Language Representations for Generalization in Reinforcement Learning

Nikolaj Goodger NIKOLAJGOODGER@STUDENTS.FEDERATION.EDU.AU
Peter Vamplew P.VAMPLEW@FEDERATION.EDU.AU
Cameron Foale C.FOALE@FEDERATION.EDU.AU
Federation University Australia — Mt Helen Campus PO Box 663 Ballarat VIC 3353

Richard Dazeley RICHARD.DAZELEY@DEAKIN.EDU.AU
Deakin University — Locked Bag 20000, Geelong, VIC 3220

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

The choice of state and action representation in Reinforcement Learning (RL) has a significant effect on agent performance for the training task. But its relationship with generalization to new tasks is under-explored. One approach to improving generalization investigated here is the use of language as a representation. We compare vector-states and discrete-actions to language representations. We find the agents using language representations generalize better and could solve tasks with more entities, new entities, and more complexity than seen in the training task. We attribute this to the compositionality of language.

Keywords: reinforcement learning; generalization; representation selection

1. Introduction

Choosing and learning representations for Markov Decision Processes (MDP)s has long been a field of active research (Konidaris, 2019; van Seijen et al., 2014; Abel et al., 2018; Andre and Russell, 2002). However, the choice of representation is often considered from the perspective of solving the training problem or improving sample efficiency rather than how well the solution generalizes. Generalization is important because agents that can generalize better are more robust to changes in their environment and are more likely to be able to solve tasks they have not explicitly been trained to do.

The motivation for this work is to investigate whether we can build more general agents using language by comparing generalization for agents using non-language representations to those using language representations. The choice of language as a representation for generalization is encouraged by the language features of compositionality and productivity (Fasold and Connor-Linton, 2014) in that we can produce new meaning by exchanging sub-expressions and adding new words. This allows us to represent any state or action for which we have the vocabulary using language.

To compare language to non-language representations, we created a new *tidying* environment which allows for both vector and language-based representations of state, as well as discrete and language actions. This environment can be configured differently to test generalization to unseen entities and increased problem size, allowing us to quantitatively compare generalization between representations.

The main contributions of this work are:

- A new environment that can be used to test generalization and compare state and action representations.
- A quantitative comparison of zero-shot task generalization between representations and empirical evidence that the use of language representations allows trained agents to generalize better than agents using vector-state and discrete-actions.
- A quantitative comparison of generalization between model architectures using language representations showing that sequence to sequence models with extractive capabilities generalize better than strictly generative models and that a hybrid model can improve generalization further.

2. Related Work

The use of language as a representation was surveyed in Luketina et al. (2019). In most applications, except for dialogue agents, the language used is synthetic as opposed to natural language (Luketina et al., 2019). A synthetic language will have a different distribution to natural language potentially resulting in large differences between training and test data (Marzoev et al., 2020). The use of Language in RL can be separated into two groups, Language-Conditional RL and Language-Assisted RL (Luketina et al., 2019).

Language-conditional RL includes the application of RL to tasks in which language is a fundamental requirement, such as Dialogue agents, Instruction following, or solving text games. These problems use language in the action-space, state-space, or both. Text-based games have been explored using direct state to action encoding (Branavan et al., 2010; Depristo et al., 2001) and later with deep-learning (Narasimhan et al., 2015; Yuan et al., 2018). The high difficulty of solving text-based games and lack of appropriate environments led to the development of the TextWorld Environment (Côté et al., 2019). This environment allows agents to selectively focus on solving subsets of the challenges inherent to text games. Similarly, BabyAI (Chevalier-Boisvert et al., 2019) was developed to evaluate sample efficiency when an algorithm is provided synthetic language to guide its actions.

Language-Assisted RL applies to problems where language is not required to solve the problem but is used in a supportive manner to improve performance criteria like sample efficiency or generalization. This can include injecting domain knowledge by augmenting state with additional information (Zhong et al., 2019; Narasimhan et al., 2018; Sodhani et al., 2021), structuring the policy (Jiang et al., 2019; Andreas et al., 2017; Mirchandani et al., 2021; Cideron et al., 2020) or shaping rewards or defining goals (Goyal et al., 2020, 2019; Hutsebaut-Buysse et al., 2019).

There has been some investigation of the issue of generalization in RL. In Song et al. (2019); Cobbe et al. (2019) the pattern of using training and test tasks to measure generalization was established and it is shown that agents with the same training performance can have significantly different test performance. This can be mitigated with regularization.

Some investigation of using language to generalize is found in Zhong et al. (2019); Narasimhan et al. (2018); Sodhani et al. (2021) where domain knowledge is added using language allowing generalization to new goals and environments. Jiang et al. (2019), uses

language to define goals for a low-level policy in a hierarchy. A high-level policy generates the low-level language goals in order to solve high-level goals. Generalization on held-out goals is tested and found to be greater when using language than a non-compositional representation suggesting the compositional feature of language allows the model to generalize better. In Joshi et al. (2020) the generalization properties of neural networks trained on the Travelling Salesman Problem is tested. While not utilizing language the directed investigation of generalization properties is informative. It was found that autoregressive decoding provides a more general inductive bias. This is comparable to the autoregressive inductive bias inherent to generative language models that can be used for generating language actions. It is also observed that training on variable size problems improves generalization to larger problems. In Oh et al. (2017) zero-shot task generalization was used as an evaluation method of generalization to longer and unseen language instructions.

In Schwartz et al. (2020) visual and semantic state-representations are compared with Synthetic Language for the Visdoom environment (Wydmuch et al., 2018). The metrics compared are mean reward by environment interactions and "nuisance" which is defined as random variables that affect the observed data but not the task being solved. They find that using language representations can sometimes achieve higher reward but they do not test generalization to other tasks.

3. Preliminaries

Reinforcement learning To define reinforcement learning (Sutton and Barto, 2018) we should first consider a Markov Decision Process (MDP) which is defined by a set of states S, a set of actions A_s available from state s, the probabilities of actions P_a as a function of current state s will lead to the next state s', and finally a set of rewards $R(s, a)$. γ is the discount rate. The reinforcement learning problem is to find a policy $\pi(a_t|s_t)$ which maximizes the discounted reward for all timesteps t is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

Proximal Policy Optimization (Schulman et al., 2017) is a policy gradient method that improves a parameterized policy directly. It switches iteratively between collecting data and updating the policy with mini-batch gradient descent. Let r_t be the reward at time t, $V_{(s_t)}$ and $V_{(s_{t+1})}$ the estimated value for states at timesteps s_t and s_{t+1} respectively. The generalized advantage (Schulman et al., 2018) for timestep t in range 0 to T is given by:

$$\begin{aligned} &\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ &\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{2}$$

The probability ratio of the current policy $\pi(\theta)$ and the old policy $\pi(\theta_{old})$ used for data collection in the current iteration is defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3}$$

The clipped policy loss $L_t(\theta)$ is defined as:

$$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{4}$$

where ϵ is a hyperparameter. Finally the value function loss is given by the squared error of the estimated value V_t and the target value V_t^{targ} which is simply the sampled discounted return.

$$L_t^{\text{VF}}(\theta) = (V_\theta(s_t) - V_t^{\text{targ}})^2 \tag{5}$$

4. Environment

Consider the task of tidying a messy room; Objects are randomly distributed in a room and need to be moved to the correct position to be considered tidy. Sometimes the nature of the objects means that they need to be treated in a specific way. For example, a toy block can be lifted but tables are too heavy and must be pushed. Sometimes the order in which objects are moved is determined by other objects. A stack of books on the lid of a box must be moved to the bookshelf (or at least out of the way) before toys can be put in the box. Tidy is subjective but using prior knowledge humans can do a reasonable job at tidying a room they have never seen before, containing objects they have never seen before. If the room grows in size becoming larger and messier (more scattered objects) it's still possible to eventually reach a tidy state by applying the same principle of moving objects to the correct position one at a time.

The *TidyBlock Environment* [1] presented here is inspired by the task of tidying. In a given rollout of the environment, there is a variable number of boxes, each uniquely colored. There is also a variable number of *big* and *small* colored blocks distributed among the boxes with each box having a limited capacity. An action consists of an action verb (*lift* / *push*), a target block (specified by size and color), a source box (specified by color) and a destination box (specified by color). Actions are valid if the action verb is appropriate for the chosen block, the chosen block exists in the source box and there is space in the destination box. A successful action results in a block being moved from one box to another, otherwise nothing happens. Blocks of different sizes require different action verbs to move them: Large blocks must be *pushed* while small blocks must be *lifted*. Sometimes actions are blocked due to box capacity. For example, a small block can not be moved to a full box even if it is of the correct color until another block is moved out of the box. Similarly, big blocks require twice the space of small blocks so there must be adequate space in the destination box. Ordering and position of blocks within boxes are not considered when checking if a move is valid. The environment is solved when all the blocks are moved into boxes of matching colors which returns a reward of 1. There is a small negative reward of -0.01 for each step to encourage finding the shortest path to a solution. A visualized example of a successful rollout is shown in Fig. 1.

The environment always resets to an unsolved yet solvable state. The colors used for the boxes and blocks can be fixed after resetting or they can be sampled from a set of 48 training colors. There is also a test set of 39 held-out colors used for testing generalization.

Technically the environment is implemented in a vectorized fashion in PyTorch (Paszke et al., 2019), making it is easy to scale to a very large number of environments on a CPU or GPU. This was motivated by McCandlish et al. (2018) where it was shown that models can retain data efficacy while training on large batches to speed up training.

1. https://gitlab.com/ngoodger/tidy-block-env

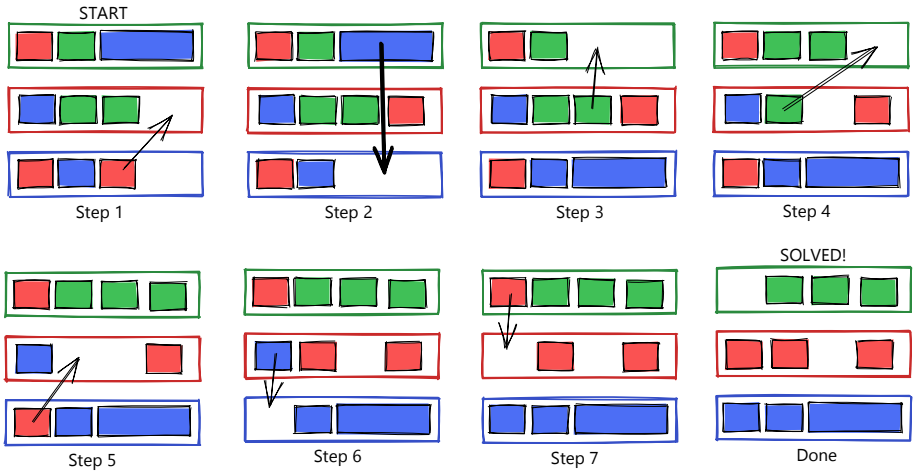


Figure 1: Example environment and solution (Shown visually to improve understanding, no visual state implemented currently). The lift actions are indicated by the thin arrows and thick arrow indicates a push action. Ordering does not matter. Box capacity can not be violated.

4.1. State Representation

There are two primary implementations of state in the environment. An example state of the environment is visualized in Fig. 2.

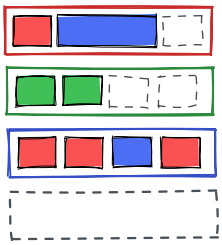
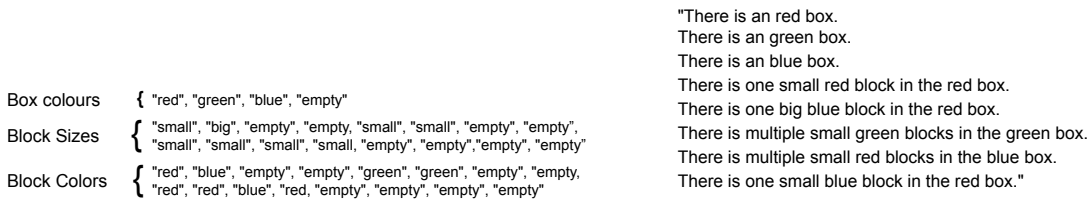


Figure 2: Example state visualized. There are 3 boxes of colors red, green, blue and 1 non-existent box which is included in the state for the generalization tasks. There is 1 big blue block, 1 small blue block, 4 small red blocks and 2 small green blocks distributed among the 3 existent boxes.

Vector-state representation. This concatenates a box-state vector, a block-size-state vector, and a block-color vector together. The box-state vector has a *color* token for each box and an *empty* token used if the box is not present. The block size vector has a token for each position in each box. The values include *empty* token, *big* token and a *small* token (only one space is used regardless of size which is calculated separately). The block-color-

state vector includes a token for each possible space in each box. Again an *empty* token is used when no block exists and otherwise, it takes the blocks appropriate *color* token. The vector-state with the tokens converted to words for visualization is shown in Fig. 3(*a*).



(*a*) Vector-state Example (*b*) Language-state Example

Figure 3: Example state representations corresponding to the visualized example in Fig. 2. In this example the environment is configured for a maximum of 4 boxes and possible 4 blocks per box which is why there are "empty" values in the vector-state.

Language-state representation. Uses synthetic language in that it conforms to a specific template. This means learned behavior could not be expected to generalize to natural language descriptions of the same environment. However, it has the advantage of allowing us to carefully control variables in the results and deepen our understanding of which parts of the representation are important. As with the vector-state, the boxes are first described before their contents. There is an option to omit redundant facts to avoid exhaustively describing possible states by removing statements like: "There are no red blocks in the green box." It can also be configured to shuffle the order of statements. In the case where there are two or more of a particular block type the token "multiple" replaces "one". An example of the language state is shown in Fig. 3(*b*).

4.2. Action Representation

There are also two action representations:
discrete-action is where every combination of action-verb, block-size, block-color, start-box-color, end-box-color is mapped to a separate discrete action.
language-action has two versions. The first uses a full template: "{*verb*} *the* {*block_size*} {*block_color*} *block from the* {*start_box_color*} *box to the* {*end_box_color*} *box .*" and a simple version where the fixed words are removed: "{*verb*} {*block_size*} {*block_color*} {*start_box_color*} {*end_box_color*}". The simple version with fewer words was created to limit the action-space as each word must be sampled from a vocabulary.

5. Experiments

Through this work, we are trying to compare generalization between representations. To approach this systematically we test zero-shot task generalization. This involves training using reinforcement learning on a training task and measuring achieved reward on several

test tasks. This is similar to the use of a training-set and test-set in supervised learning to evaluate generalization and the same as the approach used in Song et al. (2019); Cobbe et al. (2019). It allows us to test how well the inductive biases learned during training work on the new tasks. By performing this comparison we try to answer the following research questions: **1.** How well do vector-state representations generalize in comparison to language-state representations? **2.** How do discrete-action representations compare to auto-regressive language actions for generalization. **3.** How does the model architecture affect generalization when using language representations? **4.** How does the training data affect generalization? To quantitatively compare generalization we devised a number of tasks.

One training task and four test tasks are defined:

1. **Training task**: Maximum of 3 boxes (and 3 colors sampled from the training set). The boxes contain 4 spaces where blocks can potentially be generated (space-permitting).
2. **More Blocks Task**: Increases the size of the boxes from 4 to 8. This increases the number of steps to solve the problem and the size of the state. This change does not affect the action spaces. Tests the agent's ability to generalize to a larger version of the task.
3. **More Boxes Task**: Adds an additional box and color (sampled from the training set). The main difference is that there are objects of 4 colors as opposed to just 3 in the training task. Additional blocks can be generated in the new box increasing the problem size. The valid action-space will also grow because new colors and boxes will need to be selected to solve the environment. Tests the agent's ability to generalize to a more complex version of the task.
4. **New Colors Task**: Samples the colors from the test set rather than the training set. The problem size remains the same but the colors which are used as adjectives to identify the blocks and boxes are switched to held-out test colors. Tests the agent's ability to generalize to unseen entities.
5. **Combined Generalization Task**: The modifications from all 3 generalization test tasks are combined in a single task.

During training and testing the states and action spaces are padded with *empty* tokens to the maximum size. For example, the vector-state size increases when testing increasing the number of colors/boxes from 3 to 4 so the model is trained with size 4 states however the states relating to the 4th box correspond to the *empty* token throughout training.

Training was run for 3k **Proximal Policy Optimization (PPO) algorithm** iterations for all experiments.

5.1. Model Architectures

The state and action representations are tested with five different model architectures. For all architectures and state representations, the first layer of the model is a randomly initialized embedding space.

1. A Feed-forward network is used to map the **vector-state** to a **discrete-action**. This choice of architecture for use with the vector-state was made because the vector-state is not inherently ordered like language. It is processed in parallel and finally, output to a softmax normalized distribution across discrete actions. A similar feed-forward critic is used to approximate the value function.

2. A Recurrent network using an LSTM (Hochreiter and Schmidhuber, 1997) is used for the **language-state** and the **discrete-action**. A recurrent model was chosen because the language-state is ordered and this model has a sequential bias. The language-state is input to the model and the final hidden state is input to a feed-forward network to define a softmax normalized distribution across discrete actions. A similar recurrent critic is used to approximate the value function.

3. A sequence to sequence model (Sutskever et al., 2014) with three different heads were used for the **language-state** to **language-action**. These models take the language state as input and auto-regressively produce a language-action by sampling greedily. They all use a recurrent critic to approximate the value function:

   (a) A generative head which samples a token from the distribution defined by a softmax of tokens in the synthetic language.

   (b) A pointer head (Vinyals et al., 2015) samples a token from the distribution defined by a softmax over the language-state input tokens. This model is extractive in that it selectively extracts tokens from the input to generate the output. Additional tokens required for the language-action but not present in the language-state are prepended to the language-state so that they can be selected for the output.

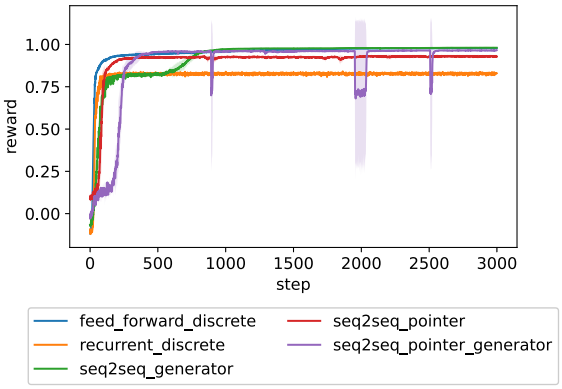
   (c) A hybrid head including both a generative head and a pointer head similar to the implementation in Gulcehre et al. (2016). It includes an additional feed-forward network conditioned on the decoder hidden state that defines the distribution for switching between the generative or pointer heads. In this implementation, the additional tokens required for the language-action are not prepended to the state so the model must learn to use the generative head to produce this part of the action sequence.

5.2. Language-action Constraint

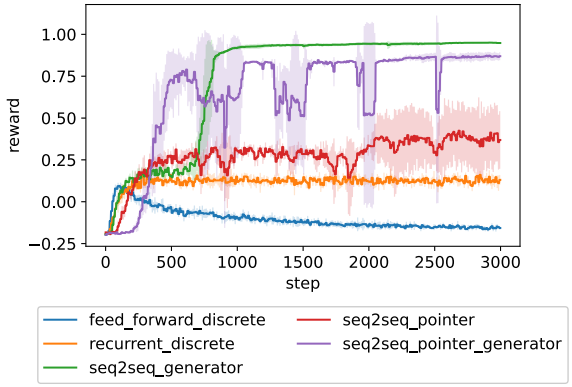
Despite the limited vocabulary, the language action-space is still very large. With 110 words in the vocabulary and 14 words in the action template the action-space is 4E28. To reduce the language-action-space to a range similar to the discrete-action-space we mask invalid actions. For example, when choosing the second word which is block-size, all logits for the generator head and/or the pointer head representing tokens other than block size are set to a large negative number before going through softmax normalization preventing sampling.

5.3. Results

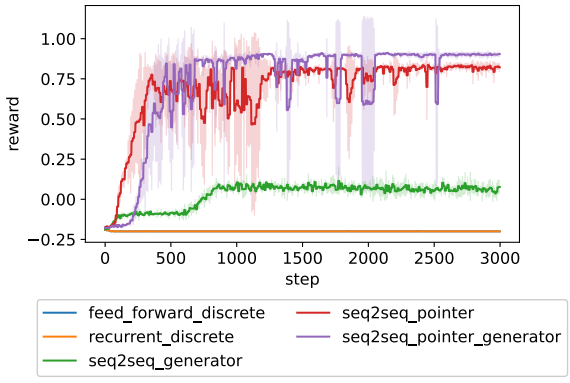
The rewards obtained on each task are shown in Fig. 4. These results were obtained using the *Many Sizes* training regime (See Appendix. A for details on the training implemen-



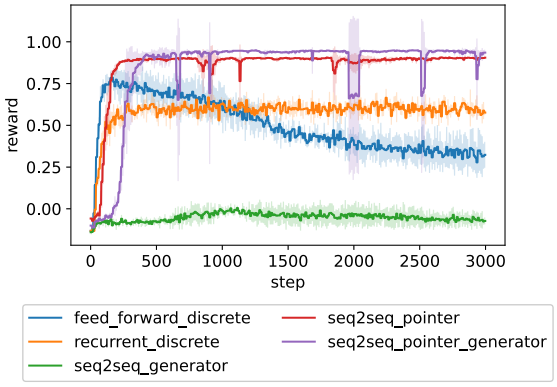
(*a*) Training Task reward



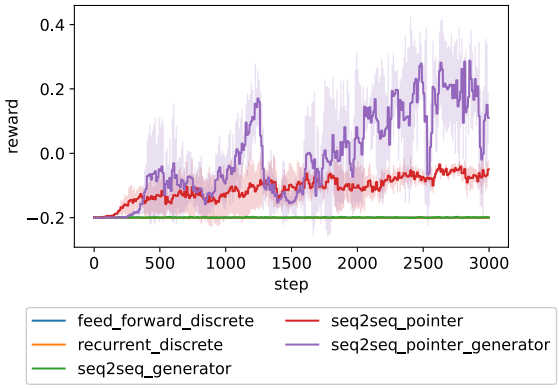
(*b*) More Blocks Task reward



(*c*) More Boxes Task reward



(*d*) New Colors Task reward



(*e*) Combined Task reward

Figure 4: Rewards obtained using the *Many Sizes* training regime. Runs were repeated 3 times and the shaded area represents one standard deviation.
Middle Left: Increase the maximum generated blocks in each box from 4 to 8.
Middle Right: Number of boxes and colors in the task increased from 3 to 4.
Bottom Left: Colours sampled from unseen test set rather than training set.
Bottom Right: Combination of all 3 test task modifications.

tations tested) and the simple language template where language-actions are used. This approach produced the best results for most architectures although the best result on the combined generalization was obtained with the *Many Colors Many Sizes* training regime. Results using the other training regimes can be seen in Appendix. B. All state and action representations were able to learn to solve the training task, see Fig. 4(*a*). However, reward on test tasks varied significantly between representations and architectures.

Testing generalization for the *More Blocks Task* in Fig. 4(*b*) showed that all configurations learned to generalize to different degrees. This can be explained by considering the state spaces. The vector state-space remains partially consistent with the training data except that some of the always empty values will now be populated. For the language-state the only difference is that for this test task it can be larger. Both the discrete and language action spaces remain the same in that there are no new colors or boxes. The sequence to sequence generative model worked best. This may be because the pointer architecture struggled to select the correct output from the larger state. One explanation for the difference between the recurrent-discrete model and the sequence to sequence model is that the learned auto-regressive language action generalizes better because the output is compositional and generated over several steps while the discrete-action must be output in a single step. The feed-forward network using the vector-state and discrete-action reward decreased with increasing training steps, indicating that the model is overfitting.

Testing generalization to the *More Boxes Task* shown in Fig. 4(*c*) resulted in zero reward for the recurrent-discrete and feed-forward model using discrete actions. The discrete action-space requires the use of untrained actions while the language-action can generalize to new compositions. For the sequence to sequence models, the extractive models outperformed the generative model. We think this is due to the additional color present in this task.

For the *New Colors Task* in Fig. 4(*d*) we see that the feed-forward model using the vector-state and discrete-action initially learns, however as the number of steps increases performance drops due to overfitting on the training task. The generative model fails to improve much in performance which is expected because the new colors required in the output have not been trained. The recurrent discrete model quickly improves but the mean reward plateaus. It is surprising that it achieves a reward higher than the generative model as its output also refers to new actions for this task. The difference may be explained by the discrete-action-space still using the same range as during training. The actions just refer to different colors unlike the generative model which must generate untrained tokens. The pointer network and pointer-generator networks achieve high reward because they have learned a program that extracts the new color adjective from the input and therefore can generalize to this task.

Finally on the hardest *Combined Task* shown in Fig. 4(*e*) only the language state and action models with an extractive capability achieved any reward and of those the pointer-generator achieved the highest reward. This seems to indicate that it has learned to synergize the best features from both the pointer and generative heads of the model.

The experiments used the same hyperparameters for all runs. All details of the implementation can be seen in the open-source implementation[2].

2. https://gitlab.com/ngoodger/tidy-block-generalization-comparison

6. Discussion

Agents using different representations and architectures resulted in similar training reward. If the motivation of this paper was maximizing training reward on this environment we might conclude they were equivalent. However, the results show that the achieved reward on the zero-shot generalization tasks varied significantly between representations and models. We found that using a language state-space in the recurrent model compared to a vector-state in the feed-forward model aligned the learned inductive bias such that it prevented overfitting to the training task. Further, we found that by utilizing a language-state and language-action space together with extractive models we were able to generalize to new entities, more complex and larger problems than those used for training.

We believe that the advantage of the language state and action spaces is related to compositionality. The state is made of sub-expressions that contextualize the information while in the vector-state the information is only contextualized by its position. Similarly the language-action is also compositional. The lexical parts of the action can be learned and then combined in new ways to generate new unique actions, unlike the discrete-action where every unique action must be trained independently. Additionally, the language-action is generated auto-regressively so it will be conditioned on the previously generated tokens in the action-sequence helping to generate valid sequences.

These results indicate that using language as a representation for states and actions may be a viable approach to build more general agents. Increased generalization is an attractive feature as it can allow increased robustness to out-of-distribution states and actions, adaptability to new tasks, or even adaptability to more complex tasks.

It's important to note that these results are also a function of model architecture and environment. So we can not be certain that the behavior would generalize to other architectures or environments without testing them. A limitation of the approach outlined in this paper is the action constraint on the language model's action-space, without which this approach does not find valid actions. It would be interesting to explore whether bootstrapping the model with supervised training data or using other RL techniques could allow for training sequence to sequence models in an RL context without this constraint.

Acknowledgments

We would like to thank Guillem Orellana for sharing their pointer network implementation[3]

References

David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 10–19. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/abel18a.html.

3. https://github.com/Guillem96/pointer-nn-pytorch

David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*, page 119–125, USA, 2002. American Association for Artificial Intelligence. ISBN 0262511290.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 166–175. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/andreas17a.html.

S.R.K. Branavan, Luke Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1268–1277, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P10-1129.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, L. Willems, Chitwan Saharia, T. Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *ICLR*, 2019.

Geoffrey Cideron, Mathieu Seurin, Florian Strub, and Olivier Pietquin. Higher: Improving instruction following with hindsight generation for experience replay. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 225–232, 2020. doi: 10.1109/SSCI47803.2020.9308603.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning, 2019.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. In Tristan Cazenave, Abdallah Saffidine, and Nathan Sturtevant, editors, *Computer Games*, pages 41–75, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24337-1.

Mark Depristo, Hughes Hall, and Robert Zubek. Being-in-the-world. 12 2001.

Ralph W. Fasold and Jeff Connor-Linton. *An Introduction to Language and Linguistics*. Cambridge University Press, 2 edition, 2014. doi: 10.1017/CBO9781107707511.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2385–2391. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/331.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. In *4th Conference on Robot Learning (CoRL)*, November 2020. URL http://www.cs.utexas.edu/users/ai-labpub-view.php?PubID=127846.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1014.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Fast task-adaptation for tasks labeled using natural language in reinforcement learning. *CoRR*, abs/1910.04040, 2019. URL http://arxiv.org/abs/1910.04040.

YiDing Jiang, Shixiang (Shane) Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/0af787945872196b42c9f73ead2565c8-Paper.pdf.

Chaitanya K. Joshi, Quentin Cappart, L. Rousseau, T. Laurent, and X. Bresson. Learning tsp requires rethinking generalization. *ArXiv*, abs/2006.07054, 2020.

George Konidaris. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2018.11.005. Artificial Intelligence.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6309–6317. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/880.

Alana Marzoev, S. Madden, M. Kaashoek, Michael J. Cafarella, and Jacob Andreas. Unnatural language processing: Bridging the gap between synthetic and natural language data. *ArXiv*, abs/2004.13645, 2020.

Sam McCandlish, J. Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *ArXiv*, abs/1812.06162, 2018.

Suvir Mirchandani, Siddharth Karamcheti, and Dorsa Sadigh. Ella: Exploration through learned language abstraction. *arXiv preprint arXiv:2103.05825*, 2021.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1001.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning. *J. Artif. Int. Res.*, 63(1):849–874, September 2018. ISSN 1076-9757. doi: 10.1613/jair.1.11263.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2661–2670. PMLR, 06–11 Aug 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

Erez Schwartz, Guy Tennenholtz, Chen Tessler, and Shie Mannor. Language is power: Representing states using natural language in reinforcement learning. *arXiv: Computation and Language*, 2020.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. *ArXiv*, abs/2102.06177, 2021.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. *CoRR*, abs/1912.02975, 2019. URL http://arxiv.org/abs/1912.02975.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Harm van Seijen, Shimon Whiteson, and Leon Kester. Efficient abstraction selection in reinforcement learning. *Computational Intelligence*, 30(4):657–699, 2014. doi: https://doi.org/10.1111/coin.12016.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018.

Xingdi (Eric) Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. Counting to explore and generalize in text-based games. In *ICML 2018*, pages 1–12, June 2018. URL https://www.microsoft.com/en-us/research/publication/counting-to-explore-and-generalize-in-text-based-games/.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. RTFM: Generalising to Novel Environment Dynamics via Reading. *arXiv e-prints*, art. arXiv:1910.08210, October 2019.

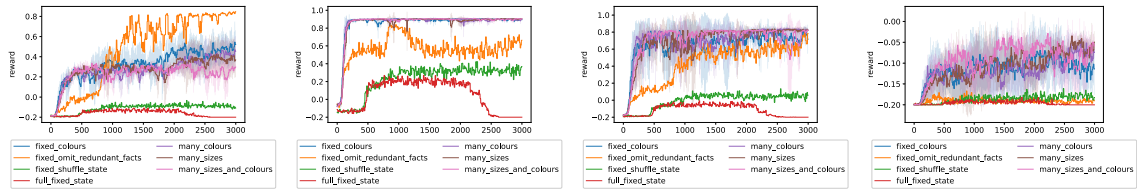
Appendix A. Training data configuration

The training environment is configured with a maximum of 3 boxes/colors sampled from a training set. With a possible 4 spaces in each box. A number of different training regimes within this specification were tested:

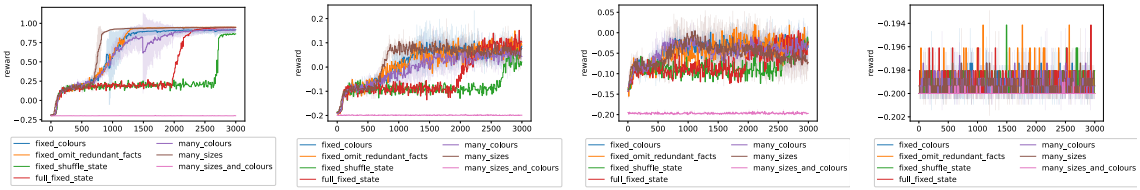
1. *Fixed Colours*: Always uses the first 3 colors of the training set so they are consistent throughout training. Always 3 boxes present.
2. *Many Colours*: Randomly sample training colors from 20 in the training set so the model sees different colors in each rollout. Always 3 boxes present.
3. *Many Sizes*: Always uses the first 3 colors of the training set so they are consistent throughout training. The number of boxes is varied from 2 to 3 during training.
4. *Many Sizes and Colors*: Randomly sample training colors from 20 in the training set so the model sees different colors in each rollout. The number of boxes is varied from 2 to 3 during training.
5. *Fixed Colors Full Fixed State*: Same as *Fixed Colours* except that for language-state the option to omit-redundant facts and shuffle state is turned off.
6. *Fixed Colors Omit Redundant Facts*: Same as *Fixed Colours* except that for language-state the option to omit-redundant facts is turned off.
7. *Fixed Colors Shuffle State*: Same as *Fixed Colours* except that for language-state the option to shuffle state is turned off.

Appendix B. Training data affect on generalization

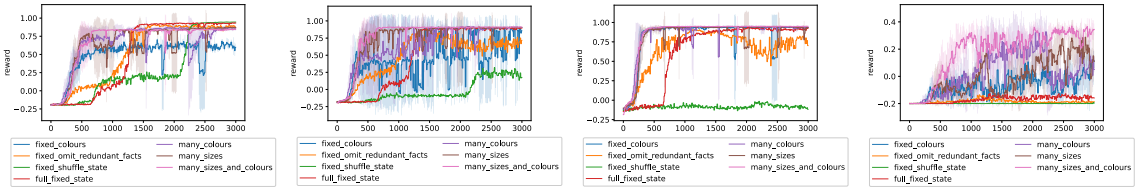
Zero-shot generalization reward under different training data regimes compared for the state-representations and model architectures tested can be seen in Fig. 5.



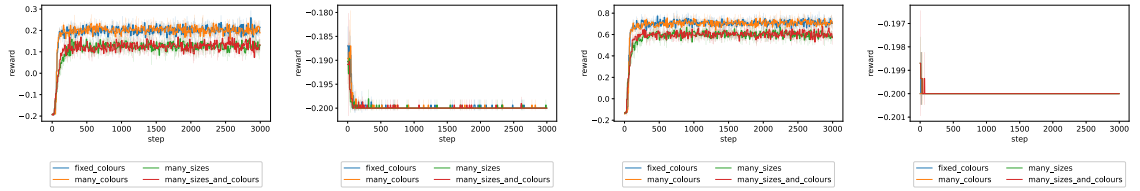
(a) Pointer / More Blocks Task (b) Pointer / More Boxes Task (c) Pointer / New Colors Task (d) Pointer / Combined Task



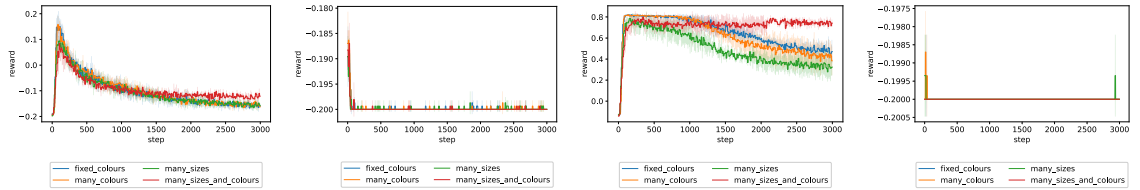
(e) Generator / More Blocks Task (f) Generator / More Boxes Task (g) Generator / New Colors Task (h) Generator / Combined Task



(i) Pointer-generator: More Blocks Task (j) Pointer-generator / More Boxes Task (k) Pointer-generator / New Colors Task (l) Pointer-generator / Combined Task



(m) Recurrent-discrete / More Blocks Task (n) Recurrent-discrete / More Boxes Task (o) Recurrent-discrete / New Colors Task (p) Recurrent-discrete / Combined Task



(q) Feed-forward / More Blocks Task (r) Feed-forward / More Boxes Task (s) Feed-forward / New Colors Task (t) Feed-forward / Combined Task

Figure 5: Test task rewards