Learning to Follow Language Instructions with Compositional Policies

Vanya Cohen,[1†*] **Geraud Nangue Tasse,**[2†] **Nakul Gopalan,**[3]
Steven James,[2] **Matthew Gombolay,**[3] **Benjamin Rosman**[2]

[1]The University of Texas at Austin
[2]University of the Witwatersrand
[3]Georgia Institute of Technology
[†] Equal contribution
vanya@utexas.edu, geraudnt@gmail.com, nakul_gopalan@gatech.edu, steven.james@wits.ac.za,
matthew.gombolay@cc.gatech.edu, benjamin.rosman1@wits.ac.za

Abstract

We propose a framework that learns to execute natural language instructions in an environment consisting of goal-reaching tasks that share components of their task descriptions. Our approach leverages the compositionality of both value functions and language, with the aim of reducing the sample complexity of learning novel tasks. First, we train a reinforcement learning agent to learn value functions that can be subsequently composed through a Boolean algebra to solve novel tasks. Second, we fine-tune a seq2seq model pretrained on web-scale corpora to map language to logical expressions that specify the required value function compositions. Evaluating our agent in the BabyAI domain, we observe a decrease of 86% in the number of training steps needed to learn a second task after mastering a single task. Results from ablation studies further indicate that it is the *combination* of compositional value functions and language representations that allows the agent to quickly generalize to new tasks.

Introduction

Natural language provides an intuitive way for people to specify tasks and instructions to satisfy goals. However, instruction following is a difficult problem for artificial agents because they need to simultaneously learn the (a) meaning of the instructions to solve the task, (b) representation of the world in which the task is to be solved, and (c) sequence of actions that will lead to task completion. Learning to follow instructions is even more challenging in the multitask setting, where an agent must also acquire knowledge that will allow it to generalize to new, unseen tasks.

One of the most common approaches to this problem is to encode a language command into a real-valued vector embedding. The language embedding, together with the agent's current state, is then used to parametrize the policy that controls the agent (Blukis et al. 2019; Chaplot et al. 2018; Tambwekar et al. 2021). These end-to-end methods succeed in the presence of a simulator, but require large amounts of data. These methods also suffer from generalization issues as the agents require a large number of samples from the environment for every novel task presented (Lake and Baroni 2018).

To overcome the generalization issue, we leverage the principle of *compositionality*, which states that a complex expression's meaning is defined by the meanings of its constituent expressions and the rules used to combine the meanings of these expressions (Szabó 2020). In particular, we exploit the compositional nature of task specifications and their solutions (in the form of a special type of value function). Our approach is possible as both natural language and the learned value functions exhibit the property of compositionality.

In our work, the constituent expressions are possible atomic goal specifications. These goal specifications have *value functions* which define the agent's behavior. The value functions can be combined using the rules of Boolean algebra to solve novel tasks. We use machine translation approaches to map linguistic task specifications onto corresponding logical expressions, which are then used to combine value functions to solve the specified tasks.

First, we train task-specific value functions in the manner of Nangue Tasse, James, and Rosman (2020) for a set of preselected atomic tasks from the BabyAI environment (Chevalier-Boisvert et al. 2019). Additionally, we fine-tune a T5 model (Raffel et al. 2020) using reinforcement learning to translate natural language instructions into logical expressions that specify the compositions of the task-specific value functions. The compositional value functions are then used by the agent to form policies for acting in the environment. Finally, the agent's collected environment rewards are used as a signal to improve the translation model.

We evaluate the agent by learning a set of compositional tasks in series and observe the number of training steps needed to learn each additional task in the series. Further, we perform ablation studies to understand the effect of model pretraining on web-scale corpora and the stochastic nature of feedback from the environment on sample complexity. With a pretrained T5 model (Raffel et al. 2020), the mean number of training steps needed to learn an additional task drops by 86% after learning just one task. Without model pretraining, the mean training steps drops by only 6%, although the number of training steps continues to drop as more tasks are learned.

When learning all available tasks in the environment, the number of training steps needed to learn the final task decreases by 98% for the pretrained model, compared to only

[*]Work performed while an independent researcher.

80% for the randomly initialized model. In terms of the fractional improvement in the training steps needed to learn the final task, the pretrained model provides a $10\times$ improvement (2% versus 20%) over the randomly-initialized model.

Overall, our paper makes the following contributions:

- We connect pretrained compositional policies to a translation model capable of mapping natural language statements to logical expressions specifying compositions of those policies.
- We demonstrate empirically that pretraining of the translation model on non-task-specific data is sufficient to generate compositional expressions. We found the T5 (Raffel et al. 2020) language model sufficient to generate novel Boolean expressions given language commands. This ability to generate novel expressions leads to a significant reduction in samples from the environment after the pretrained model learns to solve just a single task.
- We detail learning results for 18 tasks in the BabyAI showing that compositional policies, along with a pretrained model, lead to substantial savings in the number of samples required to learn novel tasks.
- We provide ablation results with and without a pretrained language model, and with and without our compositional policies evaluated in the environment.

Background

We are interested in the multitask setting, where an agent is required to solve a series of related tasks, modelled by a unordered set of Markov Decision Processes (MDPs). An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, p, r \rangle$, where (i) $\mathcal{S}$ is the state space, (ii) $\mathcal{A}$ is the action space, (iii) p is a Markov transition kernel $(s, a) \mapsto \rho_{(s,a)}$ from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{S}$, and (iv) r is the real-valued reward function bounded by $[r_{\text{MIN}}, r_{\text{MAX}}]$. We focus here on stochastic shortest path problems (Bertsekas and Tsitsiklis 1991), where an agent must optimally reach a set of absorbing goal states $\mathcal{G} \subseteq \mathcal{S}$.

We assume that all tasks share the same state space, action space and dynamics, but differ in their reward functions. More specifically, we define the background MDP $M_0 = \langle \mathcal{S}_0, \mathcal{A}_0, p_0, r_0 \rangle$ with its own state space, action space, transition dynamics and background reward function. Any individual task τ is specified by a task-specific reward function r_τ that is non-zero only for transitions entering a state in $\mathcal{G}$. The reward function for the resulting MDP is then simply $r_0 + r_\tau$.

Given a task, the agent's aim is to learn an optimal Markov policy π from $\mathcal{S}$ to $\mathcal{A}$. A given policy π induces a value function $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} r(s_t, a_t)\right]$, representing the expected return obtained under π starting from state s. The *optimal* policy π^* is the policy that obtains the greatest expected return at each state: $V^{\pi^*}(s) = V^*(s) = \max_\pi V^\pi(s)$ for all $s \in \mathcal{S}$. A related quantity is the Q-value function, $Q^\pi(s, a)$, which defines the expected return obtained by executing a from s, and thereafter following π. Similarly, the optimal Q-value function is given by $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. Finally, we define a *proper policy* to be a policy that is guaranteed to eventually reach $\mathcal{G}$ (James and Collins 2006; Van Niekerk et al. 2019). We assume the value functions for improper policies—those that never reach absorbing states—are unbounded from below.

Logical Composition of Tasks and Value Functions

Nangue Tasse, James, and Rosman (2020) recently proposed a framework for agents to apply logical operations—conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$)—over the space of tasks and value functions. This is achieved by first defining the goal-oriented reward function $\bar{r}$ which extends the task rewards r to penalise an agent for achieving goals different from the one it wished to achieve:

$$\bar{r}(s, g, a) = \begin{cases} \bar{r}_{MIN} & \text{if } g \neq s \in \mathcal{G} \\ r(s, a) & \text{otherwise,} \end{cases} \tag{1}$$

where $\bar{r}_{MIN} \leq \min\{r_{\text{MIN}}, (r_{\text{MIN}} - r_{\text{MAX}})D\}$, and D is the diameter of the MDP (Jaksch, Ortner, and Auer 2010). Equation 1 specifies the reward function for the agent to achieve all reachable goals.

Using Equation 1, we can define the related goal-oriented value function as:

$$\bar{Q}(s, g, a) = \bar{r}(s, g, a) + \int_{\mathcal{S}} \bar{V}^{\bar{\pi}}(s', g) p_{(s,a)}(ds'), \tag{2}$$

where $\bar{V}^{\bar{\pi}}(s, g) = \mathbb{E}_{\bar{\pi}} \left[\sum_{t=0}^{\infty} \bar{r}(s_t, g, a_t)\right]$.

If a new task can be represented as the logical expression of previously learned tasks, Nangue Tasse, James, and Rosman (2020) prove that the optimal policy can immediately be obtained by composing the learned goal-oriented value functions using the same expression.

For example, consider the `PickUpObj` domain shown in Figure 1, where an agent has learned to pick up the yellow object (task Y) and to pick up the ball (task B). We can then provably solve the tasks defined by their union, intersection, and negation as follows (we omit the value functions' parameters for readability):

$$\bar{Q}^*_{Y \vee B} = \bar{Q}^*_Y \vee \bar{Q}^*_B \coloneqq \max\{\bar{Q}^*_Y, \bar{Q}^*_B\}$$

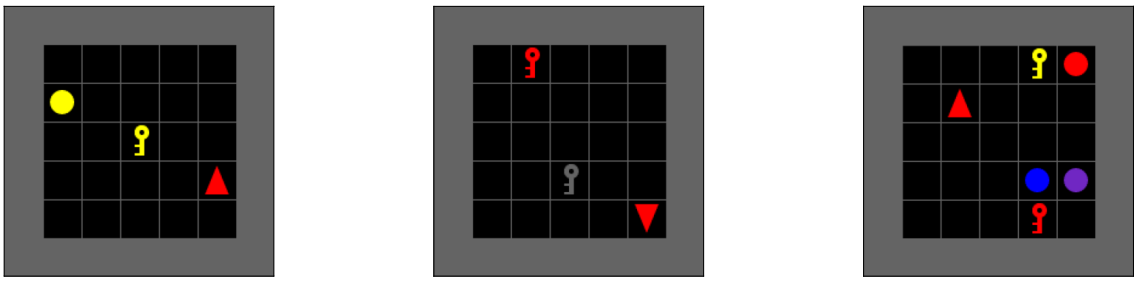
$$\bar{Q}^*_{Y \wedge B} = \bar{Q}^*_Y \wedge \bar{Q}^*_B \coloneqq \min\{\bar{Q}^*_Y, \bar{Q}^*_B\}$$

$$\bar{Q}^*_{\neg Y} = \neg \bar{Q}^*_Y \coloneqq \left(\bar{Q}^*_{MAX} + \bar{Q}^*_{MIN}\right) - \bar{Q}^*_Y,$$

where $\bar{Q}^*_{MAX}$ is the goal-oriented value function for the maximum task where $r = r_{\text{MAX}}$ for all $\mathcal{G}$. Similarly $\bar{Q}^*_{MIN}$ is goal-oriented value function for the minimum task where $r = r_{\text{MIN}}$ for all $\mathcal{G}$. We henceforth refer to these goal-oriented value functions as *compositional value functions*.

Translation with Transformer Models

Recent progress in natural language processing (NLP) has demonstrated the effectiveness of large-scale generative pretraining and subsequent fine-tuning on downstream tasks, such as translation, question answering, and classification (Devlin et al. 2018; Peters et al. 2018; Radford et al. 2018). Subsequent work has shown that scaling both model parameters and pretraining corpus size leads to better transfer learning and generalization (Radford et al. 2019).



(a) Language command "pick up the yellow ball" with corresponding logical expression `pickup_yellow` ∧ `pickup_ball`.

(b) Language command "pick up the red key" with corresponding logical expression `pickup_red` ∧ `pickup_key`.

(c) Language command "pick up the red key" with corresponding logical expression `pickup_red` ∧ `pickup_key`.

Figure 1: Examples of tasks in the BabyAI `PickUpObj` environment. For each task, there is a target and distractor object. The agent is represented by the red triangle. We also investigate performance when four distrator objects are present. a) The agent must pick up the yellow ball but not the yellow key. To solve this level, the agent must use the intersection of the "pickup" value functions for "yellow" and "ball". b) The agent must pick up the red key while not picking up the grey key. Solving this level requires using the intersection of the "pickup" value functions for "red" and "key". c) The agent must pick up the red key while not being distracted by the yellow key and red ball. Solving this level requires using the intersection of the "pickup" value functions for "red" and "key".

To map between natural language instructions and Boolean expressions specifying policy compositions, we utilize the T5 sequence-to-sequence model (Raffel et al. 2020) based on the Transformer architecture (Vaswani et al. 2017). The model is pretrained using an unsupervised learning objective on the Colossal Clean Crawled Corpus (C4) (Raffel et al. 2020), a filtered version of the Common Crawl.[1] The C4 corpus contains 750GB of text, the vast majority of which is fluent English. Raffel et al. perform exhaustive ablation studies to develop their pretrained models, which offer good performance on a variety of NLP tasks including translation.

Transformer-based models use the self-attention mechanism (Vaswani et al. 2017) to build sequence representations of text inputs, and to transform those representations into probability distributions over text outputs. As with the original Transformer architecture, the T5 model is composed of both an encoder and decoder stack of self-attention layers to map input sequences to output sequences. Self-attention layers receive input embeddings from lower layers and compose them to form higher-level embeddings.

Methods

Our agent learns to combine pretrained compositional policies by translating BabyAI "mission" statements (e.g. "pick up the blue box") into Boolean algebraic expressions which specify compositions of policies. We limit our investigation to intersections of policies, although the Boolean compositional policies also allow for disjunction and negation. Training begins with training a compositional policy to solve each of the task primitives. The agent can navigate to objects in the BabyAI domain described by three type attributes $\{box, ball, key\}$ and six color attributes $\{red, blue, green, grey, purple, yellow\}$, which yields eighteen possible navigation tasks.

[1]https://commoncrawl.org

Learning the Compositional Value Functions

Like Nangue Tasse, James, and Rosman (2020), we use deep Q-learning (Mnih et al. 2015) to learn the Q-function for each goal of the compositional value functions. We represent each compositional value function $\bar{Q}^*$ with a list of $|\mathcal{G}|$ DQNs, such that the Q-function for each goal $Q^*_g(s,a) := \bar{Q}^*(s,g,a)$ is approximated with a separate DQN.[2]

For each task, the agent starts training after 1000 steps of random exploration to populate an experience replay buffer and a goal buffer (set of reached terminal states). For each episode, the agent samples a random goal from the goal buffer and uses ϵ-greedy to act in the environment. For each action, a, that the agent takes in each state, s, it receives goal-oriented rewards (Equation 1) given by:

$$\bar{r}(s,g,a) = \begin{cases} -0.1 & \text{if } g \neq s \in \mathcal{G} \\ r(s,a) & \text{otherwise,} \end{cases}$$

where task reward $r(s,a) = 2$ for picking up the correct object and $r(s,a) = -0.1$ everywhere else.[3] The episode terminates after the agent picks up any object. The agent's compositional value function is then trained per episode using the collected experience. Training ends once the agent reaches a success rate of at least 0.98. For lower success rates, the compounding effect of composing sub-optimal policies negatively impacts the translation model's learning. For full details on policy performance please refer to Table 1 in Appendix A.

[2]Architecture and hyperparameter details are presented in the appendix.

[3]We used $\bar{r}_{MIN} = r_{MIN} = -0.1$ since that is the simplest choice and it did not result in any discernible change in the success rate of the composed policies.

Translating Missions to Boolean Expressions

We select the smallest of the publicly released T5 models as the pretrained model for our experiments (Table 2): the T5-small model which has 60 million parameters and is sufficient for our tasks based upon our empirical exploration.

We translate natural language task instructions to Boolean algebraic expressions that represent the task's value function. The Boolean algebraic expressions have tokens and operators. The legal operators are union (disjunction), intersection (conjunction) and negation. The tokens in the Boolean algebraic expressions represent goal value functions that can be composed to create richer tasks. For the BabyAI domain, these tokens represent value functions for picking up objects by type $\{pickup\_box, pickup\_ball, pickup\_key\}$, picking up objects by color $\{pickup\_red, pickup\_blue, pickup\_green, pickup\_grey, pickup\_purple, pickup\_yellow\}$, the logical operators and end-of-sentence tokens $\{and, < s >\}$.

Both the input and output tokens are byte-pair encoding (BPE) subword units (Sennrich, Haddow, and Birch 2016) learned from the C4 training corpus. For example, the Boolean task algebra token "$pickup\_purple''$ is represented by the subwords "$pickup''$, "$\_''$, "pur'', and "ple''. Each of the tokens in the Boolean task algebra is represented by one or more BPE subword units. Instead of sampling from the BPE subword units directly, continuations are sampled from the distribution of tokens in the Boolean task algebra. If BPE subwords were sampled directly, at the beginning of training the probability of outputting valid tokens from the Boolean task algebra would be vanishingly small. Decoding stops when the stop token is produced or more than three Boolean algebra tokens have been sampled. We use temperature-based sampling (Ackley, Hinton, and Sejnowski 1985) to produce translated sequences during training, and greedy sampling during evaluation. For translation model details see Table 2.

Given an input mission to the T5-small model (e.g. "pick up the red ball") we can sample a Boolean expression from its output distribution over tokens (e.g. `pickup_red and pickup_ball`). This expression is then parsed and validated for syntactic correctness by a Boolean algebra expression parser. The corresponding compositional value function is obtained as follows (we omit the value functions' parameters for readability):

$$\bar{Q}_{pickup\_red \wedge pickup\_ball} = \min\{\bar{Q}_{pickup\_red}, \bar{Q}_{pickup\_ball}\}$$

The full process for generating policies from task instructions is illustrated in Figure 2. Finally, the agent can maximise over the composed value function to act in the environment: $\pi(s) \in \arg\max_{a \in \mathcal{A}} \max_{g \in \mathcal{G}} \bar{Q}(s, g, a)$.

Baseline Model

The baseline is a non-compositional CNN-DQN (Mnih et al. 2015) conditioned on the input mission language. The model is a simplified version of the baseline used by Chevalier-Boisvert et al. (2019), and uses a CNN to extract image features and a Gated Recurrent Unit (GRU) (Cho et al. 2014) that takes the mission as input and outputs text features. The image and text features are then concatenated and passed through two fully-connected layers to compute the output Q-values.

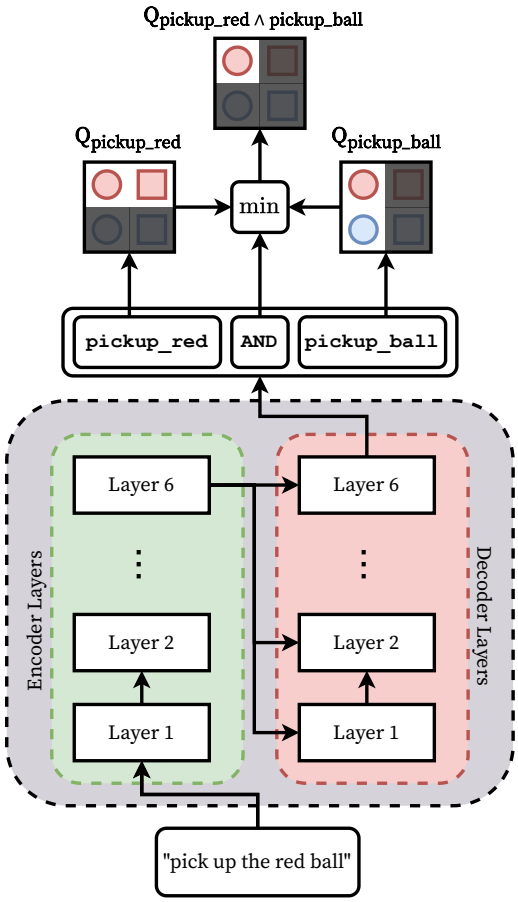


Figure 2: The T5-small model first translates the input mission command "pick up the red ball" into a Boolean expression, with variables representing the vocabulary of possible compositional value functions. Then the intersection of the value functions is computed, resulting in a value function for picking up a red ball in the environment.

In contrast to our method, the baseline is a joint model which learns a Q-function conditioned on both image state and language features. As such, its component value function and language representations are not pretrained. Our method learns both tasks and language separately and then learns to combine them compositionally. Likewise, the baseline model does not have explicit compositional structure and must instead learn to condition output Q-values on a combination of image and language features.

Experiments

To assess the agent's ability to generalize compositionally, we evaluate the agent as it learns to solve all available tasks in sequence. In each of ten trials, we randomly shuffle the order in which the 18 tasks are introduced and then train the agent to solve each task one at a time. At iteration 0 of

each new task, and every 100 training steps thereafter, the performance of the agent is evaluated using returns from 100 policy roll-outs. A task is considered solved if the agent successfully reaches the goal object in 95 out of 100 roll-outs, at which point the agent is presented with the next task in the sequence. During training, Boolean policy expressions are sampled from the translation model using temperature-based sampling with a temperature of 1.0 to inject randomness in the sampling process. However, when evaluating whether the agent has successfully solved the task, expressions are generated through greedy sampling to only select the most likely continuation tokens without noise.

Learning Tasks in Series

We adopt four experimental settings to investigate the impact of the different components in our overall system.

First, we consider two strategies for initializing the translation model: we use either 1) the pretrained T5-small model, or 2) its randomly initialized instantiation provided by Wolf et al. (2020). We also consider the effect of the pretrained policies on the overall performance of the agent by 1) using the returns of the policies executed in the environment as a learning signal for the language model, or 2) directly comparing the output of the language model to the ground-truth logical expression. The combinations of language model pretraining and feedback type form the four experiments presented.

During training the environment provides noisy feedback from randomization of object and agent positions and imperfections in the trained compositional policies. Further, each of the environments has one or four distractor objects sampled uniformly at random from the 18 object types. These objects may have the same type attributes as the target object, in which case the mission command changes from using the definite to the indefinite article (e.g. "pick up a red ball," instead of "pick up the red ball").

During inference the mission statement for the current task is translated to a Boolean expression, which is passed to a Boolean expression parser to determine syntactic correctness. If the expression is not syntactically valid, the agent receives a reward of -1.0. If the expression is valid, the corresponding compositional policy is instantiated and executed in the environment 50 times.

The agent receives the mean reward from these 50 roll-outs. Each episode receives a reward of $+1.0$ for successfully picking up the target object, and -2.0 for failing to pickup the target object within 50 time-steps. The 50 roll-out parameter was chosen by imperially establishing that 50 offers a robust estimation of the mean reward for the instantiated policy.

The rewards of $+1.0$ and -2.0 were determined empirically to incentivize the production of optimal Boolean expressions. Without asymmetrically discouragement for picking up the wrong objects, the agent can learn to simply rely on color or object type (rather than both) to act in the environment and attain reward. This behavior represents a local minimum, where the agent will attain reward in cases where color or object type distinguishes the correct object from a distractor object. By asymmetrically discouraging failure, the agent is incentivized to utilize both the color and object type information to execute more precise policies. The reward scale and sign determines whether the output Boolean expressions are made more or less likely by the cross-entropy loss.

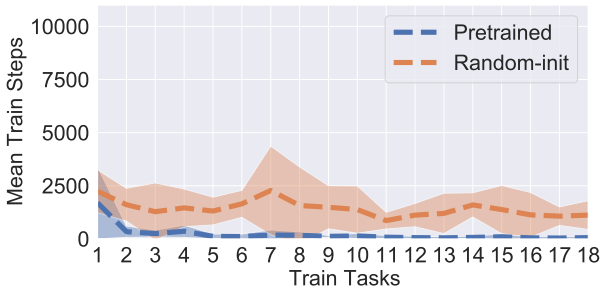
Additionally, we evaluate the effect of environment noise on the translation model's learning. The translation model is separately trained using feedback from logically comparing sampled Boolean expressions to the known true Boolean expressions for those tasks. In this setting, the translation model receives a reward of $+1.0$ for outputting equivalent Boolean expressions and -1.0 for non-equivalent expressions. This removes sources of noise in training the language model: the environmental randomization, distractor objects, and noise from imperfect policies. However, it differs from purely supervised learning in that learning only occurs on samples from the translation model, produced through temperature sampling.

Figure 3 depicts the effects of pretraining versus randomly initializing the translation model, when training with feedback from the environment, and with feedback from the equivalence of output logical expressions. The results indicate that whether acting in the real environment or with "perfect" feedback based on logical equivalence, using the pretrained model vastly outperforms the randomly-initialized translation model. While both models see a decrease in the mean train steps across the randomly shuffled 18 tasks, the number of samples required by the pretrained model drops precipitously after learning the first task. Further, in both the pretrained and randomly-initialized cases, learning in the environment is detrimental to model performance, with both the mean number of training steps, and the standard deviation higher for the agent when learning from environmental feedback. The greater number of training steps demonstrates the negative impact of the distractor objects and the agent's imperfect policies on translation model learning. In Figure 3e the addition of more distractor objects initially requires more training steps on average to learn new tasks, but with substantially higher variance. However, as more tasks are learned, the pretrained model outperforms the randomly initialized model (as in the single distractor setup). We speculate that this is due to the higher variance in the rewards when using more distractor objects.

Baseline Comparison

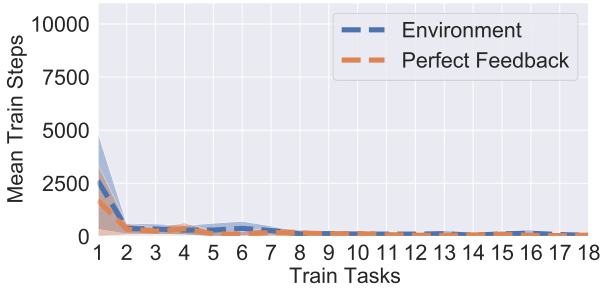
Figure 3f compares the number of training steps needed by the BabyAI baseline model to our compositional model. As with the other experiments, results are reported over 10 trials, where the agent learns to solve each task in the task set sequentially. The task order is shuffled between trials. In this experiment the number of training steps is capped at $20,000$ and a single distractor object is present in the environment. Initially, the baseline model succeeds in learning the first several tasks. However, this model eventually begins to overfit, and reaches the training step limit for the remaining tasks in the set. Despite a much larger parameter count, neither of the compositional models overfit, and the compositional model with language model pretraining needs close to zero additional samples to learn the later tasks.

(a) With feedback from evaluating the compositional policies in the environment with one distractor, compares the effect of pretraining on the number of training steps needed to learn each translation from each mission to each Boolean expression.



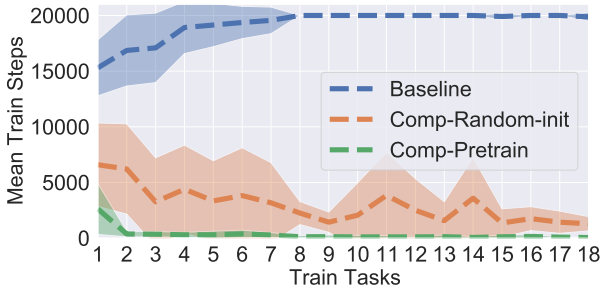
(b) With "perfect" feedback based only on the output Boolean expression, compares the effect of pretraining on the number of training steps needed to learn each translation from each mission to each Boolean expression.

(c) With a randomly-initialized T5 model, compares the learning of the agent from policies evaluated in the environment with one distractor to "perfect" feedback based only on the output Boolean expression.



(d) With a pretrained T5 model, compares the learning of the agent from policies evaluated in the environment with one distractor to "perfect" feedback based only on the output Boolean expression.

(e) With four distractor objects and feedback from evaluating the compositional policies in the environment, compares the effect of pretraining on the number of training steps needed to learn each translation from each mission to each Boolean expression.



(f) Performance of the Baseline model versus Compositional models (both with and without pretraining). All three models are trained using feedback from the environment with a single distractor object. While the baseline model requires substantially more training steps to learn each task, it does not poses explicit compositional structure and does not benefit from pretrained value functions or language representations.

Figure 3: Number of training steps required by various agents to solve each task in a random sequence of tasks. The translation model used is the T5-small model with and without pretraining. Tasks are learned in series, with the same model used across tasks. Task order is randomized across trials. The shaded regions represent the standard deviations over 10 runs.

Difficulty of Translation Tasks

In this experiment, we fine-tune the pretrained translation model using reinforcement learning individually on each of the 18 tasks to compare the relative difficulty of the underlying translations. Unlike in the serial task learning experiment, the translation model learns each task individually with no transfer between tasks. The mean train steps and standard deviations are plotted for 10 trials for each task. The purpose is to determine if learning any of the translations for the tasks are significantly more challenging to learn than the others, which would lead to differential performance when learning certain sequences of tasks.

Figure 4 shows the mean train steps needed to learn each

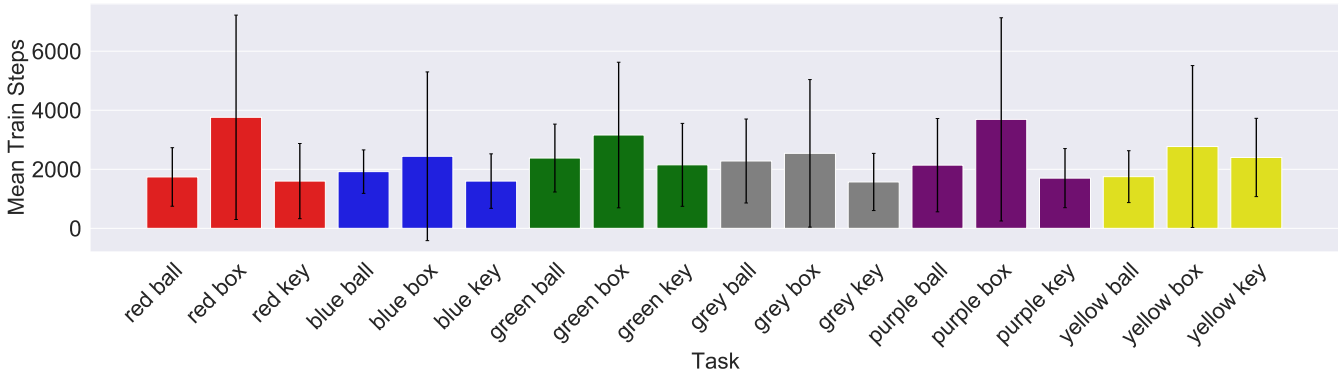


Figure 4: The mean training steps needed to learn the translation from each mission to each Boolean expression for each of the 18 potential tasks, when the translation model has perfect feedback from the environment. Rewards are $+1.0$ for equivalent output Boolean expressions and -1.0 for incorrect expressions. Means and Standard Deviations computed over 10 trials.

translation task based on the logical equivalence of the output expression to the ground-truth expression. The figure shows a similar range of difficulty in translating from each mission statement to each logical expression, indicating that no tasks are overwhelmingly more difficult than the others. However, there are differences in the translation difficulty between certain tasks. Translations for picking up "box" objects consistently require more training samples to learn. The unequal difficulty could be due to differences between the pretrained features for box objects.

Related Work

Instruction following by an artificial agent combines reasoning about an agent's current state, the language command given, and the best sequence of actions an agent can take to solve the given task, making it a challenging problem to solve. Previous approaches have attempted to solve this problem using a single neural network architecture trained using reinforcement learning (Anderson et al. 2018; Blukis et al. 2019; Chaplot et al. 2018). Another approach to solve this problem is to translate language into a sequence of symbols that can then be given as input to a planner (Gopalan et al. 2018). It is possible to learn these symbols directly from data (Gopalan et al. 2020), and compose these symbols using semantic parsing (Dzifcak et al. 2009; Williams et al. 2018) to create novel task specifications that can then be planned over.

Previous work has also attempted to learn compositional linguistic representations to solve instruction following tasks (Kuo, Katz, and Barbu 2021). We, however, do not attempt to train and use compositional language representations (Andreas 2019), and instead use the power of a large language model (Raffel et al. 2020) that has a richer representation as it is created using more data. In this work, we translate language to a set of tokens that represent compositional value functions. To the best of our knowledge, our approach is novel, and maps language to a representation that is inherently compositional allowing us to solve novel tasks with few samples.

Composing value functions was first demonstrated using the linearly-solvable MDP framework by Todorov (2007), in which value functions could be composed in an operator similar to disjunction to solve tasks (Todorov 2009). This idea was extended to compose skills that achieve zero-shot disjunction (Van Niekerk et al. 2019) and approximate conjunction (Haarnoja et al. 2018; Van Niekerk et al. 2019; Hunt et al. 2019). More recently, Nangue Tasse, James, and Rosman (2020) show that zero-shot optimal composition can be achieved for all three logical operators—disjunction, conjunction, and negation—in the stochastic shortest path setting. Our approach allows versatility in the type of expressions, and thereby the type of goal-based instructions that can be given to our agents.

Conclusion

In this work, we proposed an approach for instruction following that leverages the compositional representations present in both the Boolean Task algebra value functions and in large, pretrained language models. Since regular value functions cannot in general be optimally combined to produce desired behaviours (Todorov 2009; Van Niekerk et al. 2019), we leveraged a recently introduced form of goal-oriented value functions that admit composability. By ensuring that both the language and control aspects of the agent are compositional, we demonstrated that an agent can use its existing knowledge to quickly solve new tasks using very few samples. Such sample efficiency is critical in developing long-lived agents that are required to learn and act in the real world. In future work, we would like to create a fully differentiable model that learns to create a space of compositional goals and to map language to the space of Boolean algebra over the learned compostional goals.

Acknowledgements

The authors acknowledge the Centre for High Performance Computing (CHPC) and the Mathematical Sciences Support Unit at the University of the Witwatersrand for providing computational resources for this work.

References

Ackley, D. H.; Hinton, G. E.; and Sejnowski, T. J. 1985. A learning algorithm for Boltzmann machines. *Cognitive science* 9(1): 147–169.

Anderson, P.; Wu, Q.; Teney, D.; Bruce, J.; Johnson, M.; Sünderhauf, N.; Reid, I.; Gould, S.; and Van Den Hengel, A. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3674–3683.

Andreas, J. 2019. Good-enough compositional data augmentation. *arXiv preprint arXiv:1904.09545* .

Bertsekas, D.; and Tsitsiklis, J. 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research* 16(3): 580–595.

Blukis, V.; Terme, Y.; Niklasson, E.; Knepper, R. A.; and Artzi, Y. 2019. Learning to map natural language instructions to physical quadcopter control using simulated flight. *arXiv preprint arXiv:1910.09664* .

Chaplot, D. S.; Sathyendra, K. M.; Pasumarthi, R. K.; Rajagopal, D.; and Salakhutdinov, R. 2018. Gated-attention architectures for task-oriented language grounding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Chevalier-Boisvert, M.; Bahdanau, D.; Lahlou, S.; Willems, L.; Saharia, C.; Nguyen, T. H.; and Bengio, Y. 2019. BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. In *International Conference on Learning Representations*.

Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Dzifcak, J.; Scheutz, M.; Baral, C.; and Schermerhorn, P. 2009. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *2009 IEEE International Conference on Robotics and Automation*, 4163–4168. IEEE.

Gopalan, N.; Arumugam, D.; Wong, L. L.; and Tellex, S. 2018. Sequence-to-Sequence Language Grounding of Non-Markovian Task Specifications. *Robotics: Science and Systems XIV* .

Gopalan, N.; Rosen, E.; Konidaris, G.; and Tellex, S. 2020. Simultaneously learning transferable symbols and language groundings from perceptual data for instruction following. *Robotics: Science and Systems XVI* .

Haarnoja, T.; Pong, V.; Zhou, A.; Dalal, M.; Abbeel, P.; and Levine, S. 2018. Composable Deep Reinforcement Learning for Robotic Manipulation. In *2018 IEEE International Conference on Robotics and Automation*, 6244–6251.

Hunt, J.; Barreto, A.; Lillicrap, T.; and Heess, N. 2019. Composing Entropic Policies using Divergence Correction. In *International Conference on Machine Learning*, 2911–2920.

Jaksch, T.; Ortner, R.; and Auer, P. 2010. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research* 11(Apr): 1563–1600.

James, H.; and Collins, E. 2006. An analysis of transient Markov decision processes. *Journal of applied probability* 43(3): 603–621.

Kuo, Y.-L.; Katz, B.; and Barbu, A. 2021. Compositional RL Agents That Follow Language Commands in Temporal Logic. *Frontiers in robotics and AI* 8: 689550.

Lake, B.; and Baroni, M. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, 2873–2882. PMLR.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529.

Nangue Tasse, G.; James, S.; and Rosman, B. 2020. A Boolean Task Algebra for Reinforcement Learning. *Advances in Neural Information Processing Systems* 33.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* .

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving language understanding by generative pre-training .

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1(8): 9.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research 21* .

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units.

Szabó, Z. G. 2020. Compositionality. In Zalta, E. N., ed., *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2020 edition.

Tambwekar, P.; Silva, A.; Gopalan, N.; and Gombolay, M. 2021. Interpretable Policy Specification and Synthesis through Natural Language and RL. *arXiv preprint arXiv:2101.07140* .

Todorov, E. 2007. Linearly-solvable Markov decision problems. In *Advances in Neural Information Processing Systems*, 1369–1376.

Todorov, E. 2009. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, 1856–1864.

Van Niekerk, B.; James, S.; Earle, A.; and Rosman, B. 2019. Composing Value Functions in Reinforcement Learning. In *International Conference on Machine Learning*, 6401–6409.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Williams, E. C.; Gopalan, N.; Rhee, M.; and Tellex, S. 2018. Learning to parse natural language to grounded reward functions with weak supervision. In *IEEE International Conference on Robotics and Automation (ICRA)*, 4430–4436.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Online: Association for Computational Linguistics.

Appendix A

Task primitive	Success rate
pickup_ball	0.997 ± 0.004
pickup_box	0.996 ± 0.006
pickup_key	1.000 ± 0.000
pickup_red	0.996 ± 0.005
pickup_blue	0.999 ± 0.003
pickup_green	1.000 ± 0.000
pickup_grey	0.996 ± 0.005
pickup_purple	0.996 ± 0.005
pickup_yellow	0.995 ± 0.008

Table 1: The mean success rate of the individual pretrained compositional value functions for each task primitive over 100 episodes. The standard deviations are over 10 runs.

T5-small model parameters	
Embedding dimension	512
Fully-connected dimension	2048
Attention-heads	8
Encoder, Decoder Layers	6

Table 2: The parameters of the T5-small model used in our experiments. To train the model we use the AdamW optimizer (Loshchilov and Hutter 2019) and a learning rate of 1e-4.

Appendix B

The DQNs used to learn the compositional value functions have the following architecture, with the CNN part being identical to that used by Mnih et al. (2015):

1. Three convolutional layers:
   (a) Layer 1 has 3 input channels, 32 output channels, a kernel size of 8 and a stride of 4.
   (b) Layer 2 has 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2.
   (c) Layer 3 has 64 input channels, 64 output channels, a kernel size of 3 and a stride of 1.
2. Two fully-connected linear layers:
   (a) Layer 1 has input size 3136 and output size 512 and uses a ReLU activation function.
   (b) Layer 2 has input size 512 and output size 7 with no activation function.

We used the ADAM optimiser with batch size 256 and a learning rate of 10^{-3}. The target Q-network was updated every 1000 steps, and we used ϵ-greedy exploration, annealing ϵ from 1.0 to 0.1 over 1000000 timesteps.