to a given cell are uniformly reachable from the current cell by any action $p(c_{j+l, k+m} | a_t, c_{jk}), l, m \in [-1, 1], a \in \mathcal{A}$. In this problem, the agent's occupancy of a cell c_{jk} corresponds to a state s given the initialization s_0, so a mapping exists from values of cells to values

[4] We use mean-squared error as opposed to binary cross entropy loss for the the image-matching loss as we found that in practice it was less sensitive to label noise, which was present in this problem, since goals such as "go to a red key" and "go to the red key" involve the same object color combination but are nevertheless treated as separate goals by the discriminator.

of states up to the agent's rotation given an initialization $V(c_{jk}) \rightarrow V(s|s_0)$.

To refine our estimate of the the dynamics $p(s_{t+1}|s, a_t)$ and improve our estimate of $Q(s, a_t, g)$, we can use the above assumptions and a differentiable planning method known as a *Value Iteration Network* (VIN) (Tamar et al., 2017). Starting with $V_0(c_{jk}) = r(c_{jk}, g)$, VIN re-expresses value-iteration as a form of *convolution* performed recursively K times:

$$V_{k+1}(c_{jk}, g) = \max \begin{cases} V_k(c_{jk}, g), \\ \max\limits_{a \in \mathcal{A}} \sum\limits_{l,m \in \mathcal{N}(c_{jk})} \mathbf{P}_{a,l-j,m-k} V_k(c_{lm}, g) \end{cases} \tag{7}$$

where $\mathcal{N}(c_{jk})$ are the neighbors of a cell and $\mathbf{P}$ is a learnable linear estimate of the dynamics (the transition probabilities to neighboring cells for each action). In reality, the dynamics are dependent on what the neighboring cells actually contain. *Max Value Propagation Networks* (MVProp) (Nardelli et al., 2019) extend on VIN by replacing $\mathbf{P}$ with a scalar *propagation weight* conditioned on the current cell $\phi(c_{jk})$, where ϕ is any learnable function with non-negative output. In that sense, we learn to model how value *propagates* around the cells. Using the dataset $\mathcal{D}$ we can generate traces of returns from trajectories using an optimal planner with discount factor γ. Then learning $Q(s, a_t, g)$ is done by minimizing the empirical risk with respect to some loss function $\mathcal{L}$:

$$\arg\min_{Q_\theta} \mathbb{E}_{s,a_t \sim \mathcal{D}_{\text{tr}}} \mathcal{L}(Q(s, a_t, g), R(s, a_t))) \tag{8}$$

In the MVProp framework, it is the responsibility of the consumer of $V_K(s, g)$ to map neighboring values of a cell to Q values for actions. Both Tamar et al. (2017) and Nardelli et al. (2019) resolve this problem by including the cell that the agent is currently occupying as part of the state. However, this information is not available to us in $\mathcal{D}$ as we have only the state s and action observation a_t. In practice, this problem turns out not to be insurmountable and good performance can be achieved by simply concatenating as additional channels $V_0(s, g)$ and $V_k(s, g)$ to the initial encoding of s and using a Convolutional Neural Network to encode the image into a single vector of which represents the vector-valued output $Q(s, g) \rightarrow \mathbb{R}^{|\mathcal{A}|}$, eg the action-value function for all actions.

Finally, there is the question of which loss function to use to learn $Q(s, a_t, g)$. We observed that simply using mean-squared error loss between $R(s, a_t)$ and $Q(s, a_t, g)$ led to over-optimistic estimates of Q-values for non-chosen actions. To fix this problem, we added an additional term penalizing any non-zero value for those actions: similar to Conservative Q Learning (Kumar et al., 2020):

$$\begin{split} \mathcal{L}_{\text{VIN}}(s, a_t, g) = ||R(s, a_t, g) - Q(s, a_t, g)||_2^2 + \\ \lambda ||Q(s, a_-, g), a_- \in \{\mathcal{A} \setminus a_t\}||_2^2 \end{split} \tag{9}$$

F Training Parameters of $S(s, g)$

$S(s, g)$ is trained for 200,000 steps, using a learning rate of 10^{-5}, a batch size of 1024 and 16-bit mixed precision used for the model weights and embeddings. During training, models were evaluated both $\mathcal{D}_{\text{v\_ID}}$ and $\mathcal{D}_{\text{v\_OOD}}$ every 20 training steps. The top-10 performing model checkpoints by F_1 score on $\mathcal{D}_{\text{v\_ID}}$ were stored, along with their F_1 score on $\mathcal{D}_{\text{v\_OOD}}$.

G Soft F1 Score

The problem in Section 4.2 is unbalanced; there are a small number of goal states and a large number of non-goal states. Therefore, we propose to use a metric that is robust to the class imbalance, but also takes into account the weight of the predictions as this will be used as the reward model in the planner. The metric is a "soft F1 score" is defined as the harmonic mean of soft-precision and soft-recall, for a single trajectory i (with indexes omitted for brevity):

$$\begin{aligned} P &= \frac{\sum_{\text{HW}}^{jk} y_{jk} S(s, g)_{jk}}{\sum_{\text{HW}}^{jk} (y_{jk} S(s, g)_{jk} + (1 - y_{jk}) S(s, g)_{jk})} \\ R &= \sum_{\text{HW}}^{jk} y_{jk} S(s, g)_{jk} / \sum_{\text{HW}}^{jk} (y_{jk}) \\ F_1 &= 2PR/(P + R) \end{aligned} \tag{10}$$

A high value of soft-F_1 indicates that both precision *and* recall are high.

H End-to-end usage our proposed model

The model is trained in two phases; first, the Sparse Factored Attention model in Section 4.1 is trained using the discriminator task in Section 4.2 for 200,000 steps with a learning rate of $10e^{-5}$ and batch size of 1024. Then, the weights at the end of training (for the corresponding initialization seed and $\mathcal{D}_N$ are frozen and used as the initialization for the VIN model described in Section 4.3. The training parameters and setup used otherwise is the same as is described in Appendix C.

I Additional Limitations

Controlled Environment We used the `GoToLocal` task on BabyAI as the sole reference environment for this study. A fully observable state space, knowledge of the state-space connectivity, and disentangled factors on cell states are very strong assumptions that are leveraged to achieve the results that we present.

Computational resources Sample efficiency does not imply computational efficiency. In particular, we found that training the discriminator in Section 4.2 requires large batch sizes and a large number of samples generated from $\mathcal{D}_N$ to converge.

J Reproducibility of this work

We kept the importance of reproducible research in mind when designing our experimental method. We provide the source code for our approach and seeds used to generate each environment and trajectory in $\mathcal{D}$.

We are unable to provide pre-trained models or log files due to space constraints.

K Computational Resource usage of this work

The person responsible for developing the method took about one year to do so and used a workstation with a single NVIDIA RTX2060 GPU with 6GB of GPU memory to test different approaches. Because the methods that we present in this paper may be sensitive to different weight initializations, we believed it was necessary to show trained model performance using different initialization random initializations, using the methods in (Agarwal et al., 2021) for a more reliable presentation of results. To conduct the experiments using the final version of our methods, we used our SLURM compute cluster with an array of shared NVIDIA Tesla V100 GPUs. We ran 6 different versions of the discriminator experiment, over five different models, ten dataset sizes, ten random initializations, each one taking up to 8 hours to complete, making for 24,000 hours of GPU time used. We ran 3 different versions of the end-to-end experiments over 4 different models, with the same number of dataset sizes and random initializations each one taking up to 12 hours, making for an additional 19,200 hours.