University of Amsterdam

MSc Artificial Intelligence
Master Thesis

Mastering emergent language: learning to guide in simulated navigation

---

by
Mathijs Mul
6267939

July 5, 2019

36 credits
December 2018 - June 2019

Supervisor:
Dr E Bruni

Assessor:
Dr L Beinborn

Abstract

To cooperate with humans effectively, virtual agents need to be able to understand and execute language instructions. A typical setup to achieve this is with a scripted teacher that guides a virtual agent using language instructions. However, such a setup has clear limitations in scalability and, more importantly, it is not interactive. Here, we introduce an autonomous agent that uses discrete communication to dynamically guide other agents to navigate and act in a simulated environment. The developed communication protocol is trainable, emergent and requires no additional supervision. The emergent language speeds up learning of new agents, it generalizes across incrementally more difficult tasks and, contrary to most other emergent languages, it is highly interpretable. We demonstrate how the emitted messages correlate with particular actions and observations, and how new agents become less dependent on this guidance as training progresses. By exploiting the correlations identified in our analysis, we manage to successfully address the agents in their own language.

Acknowledgements

My greatest gratitude goes to Elia Bruni, my daily supervisor - which really meant *daily*, because you were always available to answer questions, discuss ideas and suggest papers. Your continuous guidance significantly speeded up my learning process, and made navigating towards this thesis a much easier task. I also want to thank Diane Bouchacourt of Facebook AI Research for the valuable discussions and feedback. Without your help I would have been a far less efficient agent. I am grateful to Dieuwke Hupkes for proofreading a paper based on this thesis, and to Raquel Fernández Rovira for her comments on early stages of the research. The other AI Master students in the i-machine-think research group I want to thank for the interesting discussions and shared insights. And of course I want to mention my family and friends, who managed to support me while also calling me crazy for writing a second Master thesis. Without you I might have started working on a third one.

Contents

Chapter 1

Introduction

```
w0w0 w1w0 w2w0 w0w0 w2w0 w1w1 w1w0 w1w0
w2w2 w0w2 w1w0 w2w0 w0w0 w2w0 w2w2 w0w2
```
— An unnamed agent trained for this thesis

Developing intelligent agents that use language to communicate in a cooperative way, with humans or with each other, is a longstanding goal of artificial intelligence (Wooldridge, 2009; Mikolov et al., 2016). Fully supervised approaches are usually considered too static to achieve this goal, especially if agents are also supposed to learn the interactive aspects of communication. For this reason, current research on agent communication tends to focus on processes where language can naturally emerge (Lazaridou et al., 2017; Mordatch and Abbeel, 2017; Havrylov and Titov, 2017).

Approaching communication as an emergent phenomenon eliminates the need to provide explicit supervision on agents' language comprehension and generation. This gives them more freedom to develop a protocol that is dynamic and flexible enough to cover as many use cases as possible. A disadvantage of emergent communication approaches, however, is that the resulting language can be hard to interpret. If agents are given carte blanche to evolve their own system, how do we make sure that we still understand what they mean at the end of the day? An artificial model could choose to ground expressions in all kinds of ways, many of which might seem unintuitive to humans.

In this thesis, we introduce an autonomous agent (the Guide) that uses discrete communication to interactively assist another agent (the Learner) in navigational and operational tasks. In contrast with previous work where a guiding agent's language is hardcoded (Co-Reyes et al., 2018), the communication protocol used by our Guide is fully emergent and does not require additional supervision to be trained. Nevertheless, the Guide develops a language that preserves the desired properties of

a manually programmed module: it is a language that generalizes compositionally across incrementally more difficult problems, and that is highly interpretable.

We show that the Guide, thanks to the developed message channel, strongly speeds up the learning process of new agents. This is not only the case when applied within the same challenge, but also when confronted with a curriculum of different problems. Additionally, we find that optimizing a Guide on an aggregate of related tasks does not affect performance when assisting another agent at one of the subtasks. This suggests that the developed communication scheme captures basic competencies that generalize across problems.

By analyzing the emergent communication protocols, we demonstrate strong correlations between messages and actions, and between messages and salient properties of the environment. Quantifying the causal influence of communication shows how Learners that are assisted by a Guide gradually become less dependent on the message channel. Moreover, we show how to interpret the ungrounded agent language, and even learn to 'speak' it by letting an agent follow pre-set trajectories described by discrete messages.

With this work we hope to contribute to the research on emergent language by presenting an agent that uses an autonomously developed, discrete messaging system to speed up the learning of new agents in a collection of complex sequential decision-making problems. In particular, we wish to prove that such a messaging system can be sufficiently general to apply across a range of varying tasks. Finally, we aim to offer a hopeful perspective on the interpretability issue of emergent language experiments: we show that an emergent communication system such as the one developed by our agents need not be a cryptic exchange of ungrounded symbols, but that it is possible to analyze it in terms that make sense to humans. As an ultimate test, we sucessfully address the agents in their own language. Maybe this result can even bring human and artificial agents a small step closer to each other.

Chapter 2

Related Work

In this chapter we introduce some of the main themes of the research, and reference publications that are relevant to the project.

2.1 Following language instructions

In this work we consider agents that have to follow operational and navigational instructions in an artificial environment. The instructions are expressed in (a small subset of) natural language, which agents must learn to interpret through feedback on their decisions in practice. This means that instructions are essentially interpreted in terms of the acts to which they give rise. Likewise, when we let agents develop their own communication protocol, their utterances are meaningful insofar as they serve to influence each other's actions and, as a consequence, the state of the environment. This reflects a pragmatic understanding of semantics, as famously advocated in Wittgenstein's *Philosophical Investigations*:

> For a *large* class of cases of the employment of the word 'meaning' - though not for all - this word can be explained in this way: the meaning of a word is its use in the language. (Wittgenstein, 1953, §43)

Wittgenstein's pragmatic position greatly influenced modern linguistics and philosophy of language. It facilitates an understanding of language as a means to not only contain information, but also to produce actions and events, thereby changing reality. Variations of this viewpoint are at the basis of Austin's theory of performative utterances (1962), and work on speech acts by Searle (1969) or Clark (1996).

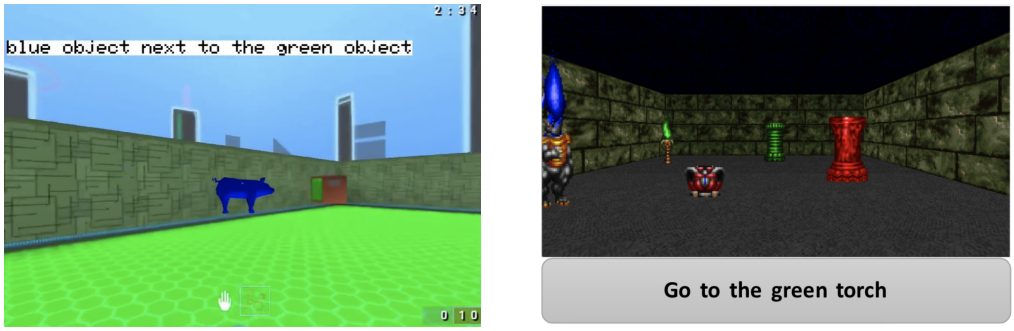
In a sense, research on linguistic instruction following is bound to endorse a pragmatic view on language, because it approaches utterances (instructions as they are expressed) as phenomena that are supposed to cause particular actions (instructions as they are followed).

Throughout the past decades, many studies have been dedicated to this topic. One of the early attempts to design a computer system that understands and executes commands formulated in English was made by Winograd (1972). Winograd

intended his program to interpret language directly, by hard-coding a great number of linguistic and physical regularities. Domain-specific knowledge was added to deal with particular subjects.

More recently, Cangelosi et al. (2006) showed that robots equipped with neural networks are capable of learning action concepts from linguistic instructions. By grounding the names of a few atomic actions, they could be taught to conceptualize and perform more complex operations. Similar results were obtained by Tellex et al. (2011), Chen and Mooney (2011) and Artzi and Zettlemoyer (2013).

In recent years, several artificial environments have been proposed to develop and assess agents' instruction-following capacities. Examples include the 3D-worlds of Hermann et al. (2017), Brodeur et al. (2017), Chaplot et al. (2018) and Wu et al. (2018). In these frameworks, agents are rewarded if they successfully follow a navigational command. Sample observations from two of these environments are shown in Figure 2.1.



(a) (b)

Figure 2.1: Screenshots of the artificial environments with language instructions proposed by (a) Hermann et al. (2017) and (b) Chaplot et al. (2018).

Other recent studies where agents are challenged to execute language commands include those conducted by Mei et al. (2016), Yu et al. (2018) and Bahdanau et al. (2018). Before exposure to instructions, their agents are assumed to have no linguistic knowledge. In contrast, Wang et al. (2016) and Williams et al. (2018) do endow agents with prior awareness of linguistic structures.

In this research we use a synthetic gridworld with natural language instructions called 'BabyAI' (Chevalier-Boisvert et al., 2018). See Section 3.1 for a more detailed description of this environment, as well as the grammar and semantics of the instructions. We adopt BabyAI in this project because it combines the following four appealing properties, which makes it stand out among the other platforms referenced in this section:

1. *partial observability*: agents are only able to perceive a small part of the environment, depending on their location. The fact that they are not omniscient makes the set-up more realistic.

2. *dynamic environment*: agents can alter the state of their surroundings, e.g. by picking up objects and putting them down at a different location. In many sophisticated 2D or 3D worlds this is not possible, and agents are limited to navigation.

3. *well-defined language*: the grammar of the artificial language used to formulate instructions is systematically defined, with semantics for each admissible utterance. Thanks to this property, a large collection of different instructions can be generated.

4. *curriculum learning*: BabyAI consists of a hierarchy of levels that gradually increase in complexity. This offers an interesting testbed for the transfer of certain skills between different situations.

Co-Reyes et al. (2018) extended the BabyAI baseline model with a correction module, which provides additional linguistic feedback to an agent. By doing so, they increase sample efficiency. Their correction module, however, is hard-coded. We introduce a comparable mechanism that is entirely trained.

2.2 Imitation learning

Most of the models in our research are trained using imitation learning. Here, the objective of an agent is to simulate the behavior of another agent with more experience or knowledge of a particular task. It is usually applied in the context of sequential decision making problems, where agents have to come up with a series of decisions based on a policy. In such cases, actions influence future observations, which violates the i.i.d. assumption of many statistical learning methods. Because imitation learning relies on exemplars provided by an expert agent or an oracle, it is also known as 'learning by demonstration'.

A range of imitation learning algorithms has been proposed in the literature. We use behavioral cloning (Pomerleau, 1991), which treats expert trajectories as series of states labelled with actions that a new agent can learn to predict. This can be done in a regular, supervised setting, by treating each labelled state as input and the assigned expert action as corresponding output. The principle of behavioral cloning is illustrated in Figure 2.2 by the green dots and arrows. The green dots denote the states that are visited by the expert agent, and the green arrows the actions taken by the expert at each of them. When applying behavioral cloning, a learning agent is presented with these expert states and challenged to predict the subsequent expert actions as if they were ordinary labels.

Behavioral cloning is an effective imitation learning method in cases where mimicking an expert policy suffices to solve a task. In other situations it may be necessary for a learner to not just copy the behavior manifested in the demonstrations, but to also to deal with observations that are not included in the expert data. This is especially important in problems where a learner easily risks deviating from the

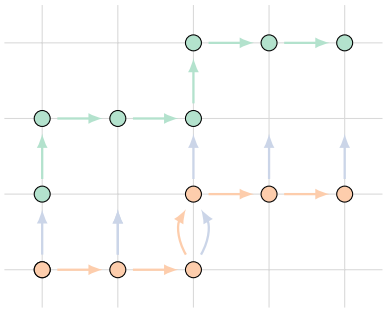


Figure 2.2: Imitation learning trajectories. The green dots and arrows represent states visited and actions performed by an expert agent. The orange dots and arrows represent states visited and actions performed by a learning agent. The blue arrows represent actions taken by an expert agent in the states visited by a learning agent.

expert's trajectories, so that configurations are encountered that the demonstrations do not cover.

Methods that address this issue include the algorithms Searn (Daumé et al., 2009) and SMILe (Ross and Bagnell, 2010), which extend supervised imitation with an iterative learning scheme. DAgger (Ross et al., 2011) relies on the aggregation of additional expert data during training. The DAgger algorithm first lets a learning agent complete a trajectory on her own. In Figure 2.2, the orange dots and arrows represent the states visited and the actions taken by the learner at this stage. Next, the expert agent is presented with the states visited by the learner, and queried about the optimal action to take in each of them. In Figure 2.2, the blue arrows departing from the orange dots represent these actions. When the learner is trained, the states encountered on her own trajectory (the orange dots) serve as inputs, while the actions recommended by the expert at each of them (the blue arrows) serve as targets. This is how DAgger deals with agents that drift away from the expert path.

An alternative form of imitation learning is inverse reinforcement learning. In standard reinforcement learning, an agent has to optimize a policy with respect to a pre-established reward function that assesses the merits of (sequences of) actions (see Appendix A.2.2 for an introduction). Especially in real-world problems, designing a suitable reward function can be far from trivial and require a lot of manual engineering. Inverse reinforcement learning sets out to *learn* reward functions from expert demonstrations. Following optimization of the reward function, it can be used to develop a policy using ordinary reinforcement learning. Inverse reinforcement learning was advocated by Russell (1998), Ng and Russell (2000) and Ziebart et al. (2008). A variant using generative adversarial models, GAIL, was introduced by Ho and Ermon (2016).

Forms of imitation learning were successfully applied in the context of following language instructions by e.g. Hemachandra et al. (2015), Bahdanau et al. (2018) and Chevalier-Boisvert et al. (2018).

2.3 Emergent communication

In this project, we introduce a neural agent that uses emergent communication methods to assist another agent in solving a task. We impose no direct supervision on the emitted language, so that speakers are free to develop their own semantics.

The emergence and evolution of language has received much attention in linguistics, game theory and cognitive science (Skyrms, 2010; Wagner et al., 2003; Crawford, 1998). In AI, communication between artificial agents in order to aid cooperation is generally considered a major goal (Wooldridge, 2009; Mikolov et al., 2016). Many researchers have claimed that supervised approaches to teach agents how to communicate will ultimately fail due to the static nature of this method, and that processes where language can naturally emerge offer a more promising approach, e.g. Lazaridou et al. (2017):

> [T]raining on 'canned' conversations does not allow learners to experience the interactive aspects of communication. Supervised approaches, which focus on the structure of language, are an excellent way to learn general statistical associations between sequences of symbols. However, they do not capture the functional aspects of communication, i.e., that humans use words to coordinate with others and make things happen (Lazaridou et al., 2017, 1)

In the current research, we study the emergence of communication through discrete language in an artificial, multi-agent set-up. Related experiments were conducted by Jorge et al. (2016) and Sukhbaatar et al. (2016), who showed that agents can autonomously evolve a communication protocol that helps them to play a game. Lazaridou et al. (2017) obtained similar results, but also investigated how to change the environment in order to interpret the emergent language more easily. Mordatch and Abbeel (2017) showed that multi-agent communities can give rise to a grounded compositional language that helps speakers achieve their goals.

Inspired by Lazaridou et al. (2017), Havrylov and Titov (2017) developed a scenario where agents learn to assist each other by sending variable-length sequences of discrete symbols. Because this set-up is closest to the one that we will develop in Section 3.2, Figure 2.3 provides a schematic visualization. Havrylov and Titov use a referential game (Lewis, 1969), where one agent (the Sender) has to describe the picture that another agent (the Receiver) must select. In Figure 2.3, the target image contains a pizza, which the Sender explains with the message $w_1 w_2 w_3 w_4$. The tokens in this message are sampled from the categorical distribution generated by projecting the hidden states of an LSTM decoder into the vocabulary space. This message is passed to an LSTM encoder that is part of the Receiver. At the same time, the Receiver sees the target image and two distractors. Communication success depends on the Receiver's ability to single out the target image, using the information stored in the Sender's message. There is no supervision on the messages that are being emitted, which can therefore be regarded as discrete latent

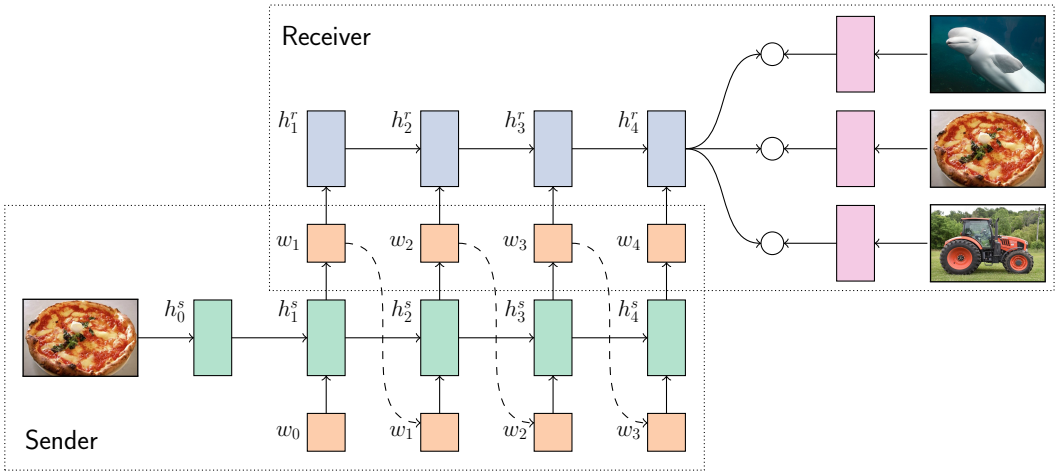


Figure 2.3: Sender-receiver architecture of Havrylov and Titov (2017). Variables h_i^s represent hidden states of the Sender's LSTM decoder, h_i^r hidden states of the Receiver's LSTM encoder and w_i word embeddings of selected tokens.

variables whose behavior is developed autonomously by the agents. In order to ensure end-to-end differentiability of the set-up when generating discrete messages, several sampling methods are considered, including straight-through Gumbel softmax estimation (Bengio et al. (2013); Jang et al. (2017); see Section 3.2.2). Havrylov and Titov report high communication success rates in this game.

A problem with languages that naturally emerge among agents is that they can be hard to interpret. This is because developed notions need not be grounded in the same way as natural language concepts. For this reason, emergent communication is often measured in relation to task success. Lowe et al. (2019) critically discuss the most prevalent such quantification methods.

2.4 Curriculum learning

Humans can learn better and faster when tasks are not presented in a random order, but according to increasing difficulty (Bengio et al., 2009). Applying this concept in machine learning by confronting a model with a hierarchy of gradually more complex assignments is known as 'curriculum learning'. The BabyAI framework contains a number of levels that are defined in terms of the competencies required to solve them. Because many such competencies transfer across levels, we will also use BabyAI to assess the impact of curriculum learning on our model's development.

It has often been shown that curriculum learning can improve the performance of machine learning models (Bengio et al., 2009; Kumar et al., 2010; Khan et al., 2011; Zaremba and Sutskever, 2014; Graves et al., 2016). More generally, it can be seen as a form of transfer learning, which considers all cases where acquired skills or knowledge may be shared across tasks, whereas curriculum learning specifically

focuses on improving model optimization by increasing convergence rates or sample efficiency.

Indeed, faster convergence is often considered one of the main benefits of curriculum learning. Bengio et al. formulate this advantage as follows:

> (...) faster training in the online setting (i.e. faster both from an optimization and statistical point of view) because the learner wastes less time with noisy or harder to predict examples (when it is not ready to incorporate them (Bengio et al., 2009, 7)

More efficient optimization in a complex setting could be explained by improved generalizing capacities of a learner that was pretrained at a simpler task, because she is able to recognize the transferability of certain core skills across problems. This effect was observed by e.g. Krueger and Dayan (2009).

Chevalier-Boisvert et al. (2018) perform curriculum learning experiments in the BabyAI game and report improved sample efficiency of models pretrained at relatively simple base levels. They quantify the effect in terms of the number of demonstrations required to solve a BabyAI level by means of imitation learning, with and without pretraining at another level requiring some of the same competencies. It turns out that this number can be significantly lower for the pretrained models. This shows that the considered models may benefit from an incremental learning scheme, as is the case for humans.

Chapter 3

Approach

This chapter is dedicated to the approach underlying the coming experiments. We first provide a more detailed explanation of the BabyAI game, then describe the model architectures we will be using in the experiments and finally specify the training routine.

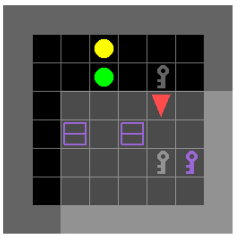
3.1 BabyAI game

All our experiments are conducted using the BabyAI framework, introduced by Chevalier-Boisvert et al. (2018).[1] The platform lets virtual agents play games at a number of different levels. Each of these levels are combinations of a partially observable two-dimensional gridworld (MiniGrid), and a mission presented as a linguistic instruction string. The challenges amount to a range of operational and navigational tasks that vary per level. See Figure 3.1 for three sample frames.

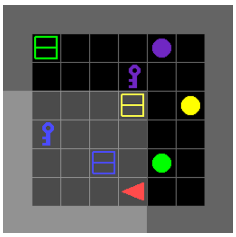
Observation and action space Levels are either situated in a single room of 6×6 tiles, or in a 3×3 maze of such rooms that are connected by doors at random positions in the walls between them. The environment contains several kinds of objects (walls, doors, keys, balls and boxes) that come in different colors (red, green, blue, purple, yellow and grey). At each time step t, an agent receives as input the instruction string i_t (constant during a game), and the tensor o_t encoding the currently observable part of the surroundings. o_t is $7 \times 7 \times 3$-dimensional, because the agent has an egocentric, local vision field of 7×7 tiles. Each of these tiles is described in terms of three variables: the type of object on it, its color and a flag showing if doors are open or locked.

Given the inputs i_t and o_t, the agent has to decide on an action a_t, which can be one of the following seven options: turning left, turning right, moving forward, picking something up, dropping something, toggling (opening a door) and finishing

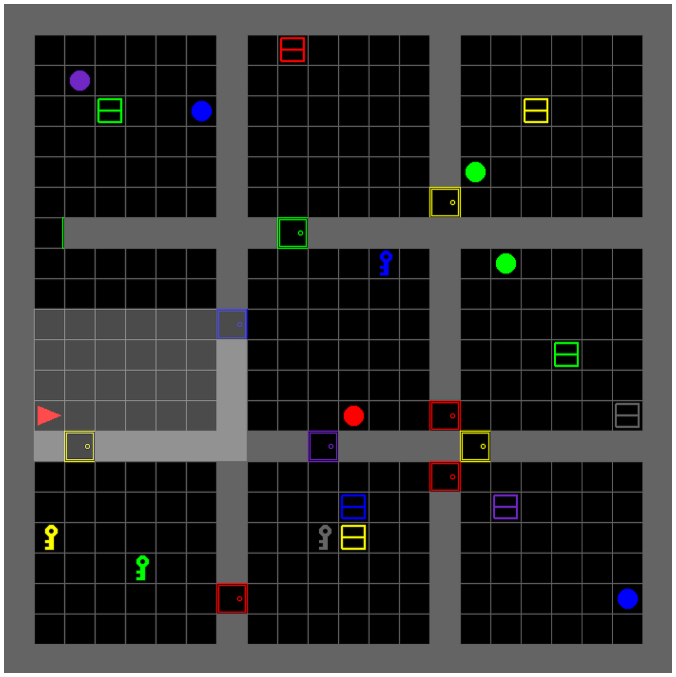
[1] Code used by Chevalier-Boisvert et al. (2018) is available on `https://github.com/mila-udem/babyai`.



(a) level: GoToLocal, mission: 'go to the purple key'



(b) level: PutNextLocal, mission: 'put the green box next to the green ball'



(c) level: GoTo, mission: 'go to a green ball'

Figure 3.1: Screenshots of indicated BabyAI levels and missions. The red arrow represents the agent and the shaded area the currently observable part of the environment.

a task. By turning around, the agent changes her direction, which can be facing up, down, left or right.

Instruction language Each BabyAI episode comes with a mission, which is formulated as a discrete instruction string. The instructions belong to a (small) subset of English. A restricted vocabulary and grammar are used, which is why it may be more appropriate to regard the BabyAI language as an artificial one. That being said, all mission statements are readily intelligible for any English speaker. This is an advantage because it makes the set-up resemble a real-life situation where humans give natural language instructions to artificial agents.

Figure 3.2 contains the Backus-Naur Form grammar for the BabyAI language, as specified by Chevalier-Boisvert et al. Some examples of well-formed instruction strings according to this grammar are shown below the sample frames in Figure 3.1. Approximately 2.48×10^{19} unique instructions can be generated using the BabyAI grammar.

Included in the BabyAI framework is a verification module that checks whether an agent has completed the command specified by an instruction string. Sometimes, an instruction entails several commands. This is clearly the case when subgoals are

⟨Sent⟩ ⊨ ⟨Sent1⟩ | ⟨Sent1⟩, then ⟨Sent1⟩ | ⟨Sent1⟩ after you ⟨Sent1⟩
⟨Sent1⟩ ⊨ ⟨Clause⟩ | ⟨Clause⟩ and ⟨Clause⟩
⟨Clause⟩ ⊨ go to ⟨Descr⟩ | pick up ⟨DescrNotDoor⟩ | open ⟨DescrDoor⟩
| put ⟨DescrNotDoor⟩ next to ⟨Descr⟩
⟨DescrDoor⟩ ⊨ ⟨Article⟩⟨Color⟩ door ⟨LocSpec⟩
⟨DescrBall⟩ ⊨ ⟨Article⟩⟨Color⟩ ball ⟨LocSpec⟩
⟨DescrBox⟩ ⊨ ⟨Article⟩⟨Color⟩ box ⟨LocSpec⟩
⟨DescrKey⟩ ⊨ ⟨Article⟩⟨Color⟩ key ⟨LocSpec⟩
⟨Descr⟩ ⊨ ⟨DescrDoor⟩ | ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨DescrNotDoor⟩ ⊨ ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨LocSpec⟩ ⊨ ϵ | on your left | on your right | in front of you | behind you
⟨Color⟩ ⊨ ϵ | red | green | blue | purple | yellow | grey
⟨Article⟩ ⊨ the | a

Figure 3.2: Backus-Naur Form grammar of the BabyAI instruction language. ϵ denotes the empty string.

combined with conjunction 'and', as well as with conjunction 'then' or 'after', in which cases there is an additional restriction on the action order. In other situations it may be less clear that several subgoals must be fulfilled, e.g. when the agent has to relocate some object to reach the target. If an indefinite article is used when specifying some item, it means that the agent is free to pick any object satifsying this description. Such underspecification makes the task harder but more realistic. In real-world object location tasks agents are also likely to encounter several items matching a description, which forces them to develop additional criteria for selecting a particular one based on considerations such as physical proximity.

Levels In total, BabyAI has 19 levels with increasing degrees of complexity. In this paper we consider six such levels, the conjunction of which we consider a representative fragment of the game as a whole. They are listed in Table 3.1, together with the core skills required to solve each of them. The respective skills are explained in the following list:

- ROOM: navigating a 6×6 room
- DISTR-BOX: ignoring distracting grey boxes
- DISTR: ignoring distracting objects of any kind
- GOTO: understanding 'go to' instructions (if there is more than one object)

- PUT: understanding 'put' instructions
- PICKUP: understanding 'pick up' instructions
- LOC: understanding location expressed relative to initial agent position
- MAZE: navigating a 3×3 maze of randomly connected 6×6 rooms

	required skills							
	ROOM	DISTR-BOX	DISTR	GOTO	PUT	PICKUP	LOC	MAZE
GoToObj	x							
GoToLocal	x	x	x	x				
GoToObjMaze	x							x
PickupLoc	x	x	x			x	x	
PutNextLocal	x	x	x		x			
GoTo	x	x	x	x				x

Table 3.1: Considered BabyAI levels and skills required to solve them, from Chevalier-Boisvert et al. (2018).

It is important to note that the BabyAI levels are designed in such a way that competencies required to solve some of them are also needed in others. E.g., in order to complete GoToLocal an agent has to know how to navigate a single room with distractors, and this skill is also necessary to address PickupLoc. Thanks to this hierarchy, BabyAI is a suitable framework for studying the effect of curriculum learning: learning by gradually increasing the complexity of a task.

Markov decision process BabyAI is a discrete, finite game, which means that the total number of possible states in each level is finite as well. Also the number of actions that can be taken is finite, as there are only seven of them. Moreover, if the state of the full environment (the entire room or maze) at time t is denoted by s_t, then the next state, s_{t+1}, is determined only by the current state, s_t, and the current decision of the agent, which is the action a_t. Because of these properties, all BabyAI levels with a single acting agent can be framed as Markov decision processes (MDP) (Howard, 1960).[2]

[2] A MDP is defined as a tuple $\langle S, A, P, R \rangle$, where

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix, expressing the probability of moving to state s' from state s after performing action a at time t, for any $s, s' \in S$, $a \in A$ and time t
- R is a reward function, expressing the reward for moving to state s' from state s by action a, for any $s, s' \in S$ and $a \in A$

More precisely, because the agent has a local field of vision and therefore only possesses incomplete information about the state of the environment, we are dealing here with a partially observable Markov decision process (Astrom, 1965), which is often abbreviated as 'POMDP'. POMDPs are popular modelling tools in real-life sequential decision-making problems such as robotic navigation or object relocation (Pajarinen and Kyrki, 2017; Chen et al., 2016).

3.2 Models

In our experiments we consider two kinds of agents, that come with their own end-to-end architectures: the Learner and the Guide. The fully continuous Learner can approach a task alone, or be assisted by a Guide. A Guide is an agent that not only learns to solve a task, but also to emit discrete messages in the process. These messages can be used to help a new Learner master a task more efficiently.

All of our models are neural. We assume that most readers are familiar with neural networks, but for those who are not we refer to the introduction in Appendix A. The described set-ups are implemented in PyTorch (Paszke et al., 2017).[3]

3.2.1 The Learner

The Learner architecture follows the set-up used by Chevalier-Boisvert et al. (2018). As input at time t, the model receives the tuple $\langle i_t, o_t \rangle$, where i_t represents the instruction string and o_t the observation tensor. The tokens contained in i_t are embedded, the result of which is presented to a GRU (Cho et al., 2014), which encodes the entire sequence into a single vector. The observation o_t is processed by a convolutional network (Fukushima, 1980; LeCun et al., 1989). Twice in a row, this network applies a two-dimensional convolution, batch normalization (Ioffe and Szegedy, 2015), rectified linear unit activation and max pooling.

When the instruction and observation are encoded, the results of these operations are processed together by a FiLM module to obtain a joint representation of the command and the environment. FiLM (abbreviation of 'Feature-wise Linear Modulation') was introduced by Perez et al. (2018) and offers a simple yet effective way of integrating vector representations. It does so by computing an affine transformation on some feature, conditioned on another feature or input. Perez et al. show that this method is especially suitable for visual reasoning tasks, where finding an adequate combination of optical and textual modalities is key. This also makes it a sensible option for BabyAI, where agents have to combine visual input with linguistic information.

In our case, the Learner uses a two-layered FiLM module to transform the output of the convolutional network, conditioned on the GRU encoding of the instruction. Let FiLM^l, GRU^l and Conv^l denote the Learner's FiLM module, instruction-

[3] Code available on `https://github.com/MathijsMul/babyai-emergent-guidance`.

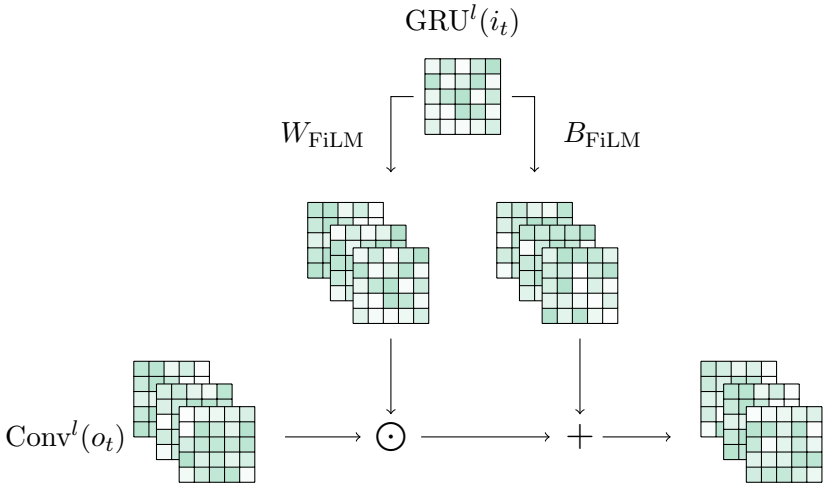


Figure 3.3: FiLM layer in the Learner. $\odot$ denotes entry-wise (Hadamard) product; $+$ entry-wise sum.

encoding GRU and observation-processing convolutional network. Then the FiLM module is visualized in Figure 3.3, and the following computation is performed:

$$\text{FiLM}^l(\text{GRU}^l(i_t), \text{Conv}^l(o_t)) = (W_{\text{FiLM}}\text{GRU}^l(i_t)) \odot \text{Conv}^l(o_t) + B_{\text{FiLM}}\text{GRU}^l(i_t) \tag{3.1}$$

Here, $\odot$ stands for entry-wise product, W_{FiLM} represents the matrix mapping GRU output to FiLM weights, and B_{FiLM} the one mapping GRU output to the FiLM bias.

Next, an LSTM cell (Hochreiter and Schmidhuber, 1997) processes this FiLM representation together with a memory of its own past hidden and cell states. For this reason, it is called the 'memory LSTM'. The FiLM output serves as its input. Its hidden state is initialized by the first half of its memory (used for storing the previous hidden state), and its cell state by the second half (used for storing the previous cell state). The output of the memory LSTM is passed to the policy layer. Let the computation of LSTM output (final hidden state) y be represented in the form of $y = \text{LSTM}(i, (h, c))$, where i is the input, h the initial hidden state and c the initial cell state. Then the Learner's input tensor to the policy module at time t, which we call x_t^l, is:

$$x_t^l = \text{LSTM}^l(\text{FiLM}^l(\text{GRU}^l(i_t), \text{Conv}^l(o_t)), (m_{t-1}^l[:h], m_{t-1}^l[h:])), \tag{3.2}$$

where LSTM^l denotes the Learner's memory LSTM, i_t the instruction and o_t the observation at t, m_{t-1}^l the Learner's memory state at $t-1$ and h the hidden size of the LSTM (half the size of the memory state). The predicted action $\hat{a}_t$ at time t is computed by the two-layered, feed-forward policy module π:

$$\hat{a}_t = \pi(x_t^l) \tag{3.3}$$

$\hat{a}_t$ is established by subsequently applying the softmax and argmax function to the final, seven-dimensional vector returned by the linear policy layers. Its seven entries correspond to the seven possible actions in the BabyAI game.

3.2.2 The Guide

We now present a variation on this model, which we call the 'Guide'. The Guide's architecture is similar to the Learner's, but differs in one crucial aspect: between the memory LSTM and the policy module we introduce an information bottleneck. Its function is to create the communication channel, where the information flowing from the input-processing layers has to be compressed in the form of a sequence of discrete tokens. These messages can later be sent to a Learner.

The Guide's discrete message channel is inspired by information bottleneck theory (Tishby et al., 1999), which studies ways of finding a compact (possibly lossy) compression of a signal X while preserving as much information as possible about another signal Y that depends on X. In our case, X stands for the input received by an agent (observations and instructions) and Y for the output returned by an agent (actions). As Tishby et al. note:

> Obviously lossy compression cannot convey more information than the original data. (...) there is a tradeoff between compressing the representation and preserving meaningful information, and there is no single right solution for the tradeoff. The assignment we are looking for is the one that keeps a fixed amount of meaningful information about the relevant signal Y while minimizing the number of bits from the original signal X (maximizing the compression). In effect we pass the information that X provides about Y through a 'bottleneck' formed by the compact summaries (Tishby et al., 1999, 9)

Likewise, we aim to develop a Guide that manages to summarize the information contained in the input as efficiently as possible. That is: insofar as this information is relevant with respect to the optimal (sequence of) actions. Compression of the representation is enforced by imposing restrictions on the maximum message length. Moreover, the messages can only contain discrete symbols. Given these limitations, the Guide is challenged to produce messages that contain as much useful information as possible.

Generation of the messages takes place in a GRU decoder that receives the Guide's memory LSTM output x_t^g as input. This tensor is computed as:

$$x_t^g = \text{LSTM}^g(\text{FiLM}^g(\text{GRU}^g(i_t), \text{Conv}^g(o_t)), (m_{t-1}^g[:h], m_{t-1}^g[h:])), \tag{3.4}$$

with notation analogous to Equation 3.2 and superscripts g denoting modules of the Guide. From x_t^g the Guide's GRU decoder, Decoder^g, produces a 'guidance' message g_t:

$$g_t = \text{Decoder}^g(x_t^g) \tag{3.5}$$

The Guide's decoder works in a similar fashion as the Sender in the architecture of Havrylov and Titov, which is visualized in Figure 2.3. The initial hidden state h_0 is given by the Guide's memory LSTM output tensor x_t^g. Embedded token w_0 represents the start-of-sequence (SOS), which initializes the message and is always the same. This is the input to the GRU decoder. Upon receiving w_0 and h_0, the decoder returns a new hidden state h_1. This hidden state is projected onto the vocabulary space. The softmax function is then applied to find the predicted distribution over vocabulary items, which is used to determine the next token. This token's embedding, w_1, is used as input at the next time step. This process continues until the maximum sequence length is reached. The guidance message g_t is the concatenation of the emitted tokens.

The most challenging part of this set-up is the selection of tokens from the predicted distribution. This step requires the discretization of continuous information to infer a message consisting of single words. Standard ways to do this are sampling from the distribution over tokens, or simply applying the argmax function to select the most likely one. These operations, however, are non-differentiable, thereby breaking the computation graph. Since we want to use backpropagation to optimize the model parameters, we need an alternative method to deduce the tokens. What we use instead is a straight-through Gumbel softmax estimator (Bengio et al., 2013; Jang et al., 2017). A schematic overview of this estimator is given in Figure 3.4.

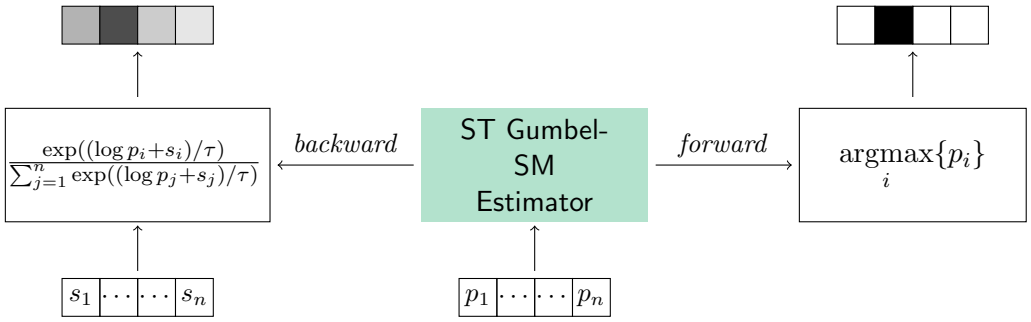


Figure 3.4: Straight-Through Gumbel Softmax Estimator. The probabilities $p_1, \cdots, p_n$ together form the normalized input vector $(p_1 \cdots p_n)$. The vector $(s_1 \cdots s_n)$ represents noise, sampled from the Gumbel distribution by computing $s_i = -\log(-\log(u_i))$, for randomly uniform samples $u_i \sim \mathsf{Unif}(0, 1)$.

Gumbel softmax is the following variation on the regular softmax function:

$$\frac{\exp((\log p_i + s_i)/\tau)}{\sum_{j=1}^{n} \exp((\log p_j + s_j)/\tau)} \tag{3.6}$$

In our case, p_i denote probabilities coming from the predicted distribution over n tokens. The values s_i are Gumbel samples: $s_i = -\log(-\log(u_i))$ for $u_i \sim \mathcal{U}(0, 1)$. τ is a temperature parameter, which we learn with a single-layered network that takes the decoder's hidden state as input. As τ approaches 0, output vectors of the Gumbel softmax function approach one-hot encodings. Therefore, the function can be seen as a continuous relaxation of the argmax operation. Yet, instead of selecting the embedding of the token whose probability is highest, we sum the convex combination of the embeddings of all tokens, using the Gumbel softmax outputs as coefficients. During backpropagation, this ensures differentiability. For this reason, Gumbel softmax sampling is used at each backward pass. On the forward pass, no differentiability is required, so we use standard argmax then. Of course, this is always the case at testing time.

There is no gold standard for the messages produced by the Guide. Indeed, no additional supervision signals whatsoever are used to train the message channel, so that a Guide is free to develop her own protocol and ground the available tokens in the way that has most practical use in the task at hand.

When the Guide has produced the string of symbols g_t, it can be passed to the policy module π. Here the message is used to predict action $\hat{a}_t$ after it has been encoded again by a GRU encoder, c.q. the Guide's own encoder Encoder^g:

$$\hat{a}_t = \pi(\text{Encoder}^g(g_t)) \tag{3.7}$$

3.2.3 Combined set-up

The messages produced by the Guide are discrete latent variables that the agent can use to talk to herself. This was the case in the previous section, where we introduced the Guide as a single agent who is both sender and receiver of the messages. We can regard this as a form of self-talk.

Of course, the Guide can also direct her messages at other agents. In our experiments, we will focus on this situation. If the Guide sends messages to another agent, we regard their emission and interpretation as a simple communication channel that models can optimize in order to improve their own or each other's performance on a task. In particular, we are interested in the effect of the Guide's messages on the learning curve and performance of a new Learner.

By first training a Guide and then pairing the experienced Guide with a new Learner, we investigate if and how agents can learn to make use of the available communication channel. The set-up used to test this is visualized in Figure 3.5. A new Learner is coupled with a pretrained Guide, and receives her discrete guidance at each time step. The Learner is provided with a new GRU encoder to interpret

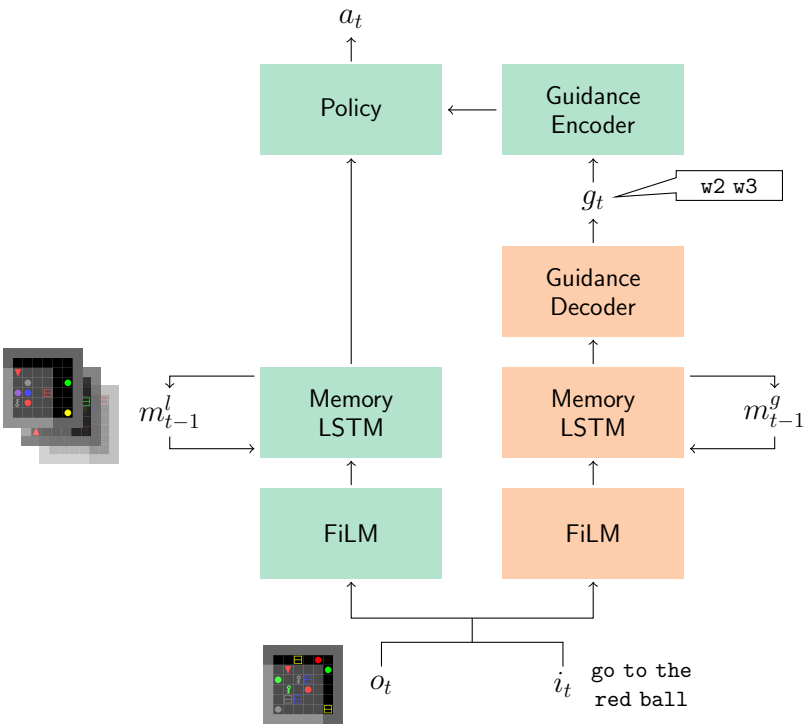


Figure 3.5: Schematic visualization of the combined set-up. Included variables are o_t: state of the (observable part of the) environment, i_t: instruction string, g_t: guidance message, a_t: action, m^l_{t-1}: Learner's memory state, m^g_{t-1}: Guide's memory state. Subscript represents point in time. Green modules are considered part of the Learner; red ones part of the Guide.

the guidance, and uses this information as additional input to her policy module. In this combined set-up, the predicted action $\hat{a}_t$ is computed as:

$$\hat{a}_t = \pi(x^l_t; \text{Encoder}^l(g_t)), \tag{3.8}$$

where the semicolon denotes concatenation and Encoder^l represents the newly initialized GRU encoder that a Learner uses to process the guidance messages. x^l_t and g_t are still as specified in Equations 3.2 and 3.5.

When training the combined set-up, a new Learner is paired with a pretrained Guide, but we do not freeze the parameters of the Guide. This is to say that we continue to optimize the Guide's weights, along with those of the Learner. In this way, a Guide can learn to adapt to a specific Learner, while simultaneously making use of the general knowledge she acquired on her own at the pretraining stage. Also, in the combined set-up, corresponding modules in the Learner and the Guide always have the same dimensionalities.

Considered alternatives Before designing the architecture outlined in this section, we conducted experiments with several other set-ups. Here we summarize three of those, together with the reason why we discarded them:

- The Guide receives the Learner's memory LSTM hidden state *in addition* to the observation and the instruction. We initially regarded this as a form of communication going from the Learner to the Guide. However, if we want to treat the Learner and the Guide as individual agents, it is hardly justifiable that the Guide should have access to an internal state of the Learner. Moreover, communication is not discrete in this case, so we cannot speak of emergent language (at least not from the Learner to the Guide).

- The Guide receives the Learner's memory LSTM hidden state *instead* of the observation and the instruction. In this set-up, the Guide is exclusively conditioned on the Learner. This is somewhat appealing because it forces the Guide to produce messages that are entirely customized to the Learner, so deducing generic hints from the observation and instruction is not an option. As with the previous option, however, it is not realistic that the Guide has access to an internal representation of the Learner. Additionally, the hidden memory LSTM state is so large and specific that optimization of the Guide would depend too much on the particular representations developed by a single Learner.

- The Guide stays the same, but the Learner receives the Guide's messages as part of her regular input (together with the observation and the instruction). The guidance messages receive no 'special treatment' in this case, and are processed just as the other external information that the Learner receives. While this approach may seem very clean, it entails that all of the Learner's modules are conditioned on the guidance messages (because they are propagated through all layers). In the current set-up, some of the Learner's modules are optimized independently of the Guide's messages. We prefer this because it motivates the Learner to also develop a self-standing solution.

3.3 Training

The Learner, the Guide and the combined Learner-Guide set-ups are all end-to-end architectures that receive observations and instructions as input, and whose parameters can be updated by backpropagating a loss function.

In our experiments, we use imitation learning: agents are supposed to internalize the policy demonstrated by an expert agent. This requires that we first have such an expert at our disposal. Chevalier-Boisvert et al. (2018) implemented a heuristic bot that is able to solve all BabyAI levels. We do not use this hard-coded expert, but instead apply proximal policy optimization (PPO) (Schulman et al., 2017) to train what we call RL (Reinforcement Learning) experts. See Appendix A.2.2 for an

introduction to RL in general, and PPO in particular. We use RL because it makes our approach more generalizable, and because Chevalier-Boisvert et al. (2018) made the following observation:

> Interestingly, we found that the demonstrations produced by [an RL expert] agent are easier for the learner to imitate [than those produced by a symbolic bot] (for example, for GoToLocal 70.7K demonstrations were sufficient to imitate the RL expert as opposed to 177K needed to imitate the bot). This can explained by the fact that the RL expert has the same neural network architecture as the learner. (Chevalier-Boisvert et al., 2018, 8)

Apparently, demonstrations generated by a neural model are easier for another neural model to imitate. The RL experts that we train are identical to the Learner models considered in the experiments in all respects except for the applied learning method. Once successfully trained, the RL experts play a large number of sample episodes of the BabyAI levels. Each input is stored and labelled with the action predicted by the expert agent. The resulting data are used to train the other agents in the experiments.

Thus, our experiments follow three consecutive training stages:

1. *Generate demonstrations*

   Train expert agents, with Learner set-up and reinforcement learning, to produce demonstration data.

2. *Train individual agents*

   Train new Learners and new Guides separately on the RL expert data, by imitation learning.

3. *Train combined set-up*

   Train new Learners + pretrained Guides (from stage 2) on the RL expert data, by imitation learning.

Stage 1 only has to be performed once to generate the expert data. It is the most costly step, because RL on complex BabyAI levels requires great amounts of computation. Of course, it is possible to also use RL in stages 2 and 3, but we use imitation learning. This is due partially to computational limitations, and to the fact that we use RL to generate the expert data in stage 1 so that we can perform imitation learning without requiring any manual labelling, human demonstrations or heuristics.

For each of the considered levels, we generate training sets of 1K, 5K, 10K, 25K, 50K, 100K and 200K samples. The validation set always contains 500 episodes. The smallest data set that can be used to train a Learner that converges to a validation success rate of at least 90% is the one that we use in the experiments. Table 3.2 shows how many training samples this set contains for each level. It also gives the mean training episode length, which is the number of frames (or time steps) that a training episode takes on average.

level	size training set	mean training episode length
GoToObj	1K	5.1
GoToLocal	50K	5.8
GoToObjMaze	25K	75.4
PickupLoc	100K	8.9
PutNextLocal	100K	15.5
GoTo	100K	57.2

Table 3.2: Sizes of the training sets and average number of frames per episode for the different BabyAI levels.

As imitation learning algorithm, we use behavioral cloning (Pomerleau, 1991). We are aware that better results could possibly be obtained with a more sophisticated imitation learning algorithm such as DAgger (see Section 2.2). However, the problem with such iterative algorithms is that they often require querying the expert agent while training a new model. This seriously slows down training, especially in the complex framework we are dealing with in this project. Another drawback of such methods is that the exact training sets are always different, even within a single level, because randomly initialized agents will always query the expert in different ways. Under these circumstances it becomes unclear how to compare the learning process of different models, which is something we aim to do ceteris paribus.

In behavioral cloning, samples must be provided as pairs of input and output values. The learning goal is to predict the correct output from the input. This effectively reduces training in stages 2 and 3 to a supervised setting. In our case, the learning signal comes from the following loss function:

$$\mathcal{L}(a_t, \theta_t) = -\log P_{\theta_t}(a_t) - c_H \cdot H(P_{\theta_t}), \tag{3.9}$$

where a_t represents the gold action and θ_t the model's parameters at time t. P_{θ_t} is the model's probability distribution over actions for parameter set θ_t and $H(P_{\theta_t})$ the entropy of this distribution. c_H is a coefficient for the entropy term, fixed at 0.01 (following Chevalier-Boisvert et al. (2018)). The second term in the loss function is meant to penalize low entropy distributions, which reflect a high confidence. This encourages models to become more exploratory, and has a regularizing effect in supervised learning (Pereyra et al., 2017).

A common pitfall when training the Guide in stage 2 is getting stuck in the local optimum of always predicting the most frequent action. This is because it requires more parameter updates to optimize the discrete communication than the continuous layers, so that the policy may be inclined to ignore the guidance altogether. In these cases, the same message is always produced, and always followed by the same action. To deal with this we apply dropout to the encoded guidance, expression $\text{Encoder}^l(g_t)$ in Equation 3.8.[4]

In all our experiments, the Guide emits messages of length 2. There are three distinct tokens (`w0`, `w1` and `w2`), so that there are 9 possible unique messages in total. We deliberately keep the alphabet size small to motivate the Guide to come up with efficient combinations of the available tokens.

We train with mini-batching and shuffle the training data at the beginning of each epoch. As optimizer, we use Adam (Kingma and Ba, 2014). Hyperparameter settings equal those reported by Chevalier-Boisvert et al. (2018), insofar as possible. For new hyperparameters (pertaining to the Guide), we perform a small grid search. See Appendix B for an overview of the hyperparameter settings used in the experiments.

[4]This problem was quite persistent. We tried many other things to remedy it: adding Gaussian noise at several locations, having dropout layers in the Guide, trying different activation functions in the Guide, more heavily penalizing low entropy actions in the loss function, increasing the learning rate for parameter groups associated with message generation to encourage exploration, improving parameter initialization, tuning the Guide's dimensions more carefully, changing the optimizer, and associating weights with different classes in the loss to correct for class imbalance. In the end, applying dropout to the encoded guidance worked best.

Chapter 4

Experiments and results

In this chapter we describe the experiments conducted in the BabayAI game with the models presented in the previous chapter, together with the obtained results.

When reporting results, performance is expressed in terms of validation success rate. This is the percentage of validation episodes (out of 500) that a model manages to complete successfully without exceeding the maximum number of time steps. This limit differs per level. It depends on the maximum number of navigations required to execute a possible command, together with the size of the grid. For the levels we consider, this yields the time limits shown in Table 4.1

level	time limit
GoToObj	64
GoToLocal	64
GoToObjMaze	576
PickupLoc	64
PutNextLocal	128
GoTo	576

Table 4.1: Maximum number of time steps an agent can take to solve missions in the different BabyAI levels.

Note that the success rate is not the same metric as the accuracy, which is the percentage of individual actions that an agent predicts correctly. An agent can obtain a high accuracy and have a low success rate if she makes a few mistakes at crucial moments in game episodes. E.g. in level PickupLoc, even with impeccable navigation, if the agent does not learn to pick up the right objects, the success rate will be zero. Vice versa, it is also possible for an agent to have a low accuracy and a high success rate. This is the case if she comes up with a new, but successful strategy, so that different actions are performed at most time steps, but the missions are eventually completed. This, however, is not a likely scenario in our case, because

we use behavioral cloning as learning algorithm, which penalizes divergence from the expert trajectories.

We use success rate as the main metric of model performance, because it expresses an agent's ability to solve full missions. Accuracy only concerns individual actions, which is subordinate to this goal. This makes success rate the clearest indicator of an agent's proficiency at a given level.

4.1 The Guides

Before investigating the impact of emergent language, we first need language to emerge. This happens according to the method described in the previous chapter. At each level, we train models with the Guide set-up (introduced in Section 3.2.2), which contains a discrete bottleneck where we hope to see meaningful messages emerge autonomously.

As opposed to the combined set-up, where a Learner receives guidance messages as input to her policy module *in addition to* the processed observations and instructions, the Guide's policy module receives the produced messages as the *only* input. Thus, if the Guide is alone, the messages must contain all the information extracted from the observations and instructions that is relevant when deciding what action to take. Hence, if a single Guide manages to achieve a high validation success rate, this already indicates that she must be sending useful messages to her own policy module.

Figure 4.1 shows how the validation success rate developed for the best-performing Guides at the considered levels. Several Guides were trained at each level, but only the scores reached by the best-performing ones are shown in Figure 4.1, because there was too much variation across runs to present them in a single plot without sacrificing readability. With the 'best-performing' Guide, we mean the Guide that obtained the highest validation success rate at some point during training. These are also the Guides that we use in the subsequent experiments. In particular, we use the Guides with the parameters as they were at the training epoch when the maximum validation success rate was obtained.

Figure 4.1 contains the results for the first 120 training epochs, unless the maximum duration of 50 hours was exceeded before reaching that point (in which cases the graph just stops). Different levels come with differently sized data sets, and also the mean demonstration length varies heavily across levels (see Table 3.2). This explains why the number of completed training epochs changes so much per level.

We mainly include Figure 4.1 because it clearly shows that the Guides tend to exhibit unstable behavior during training. The validation success rates usually keep on fluctuating, sometimes with extreme deteriorations and improvements in the course of a few parameter updates.

In general, as already noted in the previous chapter, the discrete bottleneck significantly slows down model optimization. This is because small changes in internal representations, which tend to have a small effect in a fully continuous set-up, can

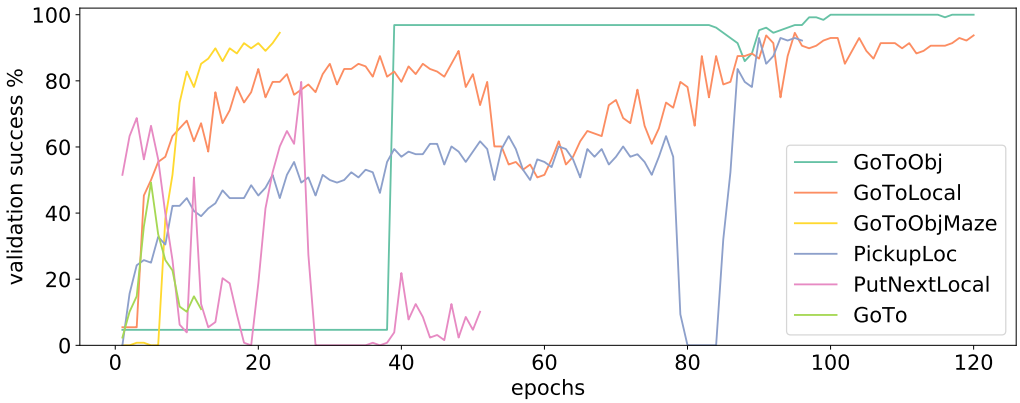


Figure 4.1: Development of the validation success rate obtained by the best-performing guides at the BabyAI levels. Results are shown for the first 120 epochs, or the first 50 training hours.

have a large 'downstream' impact when discretization is applied. For instance, a relatively small change in the distribution over tokens can cause the emission of `w1` at some time step instead of `w0`. As a consequence, the input to the policy module contains an altogether different word embedding. Such 'step-wise' changes influence the output more heavily than small mutations on a continuous spectrum.

The use of the straight-through Gumbel softmax estimator can also help to explain fluctuations when we assess a metric at validation time, as is the case for validation success rate in Figure 4.1. As explained in Section 3.2.2, this estimator uses a soft approximation of the argmax function on the backward pass, while applying a hard argmax on the forward pass. At validation time, we only have forward passes. Hence, although a model can be in the process of carefully tuning the distribution over tokens through backpropagation, the forward pass ignores all subtleties and simply selects the most likely tokens. This causes the model to behave more crudely at validation time than during training.

An overview of the maximum validation success rates obtained by the Guides is shown in Table 4.2. It also contains the validation accuracy that the Guides reach at the point where the validation success rate is highest.

Although performance clearly decays as levels become more complex, Table 4.2 shows that at most levels, we find a Guide that reaches a decent validation success rate. The only exception is GoTo, where the best-performing Guide only succeeds in 49.2% of the validation episodes. However, that does not necessarily make it a bad Guide. The validation accuracy is still 75.2%. This means that, while the Guide at GoTo cannot teach another agent how to solve most episodes, her messages can still be useful in three out of four time steps.

In general, it is important to note that the Guides are not intended to function as omniscient super-agents, who know how to perfectly solve all BabyAI missions.

level	validation success rate	validation accuracy
GoToObj	100.0	99.3
GoToLocal	95.3	83.0
GoToObjMaze	94.5	97.7
PickupLoc	93.0	86.9
PutNextLocal	79.7	82.0
GoTo	49.2	75.2

Table 4.2: Maximum validation success rate and corresponding validation accuracy (over 500 episodes) obtained by the best-performing pretrained Guides.

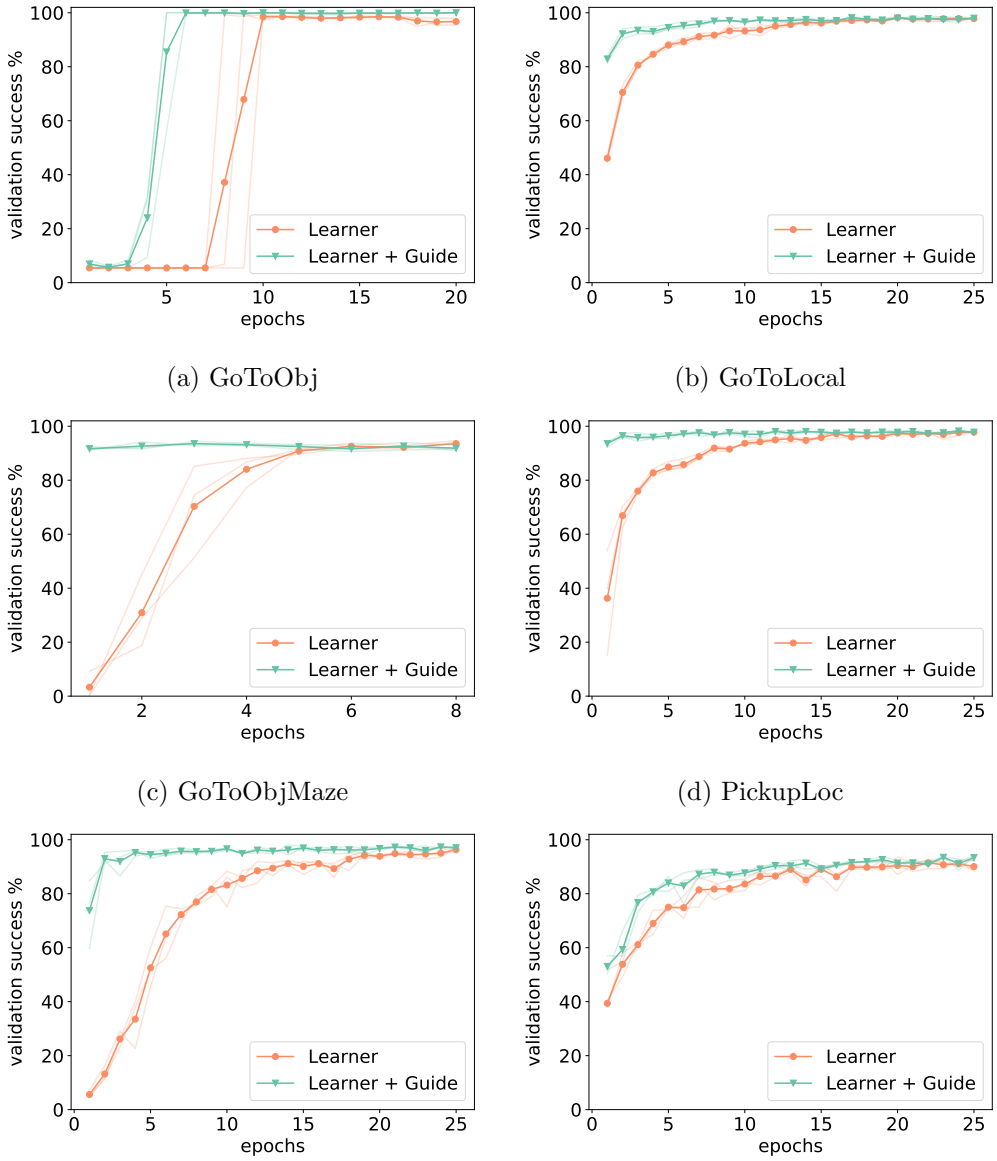
We prefer to regard them as advisors, whose pre-established knowledge of a task can prove useful to new agents, but who cannot be treated as infallible oracles. This makes the analogy with human didactic guidance more realistic. At the same time, it should motivate agents to not blindly follow the Guides. In order to reach top validation success rates on all levels, they should maintain or develop a sufficient degree of independence.

4.2 Intra-level guidance

With the best-performing Guides in place, we can now study the effect of the developed communication channel in a multi-agent setting. In this first set of experiments, we stay within single levels. We examine how a Guide pretrained at some level can help a new Learner facing that same level. We call this 'intra-level guidance'. For all of the six levels that we consider, we train the following two set-ups, each for three independent runs:

- *Learner.* This is the baseline model, which is just the default Learner described in Section 3.2.1.
- *Learner + Guide.* This is the combined set-up described in Section 3.2.3. It consists of a new Learner that receives messages from the best-performing pretrained Guide at the level concerned. This Guide is finetuned to coadapt with the Learner.

We do not expect the guided Learner to achieve a higher success rate than the baseline Learner, because we already know that the Learner is in principle able to solve all levels when given enough training time (Chevalier-Boisvert et al., 2018). What we do hope to see is that a new Learner masters a task more quickly when aided by the messages of a pretrained Guide. If the guided Learner converges to the optimal performance at an earlier point in time, this shows that the emerged language reduces the amount of computation needed to train a new agent.



(a) GoToObj

(b) GoToLocal

(c) GoToObjMaze

(d) PickupLoc

(e) PutNextLocal

(f) GoTo

Figure 4.2: Development of validation success rate when training a single Learner and a Learner assisted by a pretrained Guide on the indicated BabyAI levels. Average over three runs; results per run are shown by shaded lines.

We compare performance of the different models over time. How does their validation success rate develop as training progresses? The results for all considered levels are shown in Figure 4.2.

The plots in Figure 4.2 show that the learning curve of the guided Learner dominates the single Learner's curve in all levels. The validation success rate of the combined set-up tends to increase significantly faster and/or earlier than the one obtained by the Learner on her own. The fact that this observation holds for each of the considered levels suggests that the Guide's messages can help new agents solve arbitrary BabyAI levels more efficiently, provided that the six selected levels are indeed a representative fragment of the game as a whole.

To quantify more precisely how the messages emitted by the pretrained Guides increase the learning speed of a new agent, we include Table 4.3. It shows how many epochs the different models need to reach an average validation success rate of at least 90%.

	Learner	Learner + pretrained Guide
GoToObj	10	6
GoToLocal	7	2
GoToObjMaze	5	1
PickupLoc	8	1
PutNextLocal	14	2
GoTo	20	12

Table 4.3: Overview of number of training epochs required to reach a mean validation success rate of 90% for a baseline Learner and a Learner assisted by a pretrained Guide.

Table 4.3 demonstrates explicitly that in all of the considered levels, Learners that receive guidance messages reach the 90% success threshold considerably faster than those that have to address the task without additional language support. In levels GoToObjMaze and PickupLoc, one training epoch is already enough for the combined set-up to surpass the threshold.

Returning to the plots in Figure 4.2, we note some interesting differences between the results for the six levels. At GoToObj, the single Learner and the combined set-up have learning curves with a virtually identical shape. The only difference is that the guided Learner starts improving approximately four epochs earlier. At most other levels, the guided Learner's learning curve is also steeper. Of course, the development of the default Learner's validation success rate is already quite steep at GoToObj, which is probably the case because this level is very simple (which is why 1,000 training demonstrations is enough).

The plots for levels GoToLocal, PickupLoc and PutNextLocal are quite similar. In all of these cases, the single Learner's validation success rate increases at a gradually decreasing rate until convergence at the maximum score. The guided Learner's

curve converges to the same score, but already in one of the earliest epochs. The plot for GoToObjMaze is comparable, except that at this level, the combined set-up does not improve at all after the first epoch.

Of all levels, results are least successful for GoTo. As visible in Figure 4.2f, the learning curve of the guided Learner at this level is actually very close to the single Learner's one. There is no epoch where the guided Learner is on average worse than the single Learner, but the difference is definitely a lot smaller than we see at the other levels. This can be explained by recalling Table 4.2. Both in terms of validation success rate and validation accuracy, the best-performing Guide at GoTo is outperformed by all other Guides. We noted that this does not mean that the Guide is useless, but her suboptimal performance is clearly reflected by the plot in Figure 4.2f.

Another remark we can make about Figure 4.2 concerns the individual model runs. These are included as shaded lines in the plots. What we see is that they are very close to each other. This shows that there is little variation across runs. Apparently, the results are quite stable in this respect: they are hardly sensitive to random initialization of new module parameters or stochastic batching. Both the single Learner and the guided one always seem to find a good solution, and it never happens that a model fails to converge. This is a big difference with the training of the Guides (described in the previous section), where we found models to remain quite unstable even after many epochs.

The results of the intra-level guidance experiments prove that access to the messages produced by a pretrained Guide can greatly speed up learning of new agents. Fewer training epochs are needed to reach the same maximum validation success rate. This is already an important observation in its own right. Yet, it also suggests something else, namely that the guided Learner could do with less data than the single Learner. We only present this as a conjecture and do not investigate it any further, because it would require running many more experiments with a variety of differently sized data sets. However, it seems plausible that an agent who knows how to use data more efficiently - such as the guided Learner - could master the same task with fewer training samples.

4.3 Curriculum guidance

It has often been shown that a curriculum of tasks with gradually increasing difficulty can be helpful when training artificial agents (Graves et al., 2016; Bengio et al., 2009; Zaremba and Sutskever, 2014). Moreover, there is a natural analogy between a hierarchy of increasingly complex challenges and the didactical methods used by humans to teach others (Khan et al., 2011). See Section 2.4 for other relevant publications on this topic.

Thanks to its systematic ranking of levels, BabyAI is an excellent platform to study the impact of curriculum learning. Chevalier-Boisvert et al. (2018) already showed that pretraining a Learner on a BabyAI level requiring certain skills can

improve data efficiency when confronting this Learner with a new level that requires these capacities as well. We assess if this also holds for the Guide: are the messages emitted by a Guide pretrained at one level helpful for a Learner facing a more complex one? We call this kind of guidance 'curriculum guidance'.

We consider the transfer between the levels shown in Table 4.4.

base level	target level
GoToLocal	PickupLoc
PutNextLocal	PickupLoc
GoToLocal	PutNextLocal
PickupLoc	PutNextLocal

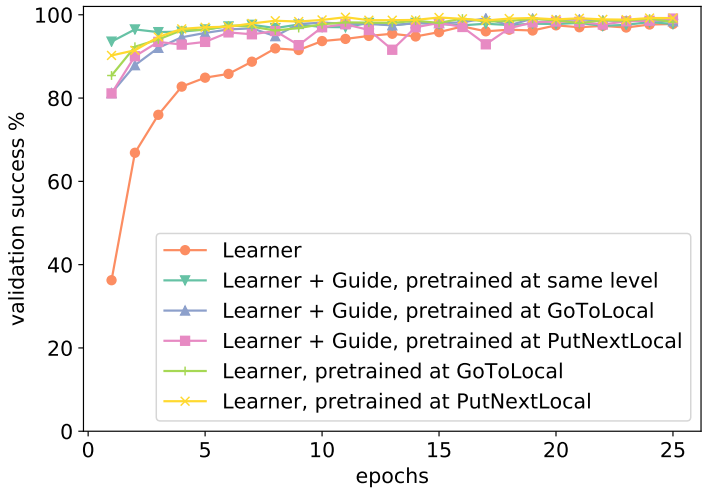
Table 4.4: Levels used in curriculum guidance experiments. Guides pretrained at the indicated base levels assist new Learners at the associated target levels.

Guides are trained at the base levels listed in Table 4.4, and coupled with new Learners at the related target levels. In all of these cases, we hypothesize that the messages sent by the Guide should be useful also at the target levels, even though the Guide never actually encountered them. This is because all of the levels concerned require an agent to be capable of navigating a single room with distractors (competencies ROOM, DISTR-BOX and DISTR in Table 3.1). Hence, if the guidance messages are sufficiently general, they should be useful across base and target levels.

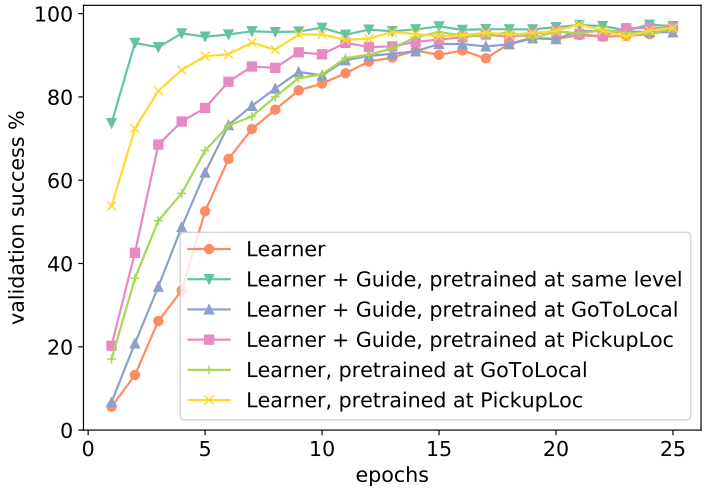
We compare performance of the combined set-ups with different base and target levels (curriculum guidance) with those for which base and target level coincide (intra-level guidance, as discussed in the previous section). Additionally, we assess the performance at the target levels of a single Learner pretrained at the base levels. We expect a pretrained Learner to perform better than a new Learner assisted by a pretrained Guide, because in the former all parameters have been optimized for a related level, whereas in the latter the Learner can only profit from the pretraining by means of the discrete guidance messages.

Figure 4.3 shows the results. In level PickupLoc, we see that pretraining the Guide at levels GoToLocal and PutNextLocal brings the learning curve close to the one of a learner with a Guide pretrained at PickupLoc itself. Surprisingly, the difference between the new Learners with Guides pretrained at the base levels and the Learners pretrained at the base levels is not significant. The Learner that is trained from scratch at PickupLoc has most difficulties and converges later than all other models.

In PutNextLocal, the results are more divergent, and the Learner with a Guide pretrained at the same level clearly outperforms all other set-ups. Pretraining at PickupLoc is more effective than pretraining at GoToLocal. This is the case for pretraining of the Learner, but also for pretraining of the Guide. Pretraining the



(a) PickupLoc



(b) PutNextLocal

Figure 4.3: Development of validation success rate when training a single Learner from scratch, a Learner assisted by a pretrained Guide, or a pretrained Learner on several BabyAI levels. Average over three runs; results per run are not shown to avoid cluttered plots.

Learner tends to accelerate learning more than pretraining of the Guide. As at PickupLoc, the learning curve of the single Learner without any pretraining is dominated by all the other ones.

Although the results differ depending on the base and target levels, they do show that Learners can profit from Guides that were pretrained at different levels. The observation that Guides can even be helpful to new Learners at levels for which they were not optimized suggests that their messages convey information that is relevant to the internalization of skills that generalize across levels. Relating this to the overview of competencies needed to solve BabyAI levels (shown in Table 3.1), it seems that curriculum guidance is successful because the Guides produce messages that capture the particular skills identified as prerequisites for solving PickupLoc and PutNextLocal. But even if the messages do not correspond exactly with isolated skills, the results of this section show that the relevance of the emerged language is not restricted to individual levels.

It is especially surprising that curriculum guidance is helpful at the levels that we consider in these experiments, because the instructions presented at GoToLocal, PickupLoc and PutNextLocal are expressed with different vocabularies. That is, at each of these three levels agents encounter words in the instruction strings that do not occur at the other two levels. This implies that the pretrained Guides are confronted with unknown words in the instructions, but nevertheless they manage to help the Learners. Perhaps the Guides overcome this handicap quite easily because the vocabularies are small, and the instructions follow the same grammatical structure at single levels, so that new expressions can quickly be accommodated.

Something very striking is that the scores obtained by Learners pretrained at certain base levels tend to be very close to those obtained by Learners assisted by Guides pretrained at those base levels. We did not expect this, because a pretrained Learner comes with a fully optimized set of parameters, whereas a Learner assisted by a pretrained Guide only receives the two-token messages. The observation that the guidance message channel, pretrained at a base level, can have an effect that is almost as strong as a full set of pretrained model weights shows how powerful the emerged language can be.

4.4 Multi-level guidance

The curriculum guidance experiments focused on the transfer of guidance from one level to another one. This transfer can be considerd *inductive* because it relies on generalization from known levels to unseen ones. In the next experiments we also check for a *deductive* transfer: if a Guide is trained on a combination of levels, how useful is she when assisting a Learner trained from scratch on only one of these levels?

To investigate this, we train a single Guide on GoToLocal, PickupLoc and PutNextLocal at the same time, by aggregating the training sets of these three levels (yielding a new training set of 250K demonstrations). We call this agent the 'multi-

level Guide'. None of the model dimensions are changed, implying that the multi-level Guide has to address three different levels with the same number of parameters as a regular Guide. Because of this handicap, we hypothesize that the multi-level Guide will not accelerate the development of a new Learner as rapidly as a Guide specialized at a single level.

The best-performing multi-level Guide obtains the following validation success rates at the three different levels:

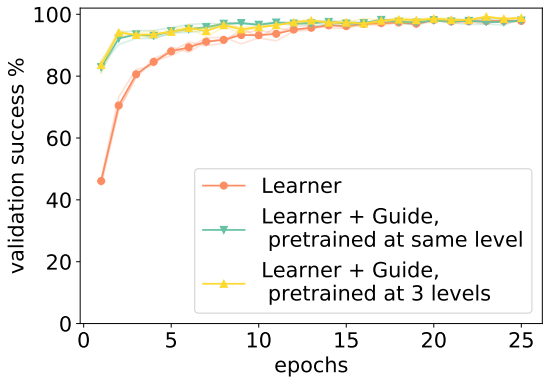
level	validation success rate
GoToLocal	48.4
PickupLoc	93.8
PutNextLocal	78.1

Table 4.5: Validation success rates per level of the best-performing multi-level Guide.

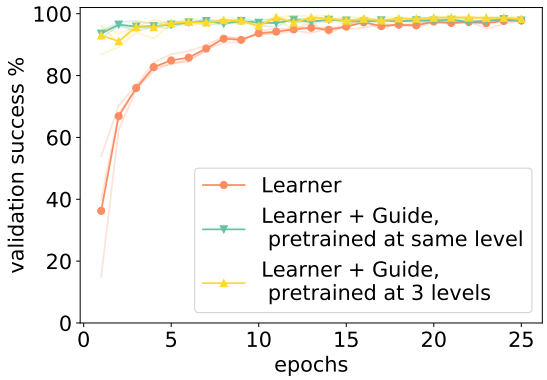
Interestingly, the multi-level Guide reaches the lowest validation success rate at the easiest level: GoToLocal. We can probably explain this by noting that the training set for GoToLocal only contains 50K demonstrations, whereas the ones for PickupLoc and PutNextLocal contain 100K samples (see Table 3.2). The multi-level Guide is trained on the combination of these data sets, so GoToLocal is somewhat underrepresented. The best-performing multi-level Guide's mean validation accuracy is 84.1%.

Next, we initialize new Learners. We train them separately at each of the three levels, assisted by the pretrained multi-level Guide. The results of this experiment are visualized in Figure 4.4. At all three levels the multi-level Guide turns out to be just as helpful to a new Learner as a Guide that was specifically optimized for a single level. This is an unexpected result, because the multi-level Guide seems to be at a disadvantage for having to approach three levels with the same number of parameters that the ordinary Guides use to solve only one level. Apparently, this assumption was mistaken, and the Guide does not need extra weights to solve the extra levels.

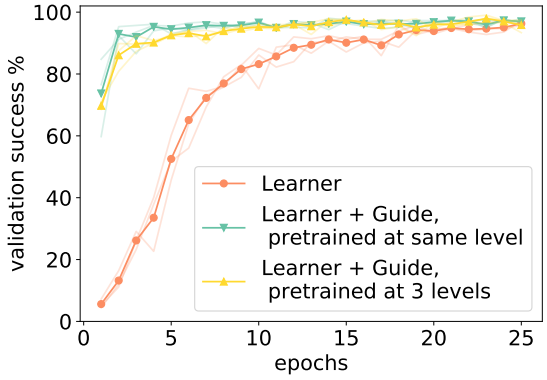
This result again suggests that the Guide does not produce messages aimed at solving specific levels, but rather at acquiring skills that are relevant in each of them. Of course, although the multi-level Guide has the same number of parameters as an ordinary Guide, she witnesses more demonstrations during training (250K vs 100K or 50K). This can be an advantage, but only if shared aspects of the aggregated levels are exploited. Otherwise, it would mean that the Guide approaches the larger data set as the conjunction of three independent tasks, which should be more challenging. The fact that the Guide seems to have no difficulties handling three levels at once shows that she must indeed be profiting from commonalities between the levels, such as the core competencies listed in Table 3.1. Perhaps we can also infer from



(a) GoToLocal



(b) PickupLoc



(c) PutNextLocal

Figure 4.4: Development of validation success rate when training a single Learner, a Learner assisted by a Guide pretrained on a single level and a Learner pretrained on GoToLocal, PickupLoc and PutNextLocal simultaneously. Average over three runs; results per run are shown by shaded lines.

this result that the Guide needs fewer parameters to solve individual levels than are currently being used.

The plots for the three levels in Figure 4.4 are very much alike. Even at GoToLocal, where the multi-level Guide only obtained a validation success rate of 48.4%, Learners profit as much from assistance by the multi-level Guide as from assistance by the Guides that were optimized at single levels. The results are also quite stable across runs: looking at the shaded lines, which show the results per run, we see that they tend to be very close to the mean.

4.5 Entropy conditioning

In the combined Learner-Guide set-up used so far, the Learner has uncharged and unrestricted access to the messages sent by the pretrained Guide at each timestep. Because the Guide is meant to have an assisting role, we would like to prevent the Learner from becoming overly focused on the guidance messages. We propose to do so by weighing the input to the policy coming from the Guide at time t, i.e. the encoded guidance, with a 'guidance weight' w_t, which is a scalar coefficient satisfying $0 \leq w_t \leq 1$. Equation 3.8 then becomes:

$$\hat{a}_t = \pi(x_t^l; w_t \cdot \text{Encoder}^l(g_t)) \tag{4.1}$$

The weighing factor w_t is conditioned on the entropy of the Learner's probability distribution over actions at the previous timestep, which is denoted by $H(P_{\theta_{t-1}})$, following the notation explained on page 25. We call this method 'entropy conditioning'. The underlying intuition is to have a Learner assign more weight to the guidance messages in situations where she faces a relatively high uncertainty, as reflected by high entropy values at the immediately preceding time step. As the Learner grows more confident, guidance should become less important. w_t is computed as

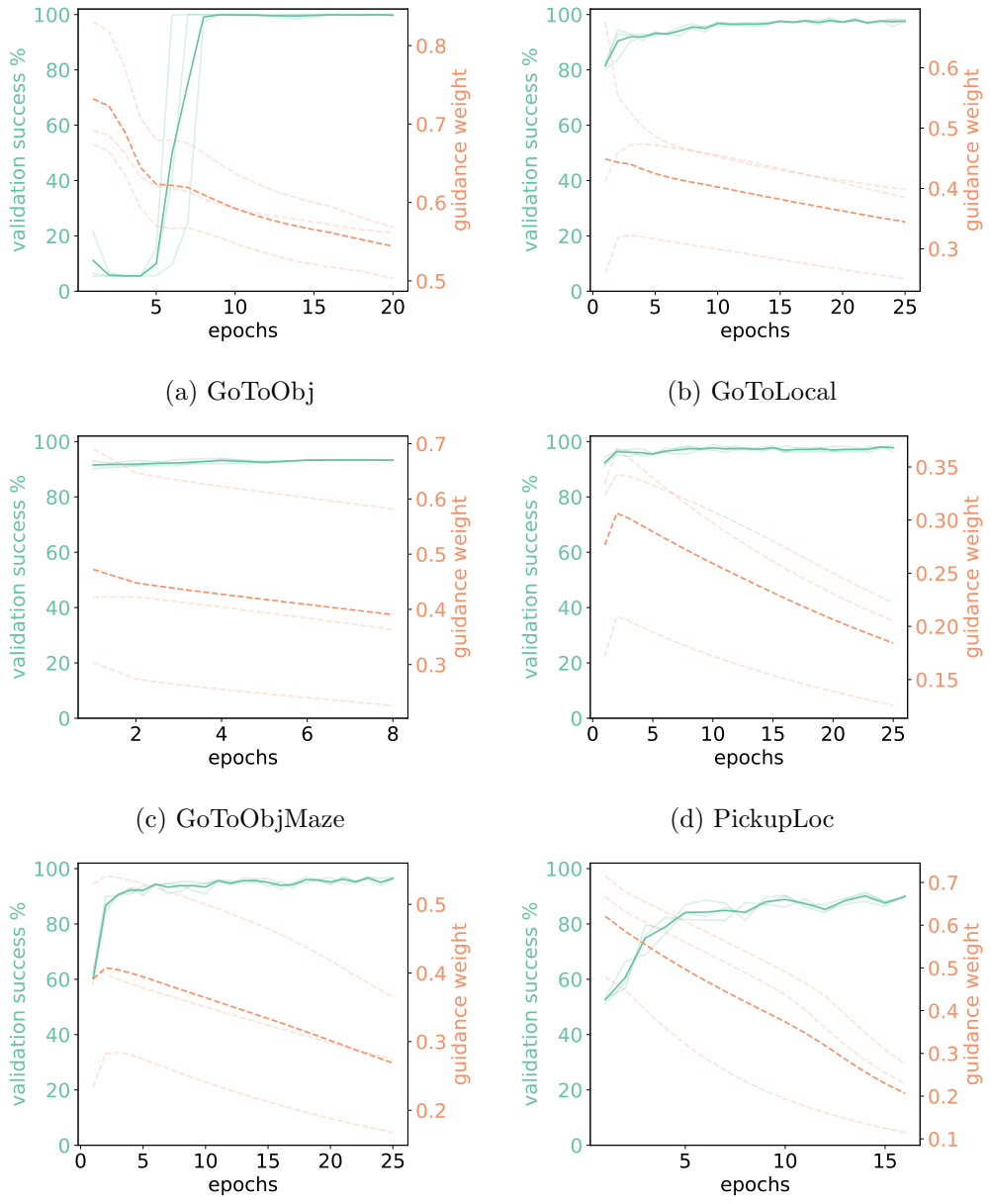
$$w_t = \sigma(\text{Linear}(H(P_{\theta_{t-1}}))), \tag{4.2}$$

where σ represents the sigmoid function and Linear a linear layer (which in this case is simply 1×1-dimensional because it maps scalar entropies to scalar guidance weights). To incentivize autonomy of the Learner, as reflected by low guidance weights, the term w_t is added to the loss function (Equation 3.9), which now becomes:

$$\mathcal{L}(a_t, \theta_t) = -\log P_{\theta_t}(a_t) - c_H \cdot H(P_{\theta_t}) + c_G \cdot w_t \tag{4.3}$$

In this new loss function c_G represents a coefficient scaling the guidance weight. It was tuned in the range $[0.5, 1.0, 2.0, 5.0, 10.0]$ and found optimal at $c_G = 2.0$.

On all six levels we train three new Learners, assisted by Guides pretrained at those levels, with entropy conditioning. The development of the validation success rate during training is plotted together with the mean guidance weight in Figure 4.5.



(a) GoToObj

(b) GoToLocal

(c) GoToObjMaze

(d) PickupLoc

(e) PutNextLocal

(f) GoTo

Figure 4.5: Development of validation success rate and mean guidance weight when training an entropy-conditioned Learner assisted by a pretrained Guide on the indicated BabyAI levels. Average over three runs; results per run are shown by shaded lines.

Comparing these results to those shown in Figure 4.2, we see that the validation success rate develops in roughly the same way with and without entropy conditioning. Apparently, applying guidance weights does not affect performance with coefficient $c_G = 2.0$. (Higher values of c_G can delay convergence because a larger guidance weight loss term encourages models to ignore the messages.)

If we look at the development of the mean weight assigned to the guidance messages, we see that it decays almost monotonically over time in all considered levels. In the early training stages the benefit of learning from the Guides' messages still outweighs the cost of consultation. This changes as learning progresses and the Learners appear to becomes less dependent on the Guides. The plots show that the guidance weight keeps decreasing after the validation success rate has stabilized.

Entropy conditioning modifies the combined set-up by penalizing the Learner for consulting guidance messages. The fact that this mechanism induces a consistently decaying guidance weight shows that it is effective in reducing the Learner's attention to the component of the policy input that stems from the pretrained Guide. However, this observation on its own is not sufficient to conclude that the Guide's causal influence on the Learner's actions is significantly reduced. It could happen, for instance, that the weights in the Learner's policy module associated with the encoded guidance increase as the guidance weight decreases, thereby counteracting the intended effect. In the entropy conditioning experiment we did not control for such dynamics, which is why we conclude with this caveat. A better way of assessing the Guide's causal influence on the Learner is to use a metric specifically designed for this purpose. Section 5.3 in the next chapter is dedicated to an analysis by such means.

Chapter 5

Analysis

This chapter is dedicated to more in-depth analsyis and interpretation of the results reported in the previous chapter, focusing in particular on the language that the agents have evolved.

5.1 Messages/actions correlations

First, we analyze the relation between messages produced by the Guide and actions taken by the Guide herself (in training stage 2) or by a coupled Learner (in training stage 3). To do so, we consider the conditional probability distribution $P(A \mid M)$. Here, A is the random variable representing the action taken by an agent. This can be the Guide when she is still on her own, or the new Learner that she assists. M is the random variable representing the message produced by the Guide.

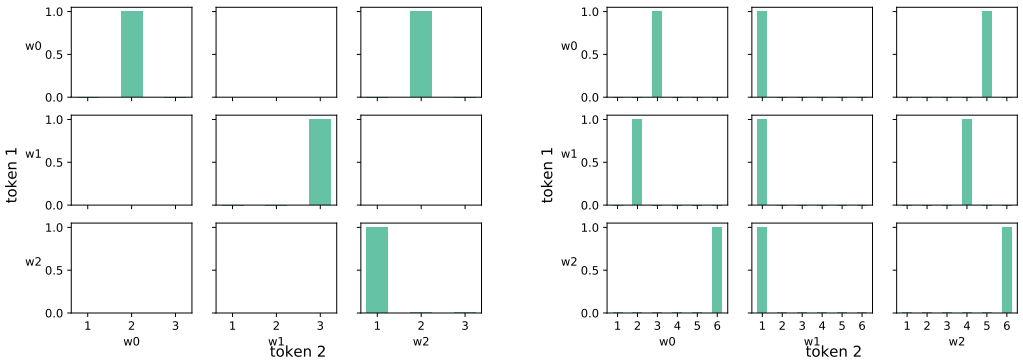
We first consider the best-performing single Guides (training stage 2), and estimate the distribution $P(A \mid M = m)$ for each possible guidance message m. This we do by sampling 500 episodes, unseen during training, and logging the messages that the Guide produces at the individual timesteps, together with the actions that she takes at each of them. For any possible action a and message m, we estimate the conditional probability $P(A = a \mid M = m)$ by means of maximum likelihood estimation (MLE).

The results are shown in Figure 5.1. We include the statistics for GoToObj, the most basic level, and PutNextLocal, because here the agent has to perform most actions. Plots for other levels can easily be generated, but are omitted here because they exhibit very similar patterns.

The two subfigures in Figure 5.1 contain nine barplots, one visualizing the conditional distribution $P(A \mid M = m)$ for each possible message m. In the figure, rows represent the first token in a message, and columns the second one. In Figure 5.1a, for instance, the top left barplot is in row `w0` and column `w0`, meaning that it visualizes the distribution $P(A \mid M = \texttt{w0 w0})$. The bins in the barplots correspond to actions. In the figure for level GoToObj, there are only three actions, because the agent has learned correctly that all she needs to do is turn left, turn right or move

forward. At PutNextLocal, agents also have to pick up, drop and toggle, so there are three more actions.

The barplots in Figure 5.1 reveal a strong correlation between messages and actions. Both at level GoToObj and PutNextLocal, we basically see a one-to-one mapping between messages and actions. This means that the best-performing Guides tend to execute the same actions after emitting particular messages.



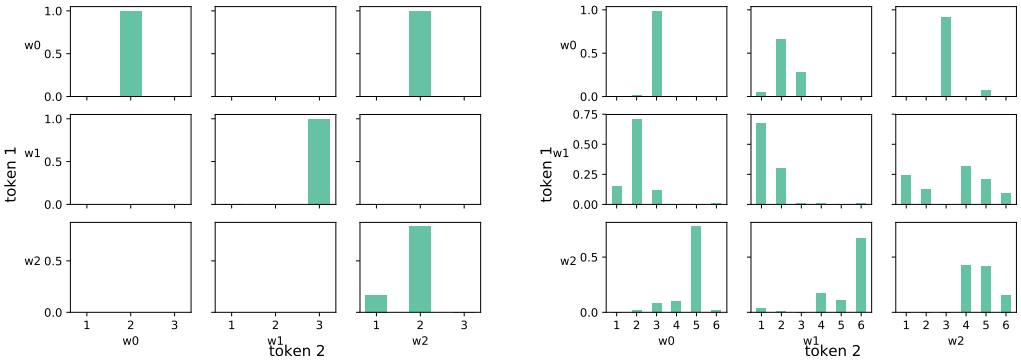
(a) GoToObj (b) PutNextLocal

Figure 5.1: Barplots visualizing the conditional distribution of actions taken by the best Guide in stage 2, given the messages she produces, based on 500 sample episodes unseen during training. The bins in the individual barplots correspond to actions and the bar heights to probabilities. E.g. the top left barplot in Figure (a) shows that the Guide performs action 2 with probability 1.0 after emitting message `w0 w0`.

Next, we consider the distribution $P(A \mid M = m)$ of the best-performing guided Learner per level after training stage 3. Now the action A is not the action taken by the Guide, but by the Learner, while M still represents the message produced by the pretrained (and finetuned) Guide.

The distributions change after pairing the Guide with a new Learner, and gradually become less spiky as learning progresses. This is clearly visible in Figure 5.2, which shows the conditional distributions of actions taken by the best-performing Learners, conditioned on the messages sent by the finetuned Guides. Especially at PutNextLocal the one-to-one mapping between messages and actions has disappeared, although per message there is usually still one action that receives a significantly higher probability than the other ones.

The difference between the distributions in Figures 5.1 and 5.2 can be due to development of the Guide, or a changing interpretation of the Guide's messages by the Learner. To check how much the Guide changes during training stage 3 (when paired with a new Learner), we isolate the Guide from the best-performing Learner-Guide combination and pair it with the policy of the best-performing isolated Guide from stage 2. We compare the performance of the old Guide and the old policy with



(a) GoToObj (b) PutNextLocal

Figure 5.2: Barplots visualizing the conditional distribution of actions taken by the best Learner in stage 3, given the messages produced by the finetuned Guide, based on 500 sample episodes unseen during training.

the performance of the finetuned Guides and the old policy. If there is a notable change, this means the Guide's message distribution must have changed. Otherwise, it cannot have developed significantly, because the isolated Guide's performance is fully dependent on the message channel.

	pretrained Guide + old policy	finetuned Guide + old policy
GoToObj	100.0	100.0 ± 0.0
GoToLocal	95.3	68.1 ± 18.8
GoToObjMaze	94.5	16.1 ± 1.8
PickupLoc	93.0	12.6 ± 0.5
PutNextLocal	79.7	0.1 ± 0.1
GoTo	49.2	9.4 ± 0.9

Table 5.1: Validation success rate (over 500 episodes) of the pretrained Guide with her own policy, and the finetuned Guides with the old policy (average over three runs with standard deviation).

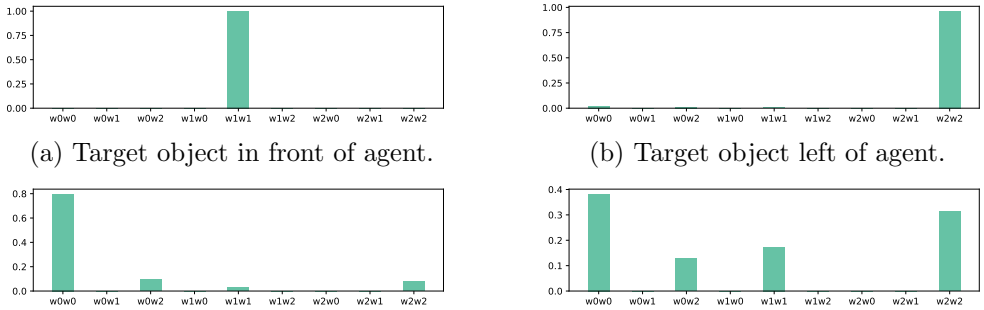
Results of this experiment are shown in Table 5.1. The pretrained Guide + old policy results are identical to those in Table 4.2, but repeated here for easier comparison. Note that, although there is only one pretrained Guide per level, there are three finetuned ones: one for each assisted Learner. Except at GoToObj, all finetuned Guides face great difficulties when combined with the old policy. This shows that most Guides must have undergone significant changes during stage 3, adapting to the development of the Learners they are supposed to assist.

5.2 Input/messages correlation

In the previous section we considered the relation between guidance messages and actions, which is essentially the relation between messages and model *output*. Now we also want to study the relation between the messages and model *input*: how do observations trigger particular messages?

For this purpose, we consider the conditional distribution of the Guide's messages, given the observational input. More particularly, we categorize the agent's field of perception according to four salient observable facts: whether the target object lies directly ahead of the agent, on her right, on her left, or whether the target object is not currently visible. These events are mutually exclusive and exhaustive, which is to say that exactly one of them is the case at any given point in time. If we let O be the random variable representing these observations, we are now considering the conditional distribution $P(M \mid O)$.

For levels GoToObj and GoToLocal, Figures 5.3 and 5.4 visualize the distribution $P(M \mid O = o)$ for any observable fact o of the four options distinguished before. Very similar plots can be obtained at other levels but are omitted here for the sake of brevity. As in the preceding section, the probabilities are estimated using MLE on data gathered after running the best-performing pretrained Guides on 500 sample episodes unseen during training. For each time step in each of these episodes, we check the emitted guidance message m, and determine the observable fact o by interpreting the agent's visual input.



(a) Target object in front of agent.

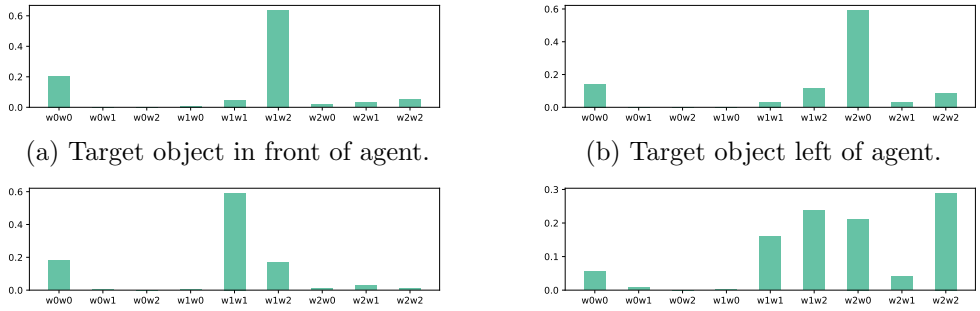
(b) Target object left of agent.

(c) Target object right of agent.

(d) Target object invisible.

Figure 5.3: Barplots visualizing the conditional distribution of messages produced by the best Guide in stage 2 at level GoToObj, given the indicated observable events, based on 500 sample episodes unseen during training.

The figures show that the Guides have a clear preference for a particular message when the object that has to be navigated towards lies ahead, on the left or on the right. Appparently, the guidance message distribution is conditioned on the observation of such situational circumstances. When the target object is invisible, the distribution is more spread out, indicating that the agent lets her policy depend on other factors in such cases.



(a) Target object in front of agent.

(b) Target object left of agent.

(c) Target object right of agent.

(d) Target object invisible.

Figure 5.4: Barplots visualizing the conditional distribution of messages produced by the best Guide in stage 2 at level GoToLocal, given the indicated observable events, based on 500 sample episodes unseen during training.

We note that the distributions for GoToLocal are more diffuse than those for GoToObj, indicating a higher uncertainty about the messages to emit. This may be due to the presence of distractors in GoToLocal, which is not the case in GoToObj. Moreover, there are bound to be other observable facts in the environment that impact the distributions, and that we have not taken into consideration in this analysis.

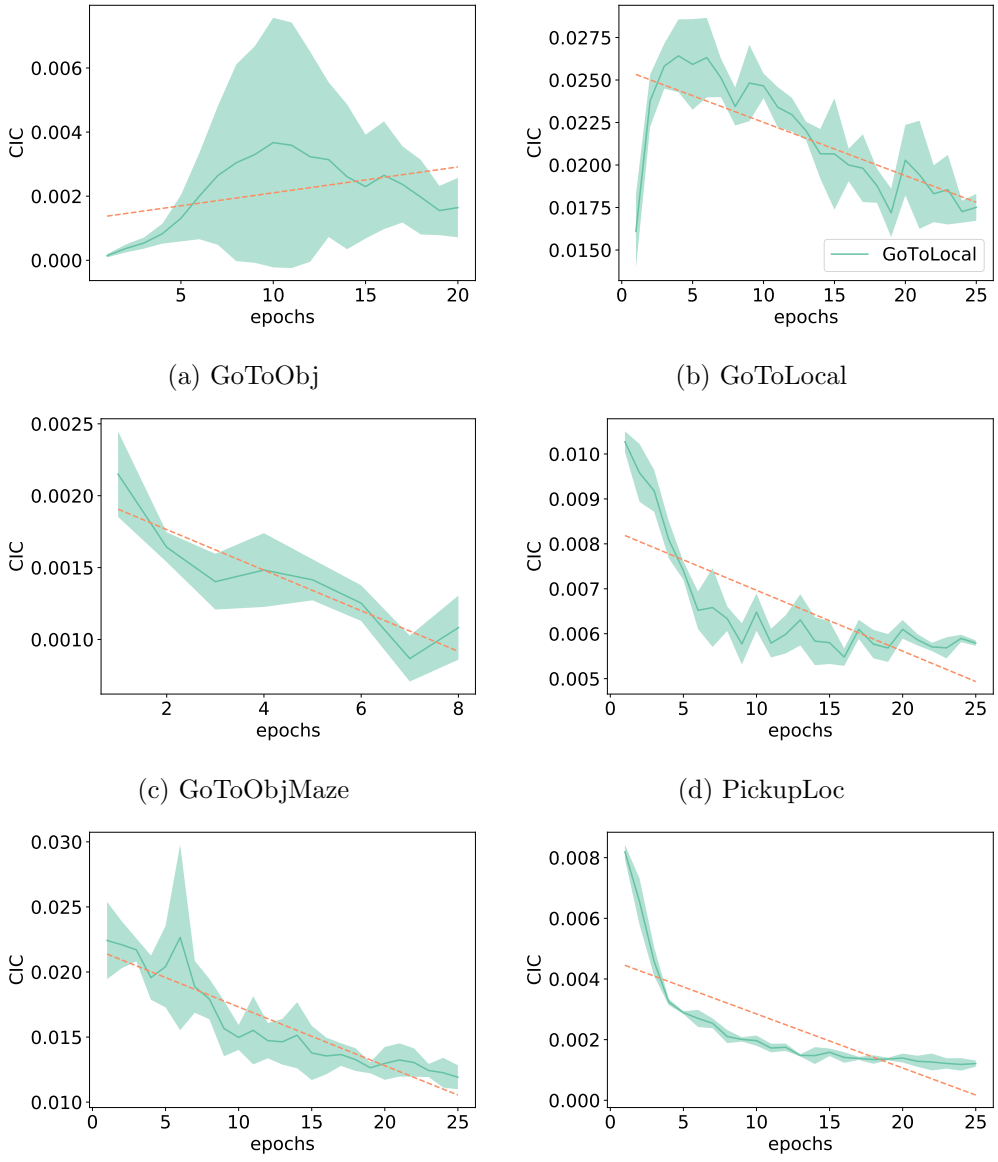
5.3 Causal influence

In the analysis so far, we looked at a few statistical correlations. This does not tell us anything yet about causality: do the Guide's messages actually *cause* behavioral patterns in the Learner? To quantify the impact that the guidance messages have on the actions taken by a new Learner, we implement and compute a metric introduced for this purpose by Lowe et al. (2019): Causal Influence of Communication (CIC).

CIC is intended to measure 'positive listening': the extent to which emergent communication influences an agent's behavior. We consider one-step causal influence, c.q. the effect of the Guide's message on the next action of the Learner. It is computed as:

$$\text{CIC} = \frac{1}{\mid T \mid} \sum_{t \in T} \sum_{m \in M} \sum_{a \in A} P_t(a, m) \log \frac{P_t(a, m)}{P_t(a) P_t(m)}, \tag{5.1}$$

where T is the set of 'test games' (c.q. the frames of the validation episodes), M the set of messages the Guide can send and A the set of actions the Learner can take. Each $t \in T$ determines the probabilities $P_t(a)$, the probability of the Learner executing action a, and $P_t(m)$, the probability of the Guide producing message m, as well as the joint probability $P_t(a, m)$. Here we make use of the following:



(a) GoToObj

(b) GoToLocal

(c) GoToObjMaze

(d) PickupLoc

(e) PutNextLocal

(f) GoTo

Figure 5.5: Development of CIC on 500 validation samples when training a Learner assisted by a pretrained Guide on the indicated BabyAI levels. Average over three runs; standard deviations are shown by shaded regions. Dashed orange lines show linear trends.

$$P_t(a) = \sum_{m \in M} P_t(a, m) \tag{5.2}$$

$$= \sum_{m \in M} P_t(a \mid m) P_t(m) \tag{5.3}$$

In fact, the one-step CIC metric that we use here is simply the mean mutual information between the messages that the Guide produces and the actions that the Learner performs. The interpretation in terms of causality is licensed by the fact that, when we intervene to change the message m, we compute $P_t(a, m) = P_t(a \mid m)P_t(m)$, where $P_t(a \mid m)$ is the probability of the *Learner* performing action a given message m, whereas $P_t(m)$ represents the probability of the *Guide* producing message m. Hence, when marginalizing as in Equations 5.2 - 5.3, we relate the probability distributions generated by two distinct agent, which according to Lowe et al. justifies using the term 'causal influence'.

Figure 5.5 visualizes the development of the CIC metric in the set-up combining a new Learner with a pretrained Guide at all considered levels (training stage 3). The plots clearly show that CIC tends to decrease over time, as illustrated by the linear trendlines. This indicates that the guidance messages have a high causal effect on the Learner's actions in the early stages, and gradually become less important as training progresses and the Learner grows more confident. The only exception in Figure 5.5 is level GoToObj, where we see a positive correlation between training time and CIC. We note, however, that also at this level the trend is negative if more epochs after model convergence are considered, as is visible in Figure 5.6.

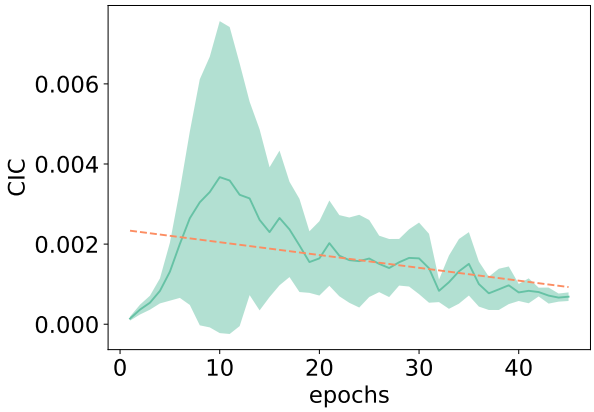


Figure 5.6: Development of CIC on 500 validation samples when training a Learner assisted by a pretrained Guide at level GoToObj for 45 epochs (instead of the usual 20). Average over three runs; standard deviation is shown by shaded region. Dashed orange line shows linear trend.

It is interesting to see that the combined Learner-Guide set-up has a natural tendency to reduce the causal influence of the guidance messages over time. There are no explicit biases or deliberate mechanisms (such as the entropy conditioning proposed in Section 4.5) to enforce this effect. It reveals a nice analogy to human learning processes, where a student is also expected to grow less dependent on a teacher as the education progresses. Step by step, a student discovers how to solve a task without external support. This is also the case for our guided Learners: the causal influence of the messages decreases, but the validation success rate does not (as we saw in Figure 4.2). This means that the Learners find a way of addressing the game levels without relying on the pretrained Guides, which is only possible if they sufficiently optimize those parameters that are not conditioned on the received messages.

5.4 Learning the agents' language

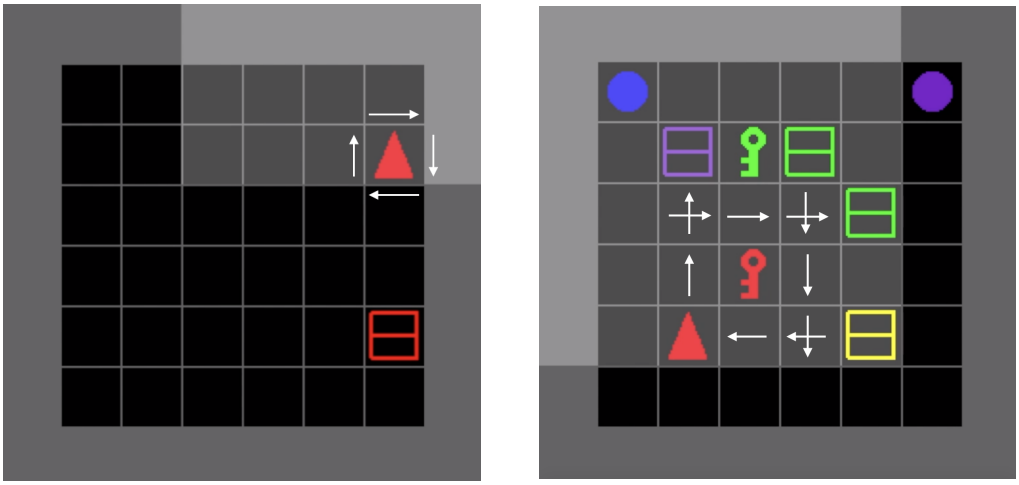
The correlations between actions and messages revealed in Section 5.1 taught us how to interpret the Guide's language. This means that we do not only know how to make sense of witnessed utterances, but that we can also try to speak the language ourselves, encouraging the Learner to perform the actions that have proven to correlate with particular messages.

To test this, we train a Learner with a pretrained Guide at levels GoToObj and GoToLocal, and identify the epoch where the CIC is highest. This should be the point in time where the Learner is most susceptible to the message channel. We compute the correlations between guidance messages and performed actions at this stage, as in Section 5.1, to establish the expected correspondence between messages and actions. Next, we 'hijack' the Guide, and start sending our own messages to the Learner.

Communication success is assessed in two ways. First, we provide the Learner with a set of scripted messages that should describe a pre-set trajectory (a 'choreography') in the grid, as per the computed message-action correlations. In particular, we try to tell the agent to turn around its own axis (the 'pirouette'), and to describe a larger circular movement (the 'waltz'). We check if the performed action sequences adhere to these choreographies. Second, we report quantitative results by randomly sending the Learner 500 messages that strongly correlate with specific actions. We then calculate the 'obedience': the percentage of messages that are followed by the expected action.

In GoToObj, the Learner successfully performs a pirouette and a waltz upon receiving the guidance messages that are supposed to describe these movements. See Figure 5.7a for the pirouette. This is also the case in GoToLocal. See Figure 5.7b for the waltz. In GoToObj we reach an obedience of 76%, and in GoToLocal one of 72%.[1]

[1]For videos of the dancing agents, see `https://github.com/MathijsMul/babyai-emergent-guidance/tree/master/videos`.



(a) Pirouette in GoToObj

(b) Waltz in GoToLocal

Figure 5.7: Trajectories of Learners at indicated levels when told by means of messages to perform (a) a pirouette (c.q. $4 \times$ `w0 w0`) and (b) a waltz (c.q. $4 \times$ `w1 w2`, `w1 w2`, `w1 w1`)

The obedience percentages show that the Learner does not blindly follow the guidance messages, even when the CIC is highest. The observational input always remains important, especially when random messages are sent so that the correlation between input and guidance is broken. Some of the cases where the Learner does not follow the random guidance can be accounted for by noting that not all recommended actions are always adequate. For instance, when an agent is facing a wall, she cannot follow a message that tells her to move forward. However, in most cases the agent does what we tell her to. This means that, to some extent, we have mastered the agent language.

Chapter 6

Conclusion

We showed how neural agents can autonomously develop a discrete communication channel to help each other solve the BabyAI game. New agents that are guided by pretrained ones converge significantly faster to optimal performance than agents that have to address a level on their own. By applying curriculum learning, we demonstrated that the guidance messages encode information that is general enough to be useful in unseen levels. We also showed that training a Guide on several levels at the same time does not negatively impact the influence of the messages at the individual levels.

In the analysis, we saw that messages correlate with actions as well as with observations. Application of the CIC metric revealed that new agents gradually become more independent of the guidance messages they receive. Finally, we used the interpretability of the emerged language to successfully communicate with agents directly.

All in all, this thesis makes three main contributions:

- introduction of an autonomous agent that can interactively guide other agents via emergent, discrete communication, speeding up the learning in a collection of sequential decision making problems
- demonstration that the developed language is general enough to be reused in levels different from the one it was developed on
- extensive analysis of the observed communication, which enables us to interpret the agents' language, and even to address them directly

In future research we would like to extend the developed method to frameworks other than BabyAI. We also wish to investigate methods to break the strong correlation between messages and actions, and incentivize guidance of a more general nature. Moreover, we are interested in experimenting with the number of tokens, the message length and the learning method, to see if similar results can be obtained if we use reinforcement instead of imitation learning.

We hope that this thesis has shed some light on relevant questions in the research on emergent language. And if we did not succeed in explaining everything as clearly as possible, we hope that our results speak for themselves - just like our agents.

Appendix A

Theoretical background

In this appendix we provide a basic introduction to neural networks and how to train them. This information is assumed familiar for most readers, but included because it concerns notions that play an essential role in the thesis. Much of the explanation given here stems from the theoretical background chapter of Mul (2018).

A.1 Neural networks

The models used in this thesis are artificial neural networks, a class of machine learning algorithms that combine linear transformations with nonlinear activation functions in order to approximate the conditional distribution of a set of classes given vectorized input. There are many different kinds of neural networks, so this section serves to introduce only some of their basic properties.

A.1.1 Feed-forward models

As a simple example, let us consider a so-called *feed-forward neural network* and its defining characteristics.

Suppose that we have a data set $\mathcal{D}$, which consists of pairs of input vectors $\mathbf{x}_i$ and corresponding targets t_i. $\mathcal{D}$ consists of a training set $\mathcal{D}_{train}$ and a test set $\mathcal{D}_{test}$. We want a neural network to predict the labels t of the data instances. To achieve this, the network is trained on the items in $\mathcal{D}_{train}$, and tested on the unseen items in $\mathcal{D}_{test}$.

Figure A.1 shows a small feed-forward neural network, which we call $\mathcal{N}$, as an example with four-dimensional input vectors $\mathbf{x}_i$, a five-dimensional hidden layer and three output classes. If data instance $\mathbf{x}_i$ is presented to $\mathcal{N}$, the corresponding hidden layer vector $\mathbf{h}_i$ is computed as follows:

$$\mathbf{h}_i = f(\mathbf{W}_h \times \mathbf{x}_i + \mathbf{b}_h), \tag{A.1}$$

where $\mathbf{W}_h$ is the input-to-hidden weight matrix (c.q. of dimensions 5×4) and $\mathbf{b}_h$ is a bias term. f is a nonlinear activation function, e.g. tanh. Next, to move

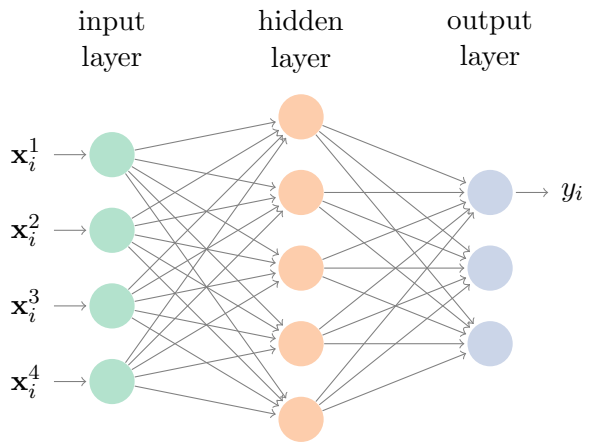


Figure A.1: Schematic visualization of the small feed-forward neural network $\mathcal{N}$.

from the hidden layer to the output layer, a similar computation is performed on $\mathbf{h}_i$:

$$\mathbf{y}_i = \mathbf{W}_o \times \mathbf{h}_i + \mathbf{b}_o, \tag{A.2}$$

where $\mathbf{y}_i$ denotes the output vector, which is three-dimensional for this three-class classification problem. $\mathbf{W}_o$ is the hidden-to-output weight matrix (in this example of dimensions 3×5) and $\mathbf{b}_o$ is a bias term. It is common to apply a softmax function to $\mathbf{y}_i$ in order to represent the output as a probability distribution. Generally, the softmax of $\mathbf{y}_i^j$, the jth entry of $\mathbf{y}_i$, is given by:

$$\text{softmax}(\mathbf{y}_i^j) = \frac{e^{\mathbf{y}_i^j}}{\sum_{j'} e^{\mathbf{y}_i^{j'}}} \tag{A.3}$$

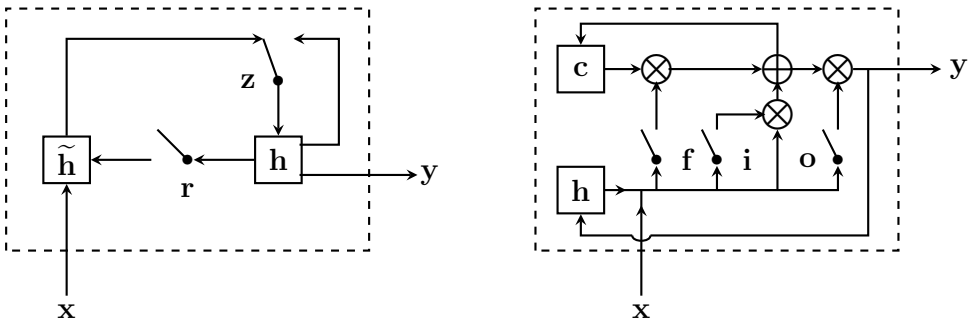
The index of the entry of $\mathbf{y}_i$ with the highest softmax value can then be returned as the predicted label y_i for input $\mathbf{x}_i$:

$$y_i = \underset{j}{\text{argmax}}(\text{softmax}(\mathbf{y}_i^j)) \tag{A.4}$$

In the example of Figure A.1, the upper node of the output layer yields the highest softmax value and is therefore returned as the predicted label.

A.1.2 Recurrent models

A variation on the feed-forward architecture discussed so far is the class of *recurrent networks*. Recurrent networks are neural models with feedback connections, where output at one time step serves as input at the next. They are especially suitable for processing sequential information. In this thesis we use two kind kinds of recurrent units: the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM).



(a) GRU (b) LSTM

Figure A.2: Schematic visualization of two recurrent units. Switch signs represent gates, $\otimes$ element-wise product and $\oplus$ summation. $\mathbf{x}$ and $\mathbf{y}$ are input and output vectors.

Gated Recurrent Unit The Gated Recurrent Unit (GRU) is a recurrent cell introduced by Cho et al. (2014). It is schematized in Figure A.2a. The switch signs in the figure represent internal states known as 'gates'. Individually, all gates are just regular feed-forward neurons that apply a nonlinearity to a weighted sum of the input vectors. In the GRU case, $\mathbf{r}$ is a reset gate used to compute the h-dimensional preliminary hidden state $\widetilde{\mathbf{h}}_t$ at time t:

$$\widetilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{A.5}$$

For d-dimensional input $\mathbf{x}_t$ at time t, $\mathbf{W}_h$ is the $h \times d$-dimensional weight matrix to the candidate hidden state. $\mathbf{U}_h$ is the $h \times h$-dimensional matrix storing the internal hidden weights, $\mathbf{r}_t$ is the reset gate value at time t, $\mathbf{h}_{t-1}$ is the previous hidden state and $\mathbf{b}_h$ an h-dimensional bias vector. $\otimes$ is the element-wise product, also known as the 'Hadamard product'. $\mathbf{r}$ is called the 'reset gate' because it determines how much of the previous hidden state is remembered and used in computation. At time t it takes the following value:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \tag{A.6}$$

$\mathbf{W}_r$ are $\mathbf{U}_r$ are weight matrices of dimensionality $h \times d$ and $h \times h$, respectively. $\mathbf{b}_r$ is another bias term. σ is the logistic sigmoid function that is used as nonlinear activation at the gates. The value of gate $\mathbf{z}$, the update gate, at time t is the result of a similar computation:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{A.7}$$

$\mathbf{W}_r$ and $\mathbf{U}_r$ are weight matrices of the same dimensions as before, $\mathbf{b}_z$ is a new bias term. The gate value $\mathbf{z}_t$ establishes the extent to which the previous hidden state $\mathbf{h}_{t-1}$ is retained, or updated with $\widetilde{\mathbf{h}}_t$:

$$\mathbf{h}_t = \mathbf{z}_t \otimes \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \otimes \widetilde{\mathbf{h}}_t \tag{A.8}$$

Long Short-Term Memory The LSTM (shown in Figure A.2b) is a predecessor of the GRU, introduced by Hochreiter and Schmidhuber (1997). It has more gates, and therefore more parameters, than the GRU. To be precise, it has a forget gate $\mathbf{f}$, an input gate $\mathbf{i}$ and an output gate $\mathbf{o}$. It also has an extra component $\mathbf{c}$, which serves as memory cell. As with the GRU, all gates are traditional neurons, which apply the logistic sigmoid function σ to a linear transformation of the input vectors. At time t, they take the following values:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{A.9}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{A.10}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{A.11}$$

As before, $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ denote the input vector at time t and the hidden state at time $t-1$. For d-dimensional input and h hidden units, weight matrices $\mathbf{W}_f, \mathbf{W}_i$ and $\mathbf{W}_o$ are $h \times d$-dimensional, and internal hidden-to-hidden matrices $\mathbf{U}_f, \mathbf{U}_i$ and $\mathbf{U}_o$ are $h \times h$-dimensional. $\mathbf{b}_f, \mathbf{b}_i$ and $\mathbf{b}_o$ are h-dimensional bias vectors for the three gates. The memory cell $\mathbf{c}$ is computed at time t as:

$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c), \tag{A.12}$$

with $\mathbf{W}_c$ an $h \times d$- and $\mathbf{U}_c$ an $h \times h$-dimensional weight matrix, and $\mathbf{b}_c$ a bias vector of length h. The new hidden state is now computed as:

$$\mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t) \tag{A.13}$$

A.2 Learning

In order for a neural network to model the distribution of a data set, its weight parameters must be optimized. This is done by means of a *learning algorithm*, which specifies how to update the network parameters in a way that gradually minimizes a notion of loss, or maximizes a notion of reward.

A.2.1 Backpropagation

The most common learning algorithm for neural models is a method called *backpropagation*. This is also the method that we use when training the models in this thesis.

We illustrate backpropagation relative to the small feed-forward network visualized in Figure A.1. Here, the aim is to predict the target value of the input, i.e. $y_i = t_i$. In order to achieve this, the parameters in the (randomly initialized) weight

matrices $\mathbf{W}_h$, $\mathbf{W}_o$ and bias vectors $\mathbf{b}_h$, $\mathbf{b}_o$ must be optimized. In backpropagation, this is done by differentiating an objective function with respect to each of the network parameters. At every update, the weights are incremented with a value that should move them closer to the point where the objective function reaches an optimum.

Objective function The objective function, also known as the loss function, expresses the deviance between the output of the network $\mathcal{N}$ and the training target.

A popular objective function is the negative log likelihood (NLL). Let $\mathbf{z}_i$ denote the vector containing the softmax output for each class (i.e. $\mathbf{z}_i^j = \text{softmax}(\mathbf{y}_i^j)$ for each index j). For target t_i, let $\mathbf{t}_i$ denote the corresponding target vector, which is a one-hot vector with the 1-entry at the index of the correct class. Then the NLL loss is given by:

$$E(\mathbf{z}_i, \mathbf{t}_i) = -\sum_j \log(\mathbf{z}_i^j)\mathbf{t}_i^j \tag{A.14}$$

Additionally, L2-regularization can be applied to prevent overfitting. This is the phenomenon occurring when a model fits the training data too closely to be capable of successful generalization to unseen data. A model is at particular risk of overfitting when its learned parameter values become excessively high. L2-regularization seeks to prevent this effect by penalizing large coefficients. It does so by extending the loss function with a weight decay term:

$$E_r(\mathbf{z}_i, \mathbf{t}_i) = E(\mathbf{z}_i, \mathbf{t}_i) + \lambda \sum_j w_j^2 \tag{A.15}$$

Here, $E(\mathbf{z}_i, \mathbf{t}_i)$ is a loss function, c.q. the NLL of Equation (A.14). The quantities w_j represent the model weights and λ is the L2 penalty term. The higher λ, the more cost is associated with large parameter values.

Optimization During the training phase, the aim is to minimize the error as computed by Equation (A.15) for any output-target pair $\mathbf{z}_i, \mathbf{t}_i$. This is done by using the outcome of the objective function as an error signal for parameter updating with a differential optimization algorithm. Stochastic Gradient Descent (SGD) is the most basic such method, which updates the network weights w_i as per the following equation:

$$w_i^{n+1} = w_i^n - \eta \nabla_{w_i} E_r, \tag{A.16}$$

where w_i^n represents the value of parameter w_i at time n, η is the learning rate and $\nabla_{w_i} E_r$ is the gradient of the objective function E_r with respect to w_i. During training, the network iterates over the training set several times. One such iteration is called an 'epoch', and starts with reshuffling the data instances. At testing time the learned parameters are frozen.

A.2.2 Reinforcement learning

So far, in this section we assumed that for any input, there is a fixed target value. This allows us to compute the loss by comparing the predicted value with the gold value. This approach is known as *supervised learning*. In this thesis, we apply it when we use behavioral cloning to imitate fully labelled expert demonstrations.

In practice, training samples are not always neatly formatted as input-output pairs. It can be the case that labelling individual samples with a target value is not the most suitable approach for a problem, or that doing so requires excessive amounts of manual labour. In such situations, supervised learning is not a good or available option. Instead, one can use *reinforcement learning*.

Reinforcement learning is especially suitable for problems that agents have to solve in more than one time step, such as sequential decision making problems. This is the case in BabyAI, for which reason we use reinforcement learning to train the initial expert agents at the individual levels. In such multi-step problems, instead of immediately evaluating the actions taken at each moment, an over-all metric of success can be computed at the end of a sequence of actions. From this metric, a reward for the indvidual actions follows, which serves as the learning signal. The lack of pre-established supervision per time step enables reinforcement learning agents to freely develop their own strategy to solve a sequential problem by maximizing the expected reward. Such a strategy is called a *policy*.

Policy gradients Many reinforcement learning methods use *policy gradients*, which estimate the gradient for a policy so that stochastic gradient methods can be used to approximate the optimal parameter configuration. Instead of minimizing a loss function, we now maximize an expected reward function. If π represents a policy (assigning probabilities to trajectories), τ a trajectory (a sequence of performed actions) and r a reward function (assigning rewards to trajectories), then the expected reward $R(\theta)$ for model parameterization θ can be expressed as

$$R(\theta) = \mathbb{E}_\pi[r(\tau)] \tag{A.17}$$

In order to maximize the expected reward $R(\theta)$ by means of a stochastic gradient method, we have to compute the gradient $\nabla_\theta R(\theta)$. To do so, we can apply the following identities:

$$\begin{aligned}
\nabla_\theta R(\theta) &= \nabla_\theta \mathbb{E}_\pi[r(\tau)] && \text{(A.18)}\\
&= \nabla_\theta \sum_\tau \pi(\tau) r(\tau) && \text{(A.19)}\\
&= \sum_\tau \nabla_\theta \pi(\tau) r(\tau) && \text{(A.20)}\\
&= \sum_\tau \pi(\tau) \nabla_\theta \log \pi(\tau) r(\tau)^1 && \text{(A.21)}
\end{aligned}$$

$$= \mathbb{E}_\pi[r(\tau)\nabla_\theta \log \pi(\tau)] \tag{A.22}$$

The equation resulting from this derivation, $\nabla_\theta R(\theta) = \mathbb{E}_\pi[r(\tau)\nabla_\theta \log \pi(\tau)]$, is known as the *policy gradient theorem*. It states that the gradient of the expected reward equals the expected value for the product of the reward and the log gradient of the policy. Computationally, this is a major simplification. In practice, the expectation $\mathbb{E}_\pi[r(\tau)\nabla_\theta \log \pi(\tau)]$ is often approximated by sampling trajectories.

Model parameters are updated using gradient *ascent* instead of *descent*, because instead of minimizing a loss function we want to maximize the expected reward. This means that an individual model weight w_i is now updated as follows:

$$w_i^{n+1} = w_i^n + \eta \nabla_{w_i} R, \tag{A.25}$$

where w_i^n represents the value of parameter w_i at time n, η is the learning rate and $\nabla_{w_i} R$ is the gradient of the reward function R with respect to w_i.

Proximal policy optimization Following the policy gradient theorem, the reward function $R(\theta)$ can be interpreted as

$$R(\theta) = \mathbb{E}_\pi[r(\tau) \log \pi(\tau)] \tag{A.26}$$

To train the reinforcement learning experts used in this thesis, we do not use this reward function, but the following, more sophisticated variation that comes from *proximal policy optimization* (Schulman et al., 2017):

$$R^{PPO}(\theta) = \mathbb{E}_\pi[\min(r(\tau)\frac{\pi(\tau)}{\pi_{\text{old}}(\tau)}, r(\tau)\text{clip}(\frac{\pi(\tau)}{\pi_{\text{old}}(\tau)}, 1-\epsilon, 1+\epsilon))] \tag{A.27}$$

Here ϵ is a new hyperparameter and π_{old} denotes the policy of the previous time step. Thus, maximization of the term $r(\tau)\frac{\pi(\tau)}{\pi_{\text{old}}(\tau)}$ induces large policy updates. In order for these updates not to become excessive, there is the second term, $r(\tau)\text{clip}(\frac{\pi(\tau)}{\pi_{\text{old}}(\tau)}, 1-\epsilon, 1+\epsilon)$. It clips the ratio between the new and the old policy, effectively restricting it to the interval $[1-\epsilon, 1+\epsilon]$. The minimum guarantees that $R^{PPO}(\theta)$ gives a lower bound on $r(\tau)\frac{\pi(\tau)}{\pi_{\text{old}}(\tau)}$. In practice, it turns out that this makes proximal policy optimization outperform other policy gradient algorithms in terms of sample complexity and convergence rate.

[1]The step from Equation A.20 to A.21 uses the chain rule to apply the following trick:

$$\pi(\tau)\nabla_\theta \log \pi(\tau) = \pi(\tau)\frac{1}{\pi(\tau)}\nabla_\theta \pi(\tau) \tag{A.23}$$

$$= \nabla_\theta \pi(\tau) \tag{A.24}$$

Appendix B

Hyperparameter settings

word embedding size	256
observation CNN output size	256
memory LSTM hidden size	2048
instruction GRU hidden size	256
guidance encoder GRU hidden size	512
guidance message length	2
guidance vocabulary size	3
batch size	128 (GoToObj, GoTo), 256 (GoToObjMaze), 512 (otherwise)
learning rate	$1 \cdot 10^{-4}$
entropy coefficient	$1 \cdot 10^{-2}$
optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 \cdot 10^{-5}$)
dropout probability	0.5
training set size	1K (GoToObj), 25K (GoToObjMaze), 50K (GoToLocal), 100K (otherwise)
validation set size	500

Table B.1: Hyperparameter settings used in experiments.

Bibliography

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Karl J Astrom. 1965. Optimal control of markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1):174–205.

John Langshaw Austin. 1962. *How to do things with words*, 2 edition. Oxford University Press.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. 2018. Learning to follow language instructions with adversarial reward induction. *arXiv preprint arXiv:1806.01946*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville. 2017. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*.

Angelo Cangelosi, Emmanouil Hourdakis, and Vadim Tikhanoff. 2006. Language acquisition and symbol grounding transfer with neural networks and cognitive robots. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1576–1582. IEEE.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. 2018. Gated-attention architectures for task-oriented language grounding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

David L Chen and Raymond J Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. 2016. Pomdp-lite for robust robot planning under uncertainty. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5427–5433. IEEE.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: First steps towards grounded language learning with a human in the loop. *arXiv preprint arXiv:1810.08272*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 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)*.

Herbert H Clark. 1996. *Using language*. Cambridge University Press.

John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. 2018. Guiding policies with language via meta-learning. *arXiv preprint arXiv:1811.07882*.

Vincent Crawford. 1998. A survey of experiments on communication via cheap talk. *Journal of Economic Theory*, 78(2):286–298.

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471.

Serhii Havrylov and Ivan Titov. 2017. Emergence of language with multi-agent games: Learning to communicate with sequences of symbols. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2149–2159. Curran Associates, Inc.

Sachithra Hemachandra, Felix Duvallet, Thomas M Howard, Nicholas Roy, Anthony Stentz, and Matthew R Walter. 2015. Learning models for following natural language directions in unknown environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5608–5615. IEEE.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, et al. 2017. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*.

Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc.

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

Ronald A Howard. 1960. *Dynamic Programming and Markov Processes*. John Wiley.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *Proceedings of the International Conference on Learning Representations*.

Emilio Jorge, Mikael Kågebäck, and Emil Gustavsson. 2016. Learning to play guess who? and inventing a grounded language as a consequence. *CoRR*, abs/1611.03218.

Faisal Khan, Bilge Mutlu, and Jerry Zhu. 2011. How do humans teach: On curriculum learning and teaching dimension. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1449–1457. Curran Associates, Inc.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kai A Krueger and Peter Dayan. 2009. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394.

M Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197.

Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. 2017. Multi-agent cooperation and the emergence of (natural) language. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

David Lewis. 1969. *Convention: a philosophical study*. Harvard University Press.

Ryan Lowe, Jakob Foerster, Y-Lan Boureau, Joelle Pineau, and Yann Dauphin. 2019. On the pitfalls of measuring emergent communication.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Tomas Mikolov, Armand Joulin, and Marco Baroni. 2016. A roadmap towards machine intelligence. In *Computational Linguistics and Intelligent Text Processing - 17th International Conference, CICLing 2016, Konya, Turkey, April 3-9, 2016, Revised Selected Papers, Part I*, pages 29–61.

Igor Mordatch and Pieter Abbeel. 2017. Emergence of grounded compositional language in multi-agent populations.

Mathijs Mul. 2018. *Recognizing Logical Entailment: Reasoning with Recursive and Recurrent Neural Networks*. MSc Thesis, University of Amsterdam.

Andrew Y. Ng and Stuart Russell. 2000. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann.

Joni Pajarinen and Ville Kyrki. 2017. Robotic manipulation of multiple objects as a pomdp. *Artificial Intelligence*, 247:213–228.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

D. A. Pomerleau. 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97.

Stéphane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 661–668. JMLR.org.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

Stuart Russell. 1998. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 101–103. ACM Press.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

John Rogers Searle. 1969. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press.

Brian Skyrms. 2010. *Signals: Evolution, Learning, and Information*.

Sainbayar Sukhbaatar, arthur szlam, and Rob Fergus. 2016. Learning multiagent communication with backpropagation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2244–2252. Curran Associates, Inc.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Naftali Tishby, Fernando C Pereira, and William Bialek. 1999. The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing*. University of Illinois Press.

Kyle Wagner, James A. Reggia, Juan Uriagereka, and Gerald S. Wilkinson. 2003. Progress in the simulation of emergent communication and language. *Adaptive Behavior*, 11(1):37–69.

Sida I Wang, Percy Liang, and Christopher D Manning. 2016. Learning language games through interaction. *arXiv preprint arXiv:1606.02447*.

Edward C Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. 2018. Learning to parse natural language to grounded reward functions with weak supervision. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE.

Terry Winograd. 1972. Understanding natural language. *Cognitive psychology*, 3(1):1–191.

Ludwig Wittgenstein. 1953. Philosophical investigations.

Michael J. Wooldridge. 2009. *An Introduction to MultiAgent Systems*, 2nd edition. Wiley.

Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. 2018. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*.

Haonan Yu, Haichao Zhang, and Wei Xu. 2018. Interactive grounded language acquisition and generalization in a 2d world. *arXiv preprint arXiv:1802.01433*.

Wojciech Zaremba and Ilya Sutskever. 2014. Learning to execute. *arXiv preprint arXiv:1410.4615*.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA.