Evolving Recurrent Neural Networks for Emergent Communication

by

Joshua Sirota

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Emergent communication is a framework for machine language acquisition that has recently been utilized to train deep neural networks to develop shared languages from scratch and use these languages to communicate and cooperate. Previous work on emergent communication has utilized gradient-based learning. Recent advances in gradient-free evolutionary computation, though, provide an alternative approach for training deep neural networks which could be beneficial for emergent communication. Certain evolutionary algorithms have been shown to be robust to misleading gradients, which can present a problem in cooperative communication tasks. Additionally, some evolutionary algorithms have been shown to train quickly and require only CPUs, rather than the GPUs needed for gradient-based training.

This thesis addresses the question of whether or not a gradient-free evolutionary approach can be used as a training methodology for emergent communication amongst deep neural networks. The evolutionary approach that we use consists of a genetic algorithm to search for both the weights and architectures of these networks. We adapt evolutionary techniques which have previously been used to evolve individual agents so as to co-evolve pairs of agents which develop languages to play a repeated referential game. We empirically demonstrate that agents trained solely with evolution perform well above a random chance baseline, although our performance is worse than that previously achieved with gradient-based reinforcement learning. We show that evolving the architecture of these agents can improve their ability to perform cooperative communication-based tasks when compared to utilization of a fixed architecture. The main contribution of this thesis is to show that an evolutionary approach can be used to train agents to communicate and suggests that these techniques could be useful for future research on cooperative multi-agent problems involving deep neural networks.

Preface

This research was conducted in a collaborative manner by myself, Dr. Vadim Bulitko, Dr. Matthew R. G. Brown, and Sergio Poo hernandez. I led the development of this research. All development and data analysis was performed by me personally. Writing and editing was shared between myself, Dr. Vadim Bulitko, Dr. Matthew R.G. Brown, and Sergio Poo Hernandez. This work has been previously published (Sirota et al., 2019; Sirota et al., 2019).

"Yes, that's so," said Sam. "And we shouldn't be here at all, if we'd known more about it before we started. But I suppose it's often that way. The brave things in the old tales and songs, Mr. Frodo: adventures, as I used to call them. I used to think that they were things the wonderful folk of the stories went out and looked for, because they wanted them, because they were exciting and life was a bit dull, a kind of sport, as you might say. But that's not the way of it with the tales that really mattered, or the ones that stay in the mind. Folk seem to have been just landed in them, usually - their paths were laid that way, as you put it. But I expect they had lots of chances, like us, of turning back, only they didn't. And if they had, we shouldn't know, because they'd have been forgotten. We hear about those as just went on - and not all to a good end, mind you; at least not to what folk inside a story and not outside it call a good end. You know, coming home and finding things all right, though not quite the same - like old Mr. Bilbo. But those aren't always the best tales to hear, though they may be the best tales to get landed in! I wonder what sort of tale we've fallen into?"

Acknowledgements

I would like to express my utmost gratitude to my supervisors, Dr. Vadim Bulitko and Dr. Matthew R. G. Brown. Dr. Bulitko spent countless hours teaching me how to be a researcher. This included guidance and support throughout my journey as a graduate student; many hours spent digging into problems as they arose; helping me become a better writer; and always providing an intriguing point of view. Dr. Brown consistently provided different angles for viewing and formulating problems, and helped greatly by editing papers, as well as this manuscript.

I thank Dr. Osmar Zaiane for being a member of my examining committee and for providing excellent feedback. I thank Dr. Martin Müller for acting as my examination chair.

I thank Everton Schumacker-Soares, Sergio Poo Hernandez, Kacy Doucet, Devon Sigurdson, Morgan Cselinacz, and Colter Macdonald from the AGI lab for discussions and help during our time working together.

I thank the Natural Sciences and Engineering Research Council for financial support, and Compute Canada for providing the computational resources on which all of my experiments were run.

I thank my father, Perry Sirota; my sister, Rebecca Sirota; and my friends. We are the average of the people whom we choose to spend our time with, and it is to this that I attribute the best parts of myself.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the fundamental goals of artificial intelligence research is the development of artificial general intelligence, that is, agents which can learn many different tasks and exhibit human-like cognition. One aspect of artificial general intelligence is the development of language, which has been of central importance to the intellectual and social development of humans and some non-human animals. Without the ability to develop shared languages, acquired knowledge not easily be shared. Additionally, a lack of language could make any non-trivial cooperation on a task infeasible. Equipping machines with the capacity to develop their own languages and use these languages to communicate and cooperate could improve the efficacy of these machines by allowing them to share knowledge and work effectively in groups. One possible framework for machine language acquisition is *emergent communication*, which is a framework through which which machines develop languages from scratch in response to some task or environment and use these languages to communicate. By developing emergent communication, we can provide machines with the capacity to communicate, which has been invaluable for human success. Equipping machines with the capacity to communicate benefits us, as improving the performance of machines further elevates their ability to solve difficult problems for us.

Emergent communication could be used to solve certain challenges surrounding deep neural networks. As networks have become increasingly complex, they have also become less interpretable. Training these networks to communicate could solve this interpretability problem. Just as humans use language to transform complex, abstract internal reasoning into a form which can be understood by others, emergent communication could force deep neural networks to distill their representations and reasoning into an interpretable form. As such, emergent communication could provide us with the tools to better understanding deep neural networks so that we can further improve their performance at various tasks, as well as monitor their behaviour.

Recent groundbreaking work (Foerster et al., 2016; Lazaridou et al., 2016; Havrylov and Titov, 2017; Mordatch and Abbeel, 2018; Lazaridou et al., 2018; Mordatch and Abbeel, 2017; Lowe et al., 2017; Cao et al., 2018) has shown emergent communication to be a promising framework for language acquisition between deep neural networks. The languages developed when deep neural networks are trained on emergent communication tasks have been shown to be *grounded*. That is, they are related to the communicating agent or the environment in which it is operating (Wagner et al., 2003). In this sense, the words used have meaning and the agents share an understanding of what these meanings are. It is this grounding that mirrors human language and makes emergent communication promising for artificial general intelligence.

All contemporary work on emergent communication amongst deep neural networks has utilized gradient-based learning. Gradient-free evolutionary computation (Collobert and Weston, 2008; Back, 1996; Fogel, 2006), though, is a promising alternative training approach for emergent communication. Lowe et al. (2017) described some difficulties that arise when current gradient-based approaches are used for cooperative communication tasks. One such problem is misleading gradient signals, particularly when the task at hand involves long time horizons. An example given by Lowe et al. (2017) is one in which an agent is emitting useful messages which are being ignored by its partner. Despite learning to communicate useful information, this agent will be poorly rewarded. We believe that a gradient-free approach could address this difficulty. Rather than following potentially misleading gradients, evolutionary approaches sample many policies in a given region of policy space. This sampling has been shown to make these approaches more robust to tasks which involve deceptive gradients (Such et al., 2017). As such, the deceptive gradients which are present in cooperative communication tasks could present less of a problem when using an evolutionary approach.

In addition to their robustness when faced with tasks which exhibit deceptive gradients, evolutionary algorithms have a number of properties which are desirable for cooperative communication tasks. Certain evolutionary algorithms have been shown to have fast runtimes and require only CPUs to train on, rather than GPUs, which are typically needed for gradient-based learning (Salimans et al., 2017; Such et al., 2017). The fast runtimes seen with evolutionary computation are due to dispensing with expensive gradient computations during training. In addition to executing quickly, these algorithms are also highly and easily parralelizable, with threads often being required to communicate only a single real number rather than more complicated information such as gradients, agent parameters, and parameter updates (Salimans et al., 2017; Such et al., 2017).

Despite the attractive properties of evolutionary computation for application to emergent communication, there has been no recent work in this direction. In this thesis we take a first step forward in this line of research. In particular, we explore whether or not a simple genetic algorithm can be used to train deep neural networks to develop shared languages from scratch and use these languages to cooperate on a referential game. Furthermore, previous research in emergent communication amongst deep neural networks used fixed, handcrafted neural networks. We explore whether or not the architecture of communicating agents can be evolved, and if these evolved agents can outperform the well-known Gated Recurrent Unit (Cho et al., 2014) network architecture.

1.1 Contributions

This thesis makes the following contributions:

- We adapt contemporary evolutionary computation techniques for training deep neural networks to the case of pairs of agents creating a shared language from scratch and using this language to cooperate at a repeated referential game. We show that the performance of these agents is well above random baseline but worse than that previously achieved using gradient-based deep reinforcement learning. We show that the emerged languages show natural language properties consistent with previous work in emergent communication. These results suggest that gradient-free evolutionary computation is a viable alternative or complement to gradient-based learning for future research in emergent communication.

- We evolve the recurrent architectures of communicating agents. We show agents with evolved architectures can outperform agents with fixed architectures, but not a Gated Recurrent Unit. This suggests that evolving the architectures of communicating agents could be beneficial for further research in emergent communication.

1.2 Thesis organization

This thesis is organized as follows. Chapter 2 precisely formulates the problem which we address. Chapter 3 reviews related work on emergent communication. Chapter 4 describes our adaptation of evolutionary computation to the problem of emergent communication. Chapter 5 presents results of my experiments and a discussion of these results, as well as details about training and hyperparameter selection. Chapter 6 describes possible extensions and modifications to our approach for future work. Chapter 7 concludes this thesis by summarizing the problem being solved, our contribution to that problem, and the implications of this research.

Chapter 2

Problem Formulation

In this chapter we state the problem which we address with this thesis and provide intuitive motivation for the environment in which we address this problem. We then formally define the problem and provide a list of preferences for our solution to this problem.

2.1 Intuitive Motivation

The problem that we address in this thesis is generation of deep neural networks which develop their own languages in an emergent fashion so as to cooperate with each other. In line with previous work (Lazaridou et al., 2018), we evaluate the communication of our agents based on their performance at a repeated referential game. The repeated referential game that we use is one in which, on a given round, a listener agent is shown a set of $k + 1$ objects, in which one object is a *target* and k objects are *distractors*. A speaker agent is shown only the target and then sends a message to the listener which the listener uses to choose an object from the set containing the target and distractors. The speaker and listener must learn a language of such messages so that the speaker's message contains sufficient information to help the listener correctly choose the target, rather than a distractor. Figure 2.1 shows a single round of the referential game with $k = 3$ distractors.

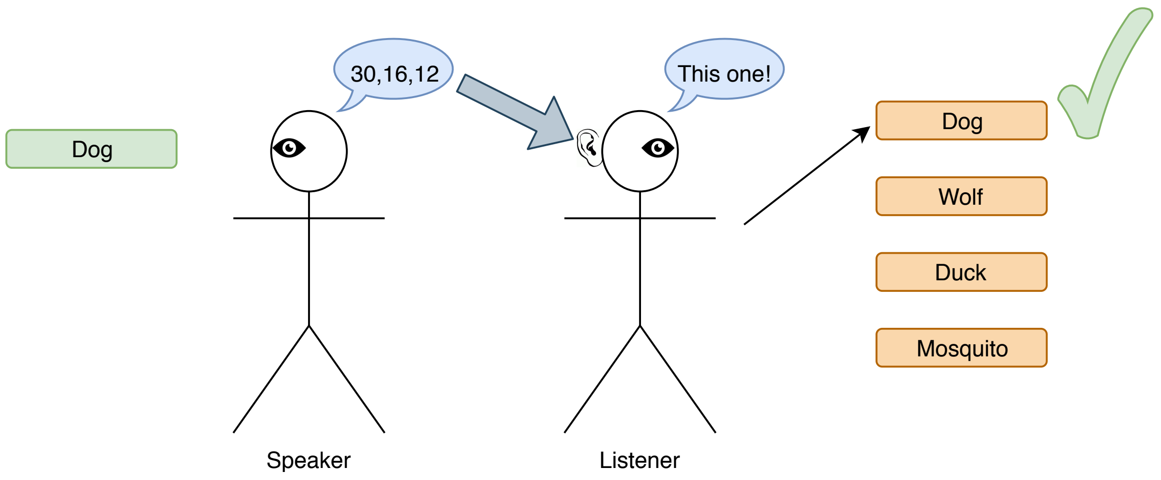

Figure 2.1: A speaker and listener playing the referential game. The speaker is shown concept *dog* and produces message "30,16,12" which the listener then hears. The listener sees target *dog* as well as distractors *wolf*, *duck*, and *mosquito*, then uses the speaker's message to correctly choose *dog*.

We believe that a referential game is a useful setting for training agents to communicate. Lazaridou et al. (2018) showed that a repeated referential game is sufficient for agents to develop complex languages which show natural-language linguistic properties such as compositionality (Vogt, 2005) and ambiguity (Piantadosi et al., 2012).

Another useful aspect of the referential game is that we can control the difficulty of the game by selecting k - the number of distractors used. This modifies the level of specificity which agents must capture with their language. For example, suppose that our dataset which agents communicate about consisted only of animals. This set could contain n concepts such as dog, wolf, duck, goose, and iguana. If we set $k = n - 1$, that is, if every round the listener had to choose the target out from the entire dataset, this would be a particularly difficult task. One reason for this is that the speaker's messages would have to distinguish dog from wolf, or duck from goose. Given that these concepts are very similar to each other, the level of specifity that the language would need to capture is very high. Conversely, if we set k to 1, then each round we sample the target and a single distractor for the agents to communicate about. In this case, the task can become significantly simpler. Distinguishing dog from duck, and bear from mosquito should be simpler. On some rounds, the agents would still have to distinguish similar words from each other, but this should not happen too often with sufficiently small k. A similar effect can be seen with human language.

As an example of how the need for increased specificity changes in response to distractors, consider the case when an artificial intelligence expert is asked about their field of work by someone unfamiliar with AI. In this case, the expert can simply state that they work in artificial intelligence. This response will be meaningful to the inquirer, as the other fields that the inquirer will compare the answer to (the inquirer's distractors) will be unrelated to AI. Conversely, when asked about their field of work by a fellow AI expert, responding with artificial intelligence will no longer be informative. Rather, more specificity will be needed (e.g., reinforcement learning, deep learning, heuristic search). This is because artificial intelligence has many meanings to a fellow expert, and so their set of AI-related distractors is large.

2.2 The Referential Game

We now present in detail the repeated referential game which we use and we define the metric by which we will evaluate agent performance. The repeated referential game is shown as Algorithm 1. The game is played by a pair of agents - a speaker A_S and listener A_L. A set V is chosen containing all objects which the agent pair may communicate about. The number of rounds T is chosen along with a maximum speaker message length L. Additionally, we choose the number of distractors k.

We initialize the agent pair's shared reward r to 0 (line 1). The referential game proceeds over T rounds. First, on round ρ, a target v_ρ is chosen from V (line 3). This is the object that the speaker must communicate to the listener. In Figure 2.1, v_ρ is *Dog*. Additionally, distractors $D = \{d_\rho^1, d_\rho^2, ..., d_\rho^k\}$ are chosen randomly without replacement from $V \setminus \{v_\rho\}$ (wolf, duck, and mosquito in Figure 2.1). We set m_0 to the sentence-start token $\mathcal{S}$ (line 5). The speaker then generates its message to be transmitted (lines 7-9). The speaker first generates symbol $m_1 = A_S(\mathcal{S})$ (line 8). The speaker then processes m_1 and generates symbol

Algorithm 1: `Referential Game`

```
   input  : agents $A_S, A_L$; game objects $V$; number of referential game rounds $T$;
            maximum message length $L$; number of distractors $k$
   output: proportion of correct guesses $\frac{r}{T}$
 1 $r \leftarrow 0$
 2 for $\rho = 1, \dots, T$ do
 3     randomly choose a target $v_\rho$ from $V$
 4     choose distractors $D = \{d_\rho^1, d_\rho^2, ..., d_\rho^k\}$ from $V \setminus \{v_\rho\}$ randomly without
         replacement
 5     $m_0 \leftarrow \mathcal{S}$
 6     $i \leftarrow 1$
 7     while $m_{i-1} \neq \mathcal{E}$ & $|\vec{m}| < L$ do
 8         $m_i \leftarrow A_S(m_{i-1})$
 9         $i \leftarrow i + 1$
10     transmit $\vec{m} = (m_1, m_2, ..., m_M)$ to $A_L$
11     $A_L$ uses $\vec{m}$ to generate interpretation $z$
12     $A_L$ uses $z$ to choose an element from $D \cup \{v_\rho\}$
13     if $A_L$ chose $v_\rho$ then
14         $r \leftarrow r + 1$
15 return $\frac{r}{T}$
```

$m_2 = A_S(m_1)$. This continues until either the speaker generates $m_i = \mathcal{E}$, the sentence-end token, or the speaker has generated L symbols, at which point the maximum message length has been reached. At this point, the message vector $\vec{m} = (m_1, m_2, ..., m_M)$ is transmitted to the listener (line 10). In Figure 2.1, $\vec{m} = (30, 16, 12)$.

Given $\vec{m}$, the listener then processes $\vec{m}$ to generate z (line 11), which we call the listener's *interpretation* of $\vec{m}$. The listener A_L then uses z to select an element from $D \cup \{v_\rho\}$, and the pair receives a reward of 1 if the chosen element is the target, v_ρ (lines 12-14).

Our objective in this thesis is to maximize the referential-game success rate $\frac{r}{T}$ of agent pairs - that is, the ratio of correct guesses to rounds of the referential game. One concern when evaluating agents is that their languages do not generalize. We prefer agent languages which can adapt to previously unseen objects, as this adaptation implies that the communication protocol which the agents have created is structured, rather than being comprised of meaningless mappings between previously seen objects and messages. To test whether the agents' communication protocols generalize, the agents' success rate will be tested with objects which are never seen during training. We define successful communication as statistically significant performance above chance when the agents are communicating about previously unseen objects. For example, if the number of distractors k is set to 9, then the agents are communicating successfully if $\frac{r}{T} > 10\%$ with $p < 0.05$ as shown by a one-tailed z-test.

2.3 Preferences

We now give a list of preferences for our solution to the problem defined above.

- The ways in which agents process and represent data, as well as the methods which they learn to communicate, should be learned entirely from scratch through repeated play of the referential game.
- There should be no pre-training - that is, the weights of agents' neural networks should be randomized when they first begin playing the referential game, and they should only be updated as a consequence of their performance on the referential game.
- The neural networks of communicating agents should share no weights nor components with each other. To clarify this preference, suppose that we have a pair of communicating agents A_S and A_L. Then any modification to the weights or architecture of A_S must not modify the weights or architecture of A_L, and vice versa.
- Communicating agents should have no information about each other during training or testing. The only information given to them should be objects being communicated about, transmitted messages, and whether or not communication was successful.
- Solutions should run efficiently on CPUs.

Chapter 3

Related Work

In this chapter, we review previous work related to the problem stated in Chapter 2 and show how it leaves room for improvement.

Steels (2015) explored the emergence of grounded languages between artificial agents. The agents involved were robots consisting of movable cameras connected to computers which performed cognitive processing. These agents played a referential game in which the objects being referred to were physical shapes of various colours distributed over a wall. While the agents in this work learned languages from scratch, these agents utilized hard-coded algorithms for processing and representing data. These algorithms came in the form of a sensory layer which performed pre-defined computer-vision algorithms such as edge detection and segmentation. This hard-coded processing and representation violates our stated preferences for a solution.

Foerster et al. (2016) were the first to explore emergent communication amongst deep neural networks. The agents learned to communicate through repeated play of two referential game variants. The authors experimented with three algorithms. In the first algorithm, a recurrent neural network was trained using deep Q-learning (Mnih et al., 2015), with each agent having its own network. The second algorithm was similar to the first, but agents shared a single network. For the third algorithm, agents were allowed to pass real-valued messages (and so gradients) to each other during training. This made communication end-to-end differentiable across agents. The second algorithm involves parameter sharing, and the third provides agents with additional information about each other during training, both of which violate the preferences stated in Chapter 2. The first algorithm is the only one which satisfies our preferences. This algorithm, though, was unable to learn one of the referential games played and performed poorly on the other. Additionally, there was no evidence provided that learned communication generalized to previously unseen data. As such, this approach does not solve the problem that we address with this thesis.

Lazaridou et al. (2016) trained feedforward neural networks to play a referential game in which the objects being communicated about were images taken from ImageNet (Deng et al., 2009). Before being inputted to the speaker and listener, images being used were processed by a VGGnet (Simonyan and Zisserman, 2014), which had been pre-trained on ImageNet to generate a representation of reduced dimensionality. For a given round of the referential game, the sender saw both the target and distracting object and was informed of which was the target. Communication consisted of transmission of a single symbol drawn from the speaker's vocabulary. Agents were jointly trained using a policy gradient algorithm, specifically William's REINFORCE algorithm (Williams, 1992). While the communicating agents were trained from scratch, they both had identical pre-trained VGGnets which generated representations of input data in violation of our solution preferences.

To produce messages, the agents trained by Lazaridou et al. (2016) generated a categorical probability distribution and sampled a discrete symbol from that distribution. This sampling results in the communication process not being end-to-end differentiable and makes supervised training through backpropagation infeasible. To combat this, Havrylov and Titov (2017) used the Gumbel-softmax technique (Jang et al., 2016; Bengio et al., 2013) in which the speaker generates a Gumbel distribution from which a message is sampled. This makes the forward pass identical to previous work (Lazaridou et al., 2016). When performing the backward pass, though, the differentiable Gumbel distribution from which message symbols were sampled is used, making communication end-to-end differentiable. Both agents used the same pre-retrained VGGnet to pre-process images before they were used for communication. As such, this violates our preferences.

Lazaridou et al. (2018) trained agents implemented as long short term memory machines (Hochreiter and Schmidhuber, 1997b) to play a referential game using both symbolic (human-annotated descriptions of images) as well as pixel inputs. Rather than using a pre-trained VGGnet (Simonyan and Zisserman, 2014) to generate representations, though, these representations had to be learned as well. To this end, the speaker and listener were each equipped with their own pre-visual network. To pre-process symbolic and pixel inputs, respectively, a single-layer feedforward network and a convolutional network were used. These pre-visual networks were initialized randomly and jointly trained along with the speaker and listener using REINFORCE (Williams, 1992). This approach used a policy gradient algorithm (Sutton et al., 2000), requiring gradient computation during backpropagation. Backpropagation on deep neural networks does not run efficiently on CPUs. As such, this approach violates our preferences.

Mordatch and Abbeel (2018) studied communication in a 2-D grid environment containing landmarks as well as agents with colour, shape, and the ability to move and communicate by uttering symbols. These agents were given various private tasks such as moving to a certain location or instructing other agents to move to some location. While agents all had their own private goal and a private memory bank, all agents shared a single policy which was implemented as a feedforward network. These agents were trained in a supervised manner using backpropagation. Communication was made end-to-end differentiable through usage of the Gumbel-softmax estimator (Jang et al., 2016; Bengio et al., 2013). This work violates our preferences in that all agents shared a single network and all parameters of that network.

The multi-agent environment of Mordatch and Abbeel (2018) was used by Lowe et al. (2017) to train agents to cooperate and communicate. Rather than making communication end-to-end differentiable, though, Lowe et al. (2017) extended the Deep Deterministic Policy Gradient (Lillicrap et al., 2015) algorithm to the multi-agent case. To do this, the agents involved had access to extra information during training, similarly to work by Foerster et al.

(2016). While training, each agent received as input the observations, actions, and rewards of all other agents. At test time, the only information provided to a given agent was its own observation. Given that agents had access to all other agents' observations, actions, and rewards during training, this work does not satisfy our stated preferences.

Chapter 4

Our Approach

In this chapter we describe the evolutionary methods which we used to train agents to develop shared languages and play a repeated referential game. We begin with an introduction followed by a background on methods used. We then describe the algorithm that we used. The work described in this chapter was done in collaboration with Vadim Bulitko, Matthew R.G. Brown, and Sergio Poo Hernandez. My personal contribution was: design of the evolutionary algorithm; implementation of agents; processing data; writing training and testing code; training and testing agents; as well as manuscript writing.

4.1 Introduction

We use evolutionary computation to train pairs of recurrent neural networks to play a repeated referential game. These neural networks all have initially randomized weights, and some have architectures which are generated to be drawn uniformly at random from a set of possible architectures.

4.2 Background on Methods

The approach that we take in this work is based on contemporary work utilizing evolutionary computation to train deep neural networks (Such et al., 2017; Rawal and Miikkulainen, 2018).

To search for the parameters of our networks, we used a genetic algorithm similar to work by Such et al. (2017). In their work, a population of deep neural networks was instantiated with random weights. The agents were evaluated on a variety of tasks, such as playing Atari games and the MuJoCo humanoid locomotion task (Todorov et al., 2012). After each agent was evaluated, the best performing agents were selected for reproduction. The weights of these agents were mutated with stochastic noise to produce offspring. This process was repeated over generations until either a maximum generation or some pre-determined performance threshold was reached.

Our inspiration for architectural evolution came from work by Rawal and Miikkulainen (2018). While the agents in this work were trained using reinforcement learning, their architecture was randomly initialized as a small network and periodically mutated. Evolved agents were shown to outperform LSTMs (Hochreiter and Schmidhuber, 1997b) on tasks such as language modelling and music generation.

4.3 Proposed Approach

In this section, we describe our communicating agents and give a detailed description of how they play the referential game. We then describe the process through which we evolve the weights and architectures of agents.

4.3.1 Agents

In this section we describe the architecture of the agents involved and give a more detailed description of how the speaker and listener generate and process messages, respectively.

Algorithm 2: `Round` ρ `of refGame`

```
input  : recurrent neural networks $A_S, A_L$; accompanying feedforward networks
         $F_S, F_L$; set of game vectors $V$; sentence-start and sentence-end tokens $\mathcal{S}, \mathcal{E}$
output: 1 if $A_L$ correctly guesses, 0 otherwise
1  randomly choose a target $\vec{v_\rho} \in V$
2  randomly choose a set of distractors $D = \{\vec{d_\rho^1}, ..., \vec{d_\rho^k}\} \subseteq V \setminus \{\vec{v_\rho}\}$
3  $\vec{h_0} \leftarrow F_S(\vec{v_\rho})$
4  $m_0 \leftarrow \mathcal{S}$
5  $i \leftarrow 1$
6  while $m_{i-1} \neq \mathcal{E}$ & $|\vec{m}| < L$ do
7      set $\vec{x_i}$ to a one-hot vector representation of $m_{i-1}$
8      $\vec{o_i} \leftarrow A_S(\vec{x_i})$
9      $m_i \leftarrow \text{argmax}(\vec{o_i})$
10     $i \leftarrow i + 1$
11 transmit $\vec{m} = (m_1, ..., m_M)$ to $A_L$
12 $i \leftarrow 1$
13 while $i \leq M$ do
14     process $m_i$ with $A_L$
15     $i \leftarrow i + 1$
16 set $\vec{z}$ to the final hidden state of $A_L$
17 compute $F_L(\vec{v_\rho})$ and $F_L(D) = \{F_L(\vec{d_\rho^1}), ..., F_L(\vec{d_\rho^k})\}$
18 if $\vec{z} \cdot F_L(\vec{v_\rho}) > F_L(\vec{d_\rho^i}) \; \forall \; 1 \leq i \leq k$ then
19     return 1
20 else
21     return 0
```

In this work the objects about which the agents communicate, as well as the listener's message interpretation z, are vectors. As such, from this point on, for round ρ, we denote the target v_ρ and distractors $d_\rho^1, ..., d_\rho^k$ as $\vec{v_\rho}, \vec{d_\rho^1}, ..., \vec{d_\rho^k}$, respectively. Additionally, we denote the listener's interpretation as $\vec{z}$.

4.3.1.1 Agent Architectures

Speakers A_S and listeners A_L are both recurrent neural networks (RNNs) (Lipton, 2015). Given that we are using architectural evolution, an initial RNN architecture for the first generation is needed. We test two starting RNN architectures for A_S and A_L. These are the architectures that agents will be initialized to at the beginning of a simulation run. The

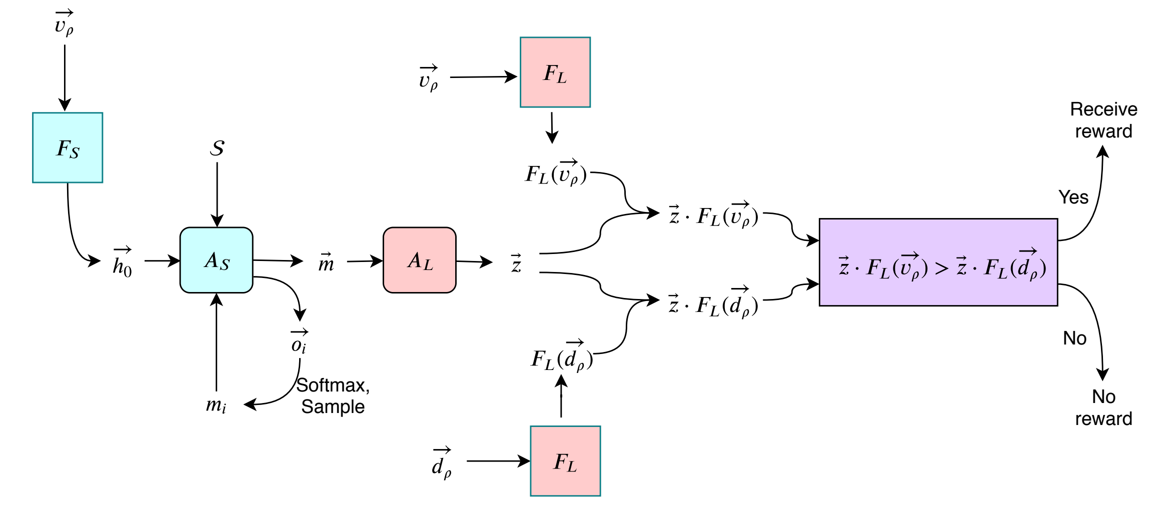

Figure 4.1: A speaker generating a message $\vec{m}$, and listener using $\vec{m}$ to choose the target out from a single distractor.

first such architecture is a gated recurrent unit (GRU) (Cho et al., 2014), and the second is an RNN which is generated to be drawn at uniformly at random from a set of possible architectures. An example of a possible randomly generated RNN is shown in a tree form in Figure 4.3. The overall structure of any of our random RNNs will be the same as that in Figure 4.3, but the operators and order of input nodes will be randomized. We chose to experiment with a GRU to explore whether evolution could further improve a well known RNN architecture's performance on a communication task. We chose the random initial architecture to test whether or not we could generate RNNs which successfully communicate from a starting architecture which has no specific engineering.

4.3.1.2 Accompanying Feedforward Networks

Speakers A_S and listeners A_L are each equipped with their own feedforward neural networks F_S and F_L respectively, which are used to preprocess target $\vec{v_\rho}$ and distractors $\{\vec{d_\rho^1}, ..., \vec{d_\rho^k}\}$ before they are input to their respective RNN, generating reduced-dimension representations of these raw inputs. Reducing these raw inputs to smaller representations serves two benefits: (i) it reduces input dimensionality to the RNNs and (ii) it allows F_S and F_L to learn

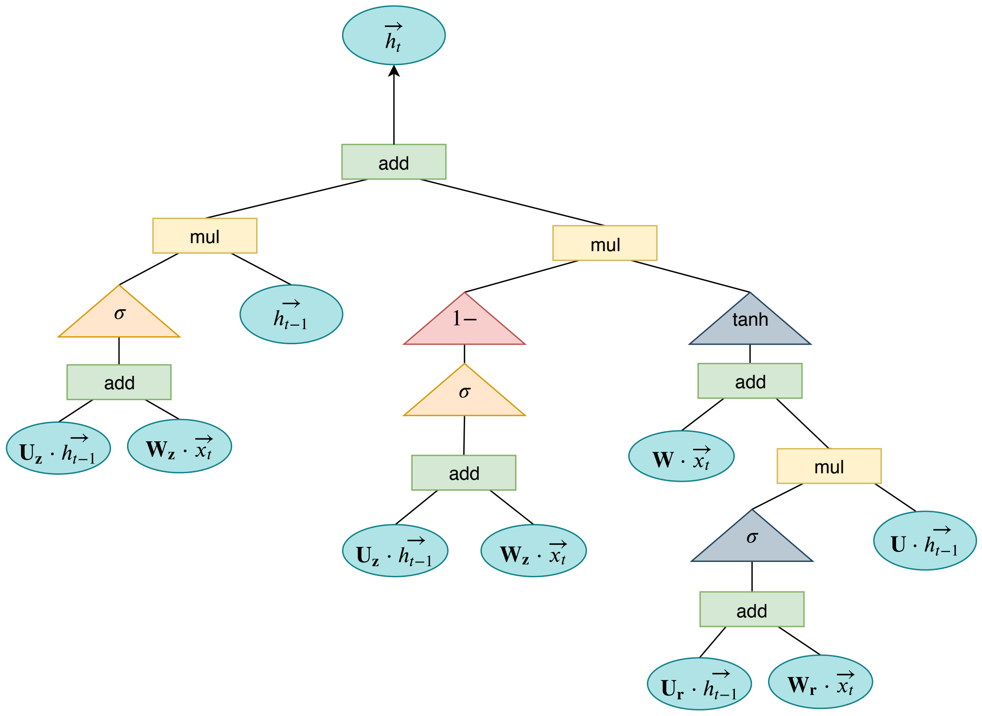
$\vec{h_t}$
add
mul
mul
σ
$\vec{h_{t-1}}$
$1-$
tanh
add
add
$\mathbf{U_z} \cdot \vec{h_{t-1}}$
$\mathbf{W_z} \cdot \vec{x_t}$
σ
$\mathbf{W} \cdot \vec{x_t}$
mul
add
σ
$\mathbf{U} \cdot \vec{h_{t-1}}$
$\mathbf{U_z} \cdot \vec{h_{t-1}}$
$\mathbf{W_z} \cdot \vec{x_t}$
add
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W_r} \cdot \vec{x_t}$

Figure 4.2: Gated Recurrent Unit tree representation.

potentially useful representations of input vectors, which may simplify the communication task of A_S and A_L. The networks F_S and F_L are both single layer neural networks with sigmoid activation.

4.3.1.3 Program Trees of Recurrent Networks

We quickly note that throughout this thesis we use the notation $a \cdot b$ flexibly. When applied to two vectors, as in $\vec{a} \cdot \vec{b}$, this denotes the dot product $\vec{a} \cdot \vec{b}^T = \sum a_1 \times b_1 +,,, + a_n \times b_n$. When applied to a matrix and vector, though, as in $\mathbf{X} \cdot \vec{a}$, this denotes the matrix-vector product of $\mathbf{X}$ with $\vec{A}$.

For the purpose of performing evolutionary search on the architecture of agents A_S and A_L, we chose to implement these RNNs as program trees (Langdon, 1999). A given program tree has internal nodes, leaf nodes, and weight matrices $\mathbf{U}, \mathbf{U_z}, \mathbf{U_r}, \mathbf{W}, \mathbf{W_z}, \mathbf{W_r}, \mathbf{V}$. When an RNN receives an input vector $\vec{x_t}$ at time t, it computes the quantities $\mathbf{U} \cdot \vec{x_t}, \mathbf{U_z} \cdot$

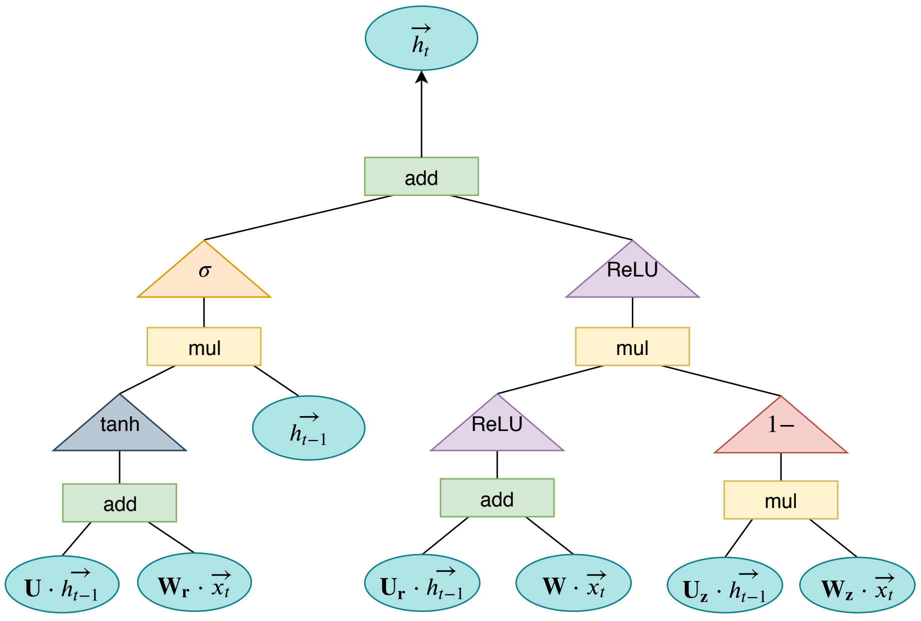
$\vec{h_t}$
add
σ
ReLU
mul
mul
tanh
$\vec{h_{t-1}}$
ReLU
$1-$
add
add
mul
$\mathbf{U} \cdot \vec{h_{t-1}}$
$\mathbf{W_r} \cdot \vec{x_t}$
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W} \cdot \vec{x_t}$
$\mathbf{U_z} \cdot \vec{h_{t-1}}$
$\mathbf{W_z} \cdot \vec{x_t}$

Figure 4.3: Example of a possible initial network architecture when using random initialization.

$\vec{x_t}, \mathbf{U_r} \cdot \vec{x_t}, \mathbf{W} \cdot \vec{h_{t-1}}, \mathbf{W_z} \cdot \vec{h_{t-1}}, \mathbf{W_r} \cdot \vec{h_{t-1}}$, where $\vec{h_{t-1}}$ is the RNN's hidden state from time $t-1$. These quantities, along with the unweighted $\vec{h_{t-1}}$, make up the inputs to the RNN's internal nodes. These internal nodes are either binary operators *add* and *multiply* or unary operators *tanh*, *ReLU*, *sigmoid* and $1-$ where $1-(\vec{a}) = \vec{1} - \vec{a}$. Internal nodes process the RNN's inputs to generate the RNN's next hidden state $\vec{h_t}$, with which we compute the RNN's output $\mathbf{V} \cdot \vec{h_t}$. We note that the seven weight matrices mentioned above are the RNN's only parameters which change throughout evolution. For any pair of internal nodes of the RNN's program tree, the connection weight between those nodes is 1. That is, when the output from some node (say a *tanh* node) passes as the input to another node (such as an *add* node), the input received by the *add* node is exactly the output from the *tanh* node. This is in contrast to the case in which the connection weight between these nodes is 2, for example, in which case the output of the *tanh* node would be multipled by 2 before being input to the *add* node.

4.3.1.4 A Detailed Round of the Referential Game

We now detail the process by which speakers generate messages and listeners use these messages to choose an object. This is essentially a single round (round ρ) from Algorithm 1 filled in with the specific details of our implementation. This process is detailed in Algorithm 2, which we now go through and explain. The process is also shown graphically in Figure 4.1. We first randomly select a target vector $\vec{v_\rho}$ from V and set of distracting vectors $D = \{\vec{d_\rho^1}, ..., \vec{d_\rho^k}\}$ from $V \setminus \{\vec{v_\rho}\}$ (lines 1,2). We generate the speaker's initial hidden state $\vec{h_0}$ by processing target $\vec{v_\rho}$ with F_S (line 3). That is, the speaker's initial hidden state is the representation of $\vec{v_\rho}$ produced by F_S. We initialize m_0 to the sentence-start token $\mathcal{S}$ (line 4). The speaker then generates a message $\vec{m}$ to be transmitted to the listener (lines 6-10). To do this, we first generate a one-hot vector representation $\vec{x_1}$ of $m_0 = \mathcal{S}$ (line 7). We process $\vec{x_1}$ with A_S and generate output vector $\vec{o_1}$, which we take the argmax of to get m_1, the first symbol of $\vec{m}$ (lines 8,9). If $m_1 \neq \mathcal{E}$, the sentence-end token, and the length $|\vec{m}|$ is less than maximum message length L, we repeat this process. This continues until either $m_M = \mathcal{E}$ for some $M < L$ or $\vec{m} = (m_1, ..., m_M)$, with $M = L$. At this point, the message $\vec{m}$ is transmitted to the listener A_L (line 11).

To choose an object from $\{\vec{v_\rho}\} \cup D$, the listener first processes $\vec{m}$ symbol by symbol, generating some final hidden state $\vec{z}$ (lines 13-16). We call $\vec{z}$ the listener's *interpretation* of $\vec{m}$. We then use the listener's feedforward network F_L to generate a representation $F_L(\vec{v_\rho})$ of the target as well as $F_L(\vec{d_\rho^i})$ for each distractor $\vec{d_\rho^i} \in D$ (line 17). We take the dot product of $\vec{z}$ with $F_L(\vec{v_\rho})$ as well as each of the $F_L(\vec{d_\rho^i})$, and the listener is said to have chosen the target if $\vec{z} \cdot F_L(\vec{v_\rho}) > \vec{z} \cdot F_L(\vec{d_\rho^i})$ for each $1 \leq i \leq k$ (lines 18,19). In this case, the pair are considered successful and receive a reward of 1. Otherwise, A_L is considered to have chosen a distractor and both agents receive a reward of 0 (line 20).

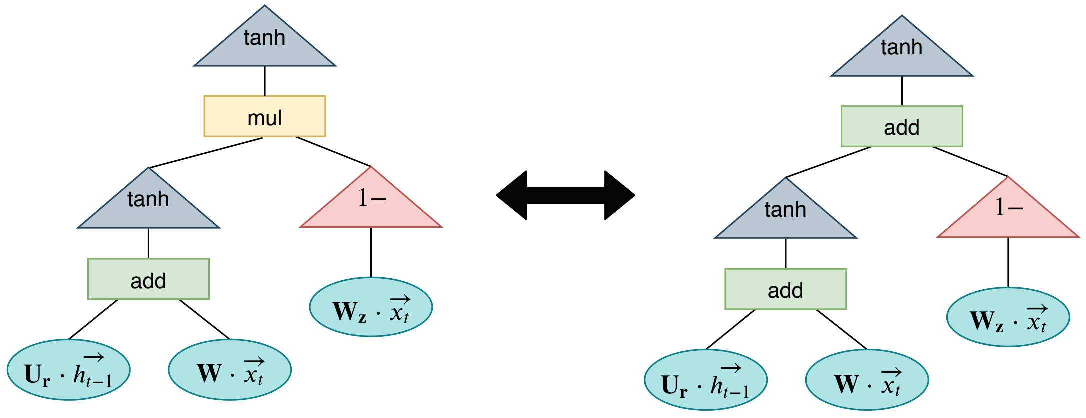
tanh
mul
tanh
1−
add
$\mathbf{W_z} \cdot \vec{x_t}$
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W} \cdot \vec{x_t}$
tanh
add
tanh
1−
add
$\mathbf{W_z} \cdot \vec{x_t}$
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W} \cdot \vec{x_t}$

Figure 4.4: Replacement mutation mechanism. Example of a mul node being replaced with an add node.

4.3.2 Evolutionary Search for Weights and Architectures

We use evolutionary search (Miikkulainen et al., 2017; Such et al., 2017) to procedurally generate the architecture of A_S and A_L as well as the weights of A_S, A_L, F_S, and F_L. Pseudocode for this evolutionary search is given in Algorithm 3.

Evolutionary search proceeds over n_g generations. We initialize the first generation of agent pairs G_0 as n speaker-listener pairs (line 1). We then evaluate each pair on their performance on a referential game with vectors being communicated about drawn from a training set (lines 3,4).

On each generation i, the top n_r performing agent pairs are chosen to reproduce asexually (lines 5-14). We first sort the agent pairs in decreasing order by fitness (line 5). We set R to contain the top performing n_r agent pairs, which are the agent pairs that will reproduce to create the next generation (line 6). We then initialize the next generation G_{i+1} and carry over the best-performing agent pair unchanged, a process known as *elitism* (Deb et al., 2000) (line 7). During reproduction, we sample agent pairs (A_S, A_L) uniformly with replacement (line 9). When a reproducing pair (A_S, A_L) is sampled, its offspring (A'_S, A'_L) are initialized to be identical to their parents (line 10). Similarly, the offsprings' accompanying feedforward networks (F'_S, F'_L) are initialized identically to their parents'

Algorithm 3: Evolutionary Search

```
input : number of agent pairs $n$; number of reproducing agents $n_r$; training set $V$;
        number of distractors $k$; number of generations $n_g$; maximum message
        length $L$; length of referential game $T$; weight perturbation scale $\sigma$;
        architectural mutation probability $p_s$; individual mutation probabilities $\vec{p_e}$
1  $G_0 \leftarrow \left(\left(A_S^1, A_L^1\right), \left(A_S^2, A_L^2\right), \ldots, \left(A_S^n, A_L^n\right)\right)$
2  for $i = 1, \ldots, n_g$ do
3      foreach $(A_S^j, A_L^j) \in G_i$ do
4          $f((A_S^j, A_L^j)) \leftarrow$ refGame$(A_S^j, A_L^j, V, T, L, k)$
5      sort $G_i$ by decreasing $f$
6      $R \leftarrow \left(\left(A_S^1, A_L^1\right), (A_S^2, A_L^2), \ldots, (A_S^{n_r}, A_L^{n_r})\right)$
7      $G_{i+1} \leftarrow ((A_S^1, A_L^1))$
8      for $j = 1, \ldots, n$ do
9          sample $(A_S, A_L) \sim R$ uniformly with replacement
10         initialize offspring $(A'_S, A'_L) \leftarrow (A_S, A_L)$
11         initialize offspring feedforward networks $(F'_S, F'_L) \leftarrow (F_S, F_L)$
12         add Gaussian noise $\sigma \cdot N(0,1)$ independently to weights of $A'_S$, $A'_L$, $F'_S$ and $F'_L$
13         with probability $p_s$, independently mutate architecture of $A'_S$ and $A'_L$
             according to individual mutation probabilities $\vec{p_e}$
14         append $(A'_S, A'_L)$ to $G_{i+1}$
```

feedforward networks (F_S, F_L) (line 11). The weights of A'_S, A'_L, F'_S, F'_L are all independently perturbed with Gaussian noise drawn from $N(0, \sigma)$, where σ is called the *perturbation scale* (line 12). With probability p_s, the offspring pair (A'_S, A'_L) is chosen for architectural mutation (line 13). If architectural mutation does occur, a mutation occurs independently for each of A'_S and A'_L (see Section 4.3.3 for a detailed explanation of architectural mutation). The next generation is comprised of all offspring pairs from the previous generation, as well as the elite pair (line 14). This process repeats for n_g generations.

Offspring of speaker-listener pairs remain paired together. This is so that agents need only learn to communicate with one other agent, rather than having to form a common language with many other agents, which future work will explore.

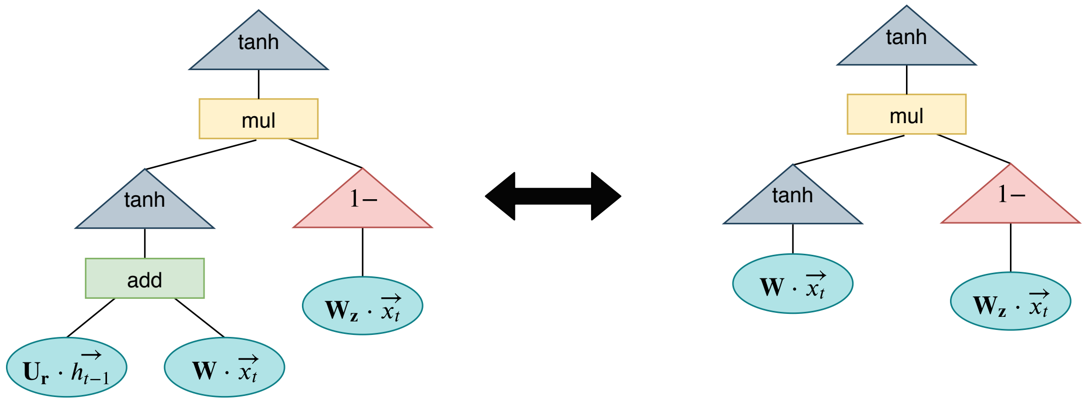
tanh
mul
tanh
1−
add
$\mathbf{W_z} \cdot \vec{x_t}$
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W} \cdot \vec{x_t}$
tanh
mul
tanh
1−
$\mathbf{W} \cdot \vec{x_t}$
$\mathbf{W_z} \cdot \vec{x_t}$

Figure 4.5: Binary insertion and removal mutation mechanisms. From left to right, this is binary removal in which the add node is removed and randomly replaced with its right child. Going from right to left is binary insertion, in which an add node is inserted and a random input node is chosen as its second child.

4.3.3 Details of Architectural Evolution

Evolution of RNN architectures for A_S and A_L is achieved through repeated mutation using five mechanisms: (i) replacement, in which binary or unary nodes are swapped for another random node of the same type (Figure 4.4), (ii) binary insertion, in which a binary node is randomly added as the parent of some node and a random input leaf is added as the binary node's other child (Figure 4.5), (iii) binary removal, in which a binary node is removed and replaced with one of its child subtrees at random (Figure 4.5), (iv) unary insertion, in which a random unary node is inserted as the parent of some node (Figure 4.6), and (v) unary removal, in which a unary operator is deleted and replaced with its child (Figure 4.6).

Throughout evolution, each time an offspring pair (A'_S, A'_L) is created, it will be chosen for architectural mutation with probability p_s. When architectural mutation occurs, the architectures of A'_S and A'_L will be mutated independently of each other. For both A_S and A_L, we independently sample a mutation operation according to individual mutation probabilities $\vec{p_e}$. The vector $\vec{p_e}$ is a probability vector ($0 \leq \vec{p_e}^{\,i} \leq 1$ for all i, and $\sum \vec{p_e}^{\,i} = 1$) such that each entry of $\vec{p_e}$ corresponds to a different architectural mutation. For example, $\vec{p_e} = (0.2, 0.1, 0.3, 0.15, 0.25)$ corresponds to a 20% chance that replacement occurs, a 10%

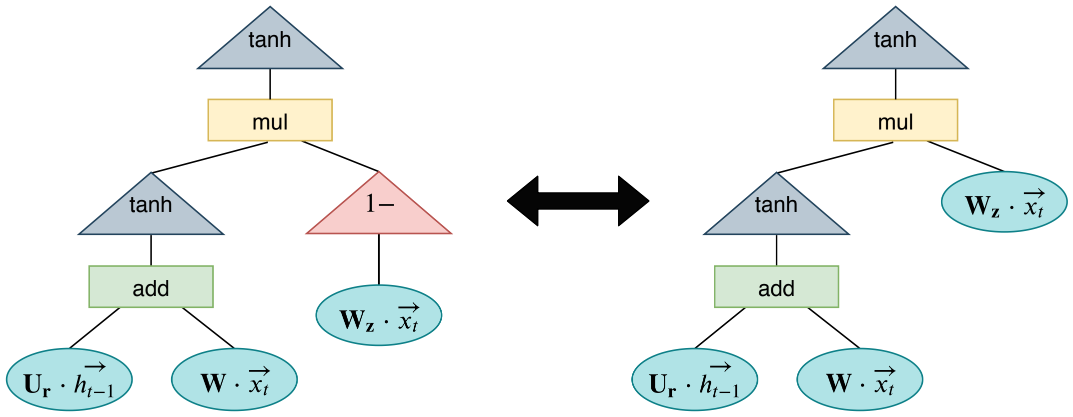
tanh
mul
tanh
1−
add
$\mathbf{W_z} \cdot \vec{x_t}$
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W} \cdot \vec{x_t}$
tanh
mul
tanh
$\mathbf{W_z} \cdot \vec{x_t}$
add
$\mathbf{U_r} \cdot \vec{h_{t-1}}$
$\mathbf{W} \cdot \vec{x_t}$

Figure 4.6: Unary insertion and removal mutation mechanisms. From left to right, this is unary removal in which the 1− node is removed and replaced with its child. Going from right to left is unary insertion, in which a 1− node is inserted as the parent of a random node.

chance that binary insertion occurs, a 30% chance that binary removal occurs, a 15% chance that unary insertion occurs and a 25% chance of unary removal. After a mutation is chosen for each of A'_S and A'_L, we randomly choose appropriate nodes in the program trees of A_S and A_L and apply these mutations.

As an example of architectural mutation occuring, suppose that agent pair (A'_S, A'_L) is chosen for mutation. We independently sample mutations for A'_S and A'_L. Suppose that binary removal is chosen for A'_S and unary insertion is chosen for A'_L. We randomly sample a binary node b in the program tree of A'_S and remove it, replacing b with one of its two inputs at random. For A'_L, we randomly select a node x from the program tree of A'_L. We then choose a unary operator (e.g., tanh, sigmoid) at random and insert a new unary node u which implements the chosen unary operator as the new parent of x.

To prevent our program trees from becoming arbitrarily large, we set a maximum program tree depth of 15. We also set a minimum tree depth of 4. Additionally, we do not allow consecutive nonlinearities (e.g., we do not allow a tanh node to become the parent of a sigmoid node).

Chapter 5

Evaluation

In this chapter, we evaluate the performance of agents trained to communicate using Algorithm 3. We compare the performance of four experimental conditions. We show that these agents not only learn to communicate but also that their communication extends to previously unseen objects drawn from a test set.

5.1 Dataset and Data Processing

We used the Visual Attributes for Concepts Dataset (Silberer et al., 2013). This dataset contains 500 concepts in 16 categories (Table 5.1). For each concept, there are human-generated annotations describing its characteristics. We processed the dataset as described by Lazaridou et al. (2018). Specifically, we removed the four smallest classes (devices, materials, plants, toys) as well as all homonym concepts (e.g., tank, bat). The final dataset contained 463 concepts in 12 classes with a total of 574 unique annotations, while the initial dataset contained 636 unique annotations. For each concept, a corresponding binary vector $\vec{b} \in \{0,1\}^{574}$ was created for which each position's entry in $\vec{b}$ corresponds to one of the 574 unique annotations, with a 1 indicating that the annotation is true for that concept and a 0 indicating that it is false. For example, the concept dog has ones in the positions corresponding to has_fur and has_legs but zeross in the positions corresponding to

Table 5.1: Categories of the Visual Attributes for Concepts dataset along with one example concept and some associated annotations

Category	Example concept	Example Annotatations
Animals	Bear	is_tall, has_claws, has_4_legs
Appliances	Stove	has_door, is_box_shaped, made_of_metal
Artefacts	Magazine	made_of_paper, has_pages, has_symbols
Clothing	Mittens	has_thumb, made_of_wool, has_2_pieces
Container	Bucket	made_of_plastic, has_handle, is_deep
Device	Microscope	has_lenses, has_eyepieces, has_arm
Food	Asparagus	has_stalks, is_green, is_long
Home	Chandelier	has_lightbulbs, is_branched, made_of_glass
Instruments	Bagpipe	has_pipes, has_tubing, made_of_different_fabrics
Material	Stone	different_shapes, made_of_minerals, different_colours
Plants	Dandelion	has_stem, is_green, is_yellow
Structures	Pyramid	made_of_stone, is_triangular, is_tall
Tools	Pliers	is_small, has_handle, has_pivot
Toys	Kite	is_flat, has_seams, has_string
Vehicles	Scooter	has_seat, has_2_wheels, has_exhaust_pipe
Weapons	Grenade	is_small, has_ring, made_of_metal

has_wheels and is_sharp. These binary vectors are the input to agents in the referential game. An example of two such binary vectors for two different concepts is shown in Figure 5.1. While we shortened the vectors in Figure 5.1 for an easier presentation, in our experiments they would both be of length 574.

5.2 Experimental Conditions

We compared the random RNN and GRU initial architectures (Section 4.3.1), each with and without architectural mutation. This resulted in four experimental conditions. We call the two conditions involving a random recurrent neural network **RT+E** and **RT**, denoting respectively the condition for which architectures are evolved, and the condition for which they are not. Similarly, we call the GRU conditions **GRU+E** and **GRU**.

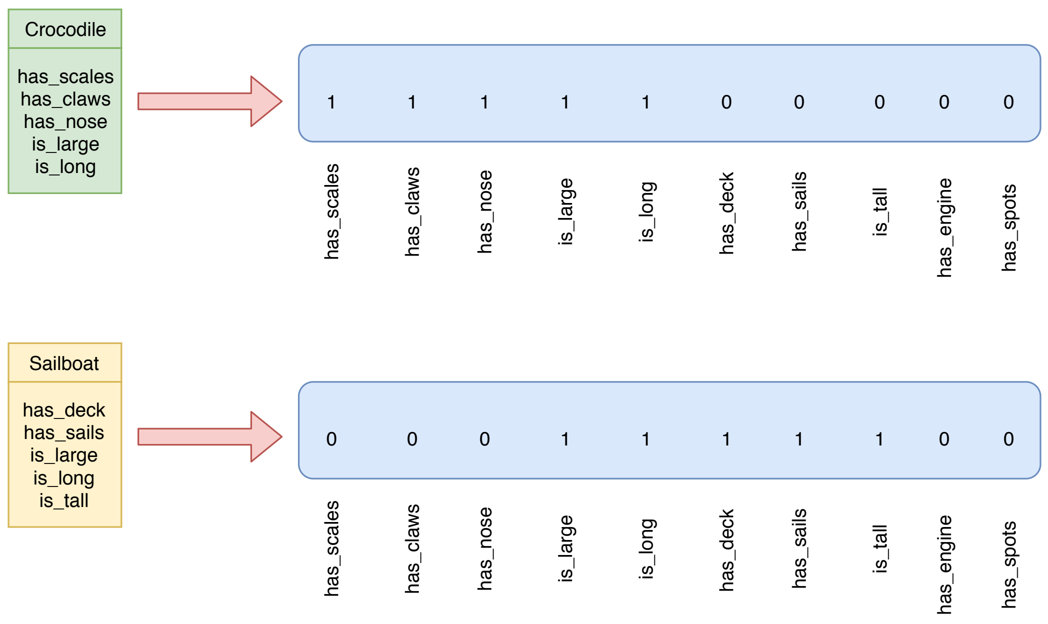

Figure 5.1: Example of two concepts (crocodile and sailboat) and their annotations being converted to binary vectors.

5.3 Evaluating Agents

In this section, we describe the process by which we evaluated each of the four experimental conditions. This process is presented in Algorithm 4, which describes the evaluation procedure for a single experimental condition, for a given maximum message length L.

We now describe Algorithm 4 in detail. To evaluate an experimental condition and given maximum message length L, we execute Φ evolution runs (line 1). On each evolution run ϕ, with $\phi \in \{1, ..., \Phi\}$, we first partition our set of concepts into a training set V_ϕ^{train} and test set V_ϕ^{test} (line 2). We note that training/test set partitions are consistent across all combinations of experimental conditions and maximum message lengths. That is, V_1^{train} will be the same for any combination of experimental condition and L, as will $V_1^{\text{test}}, V_2^{\text{train}}, V_2^{\text{test}}$, etc. For example, the first evolution run of experimental condition **RT+E** with $L = 2$ will have the same training/test set split as the first evolution run of **GRU** with $L = 5$.

Algorithm 4: Evaluation

```
input : number of evolution runs $\Phi$; number of agent pairs $n$; number of
        reproducing agents $n_r$; set of concepts $V$; number of distractors $k$; number
        of generations $n_g$; maximum message length $L$; training length of
        referential game $T$; extended training length of referential game $T_x$; testing
        length of referential game $T_\tau$; frequency of extended referential game $q$;
        weight perturbation scale $\sigma$; architectural mutation probability $p_s$;
        individual mutation probabilities $\vec{p_e}$
1  for $\phi = 1, \ldots \Phi$ do
2      randomly partition $V$ into training set $V_\phi^{\text{train}}$ and test set $V_\phi^{\text{test}}$
3      $G_0 \leftarrow ((A_S^1, A_L^1), (A_S^2, A_L^2), \ldots, (A_S^n, A_L^n))$
4      for $i = 1, \ldots, n_g$ do
5          foreach $(A_S^j, A_L^j) \in G_i$ do
6              if remainder$(i, q) = 40$ then
7                  $f((A_S^j, A_L^j)) \leftarrow$ refGame$(A_S^j, A_L^j, V_\phi^{\text{train}}, T_x, L, k)$
8              else
9                  $f((A_S^j, A_L^j)) \leftarrow$ refGame$(A_S^j, A_L^j, V_\phi^{\text{train}}, T, L, k)$
10         sort $G_i$ by decreasing $f$
11         if $i < n_g$ then
12             $R \leftarrow ((A_S^1, A_L^1), (A_S^2, A_L^2), \ldots, (A_S^{n_r}, A_L^{n_r}))$
13             $G_{i+1} \leftarrow ((A_S^1, A_L^1))$
14             populate $G_{i+1}$ with offspring of agents in $R$
15     $f_\phi \leftarrow$ refGame$(A_S^1, A_L^1, V_\phi^{\text{test}}, T_\tau, L, k)$
16 return $\frac{\sum_{\phi=1}^{\Phi} f_\phi}{\Phi}$
```

We initialize the first generation of agent pairs G_0 as n speaker-listener pairs (line 3). A given evolution run proceeds over n_g generations (line 4). Lines 5 to 9 detail the way in which agents are evaluated for a given generation. Each agent-pair in G_i is evaluated at a repeated referential game with concepts drawn from training set V_ϕ^{train}. On generations which are not divisible by q, these agents play T rounds of the referential game (line 9). Every qth generation, the agents are evaluated at a longer repeated referential game over T_x rounds (line 7). This longer repeated referential game was used because T was chosen to be small relative to the total number of possible samples. As such, agents could be chosen to reproduce which performed well on the T training samples chosen for a given generation, but performed poorly in general. So, every q generations we use a larger set of training samples to ensure that reproducing agents perform well on a larger subset of the training set.

For all generations other than the final generation n_g, agents reproduce to create the next generation (lines 11-14). We omit the details of reproduction here, as they are presented in depth in Algorithm 3.

On generation n_g, after sorting the agent pairs in G_{n_g} by decreasing fitness, we select the highest performing agent pair (A_S^1, A_L^1) from G_{n_g} to be evaluated on the evolution run's test set V_ϕ^{test} (line 15). That is, the agent pair selected to be tested from a given evolution run is the best performing agent pair from that evolution run's final generation. This best-performing agent pair is tested on T_τ samples consisting of concepts drawn from test set V_ϕ^{test}.

For a given experimental condition and maximum message length L, we average the test set performance of all Φ evolution runs executed for that experimental condition and maximum message length (line 16). This average is the performance of a given experimental condition. For example, suppose we ran Algorithm 4 for experimental condition **RT+E**, with $L = 2$ and $\Phi = 2$. Then two evolution runs would be executed, and each evolution run would produce a best-performing agent-pair in its final generation. These two agent pairs would be evaluated on their respective evolution run's test set and achieve test set performances of f_1 and f_2 respectively. Suppose $f_1 = 60\%$ and $f_2 = 50\%$. Then the performance of experimental condition **RT+E** with $L = 2$ would be $\frac{60\%+50\%}{2} = 55\%$.

5.3.1 Network Architecture

The feedforward networks F_S and F_L each had an input layer of size 574 and output layer of size 50, reducing input vectors of size 574 to representations of size 50. For agents A_S and A_L, we set weight matrices $\mathbf{U}, \mathbf{U_z}$ and $\mathbf{U_r}$ to size 60×50, $\mathbf{W}, \mathbf{W_z}$ and $\mathbf{W_r}$ to size 50×50 and $\mathbf{V}$ to size 50×60, giving these agents a hidden state of size 50 and vocabulary of size 60.

Figure 5.2 shows the sizes of all matrices and vectors involved when speaker A_S is generating a message symbol m_1. We show only a speaker, as all matrix and vector sizes are

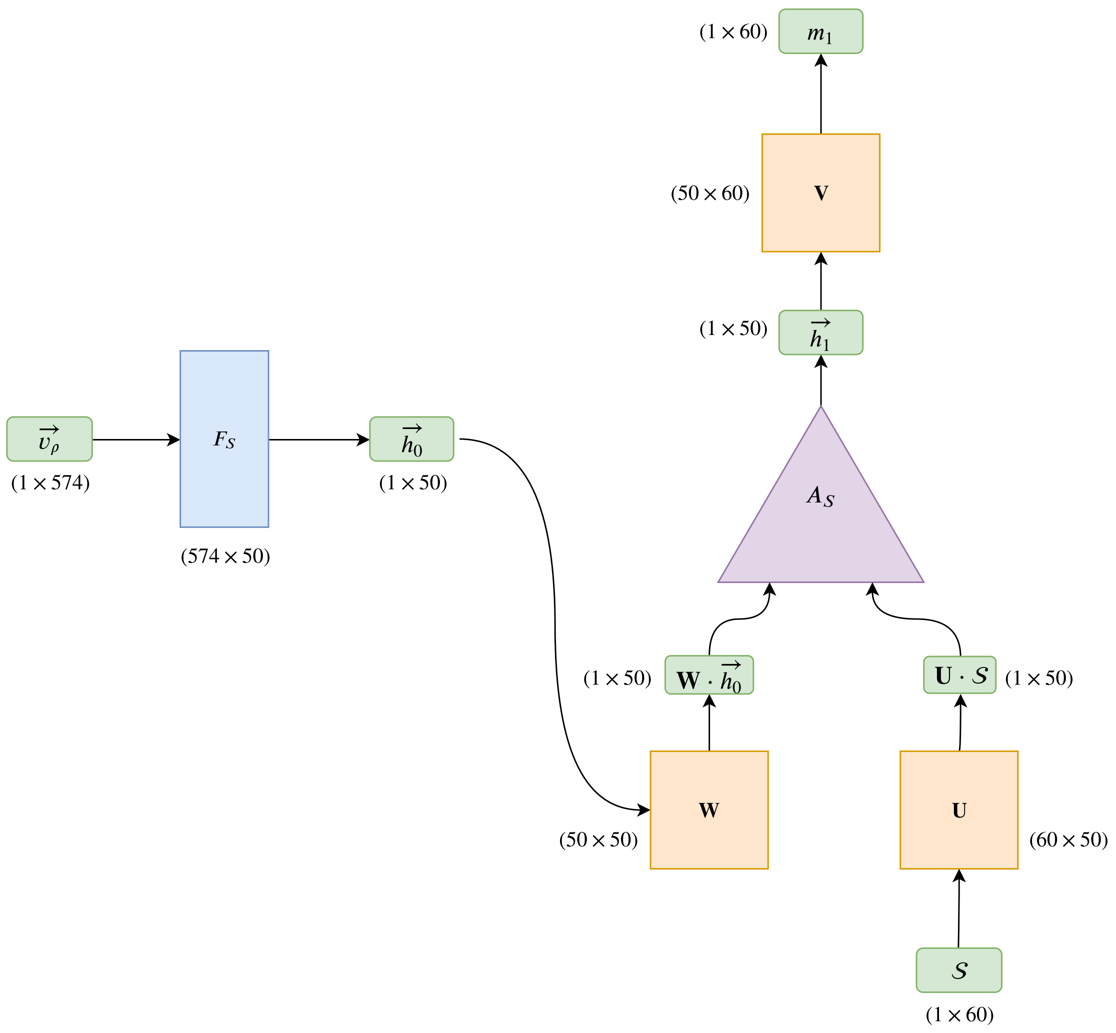

Figure 5.2: Matrix and vector sizes for a speaker generating a message.

identical for a listener. On round ρ, the speaker receives as input the target $\vec{v_\rho}$, a vector of size 1×574. $\vec{v_\rho}$ is processed by F_S, which is of size 574×50, to generate the speaker's initial state $\vec{h_0}$ of size 1×50. We compute $\mathbf{W} \cdot \vec{h_0}$, the first input to A_S which has dimension 1×50. We then multiply $\mathbf{U}$, a matrix of size 60×50 by the sentence-start token $\mathcal{S}$ of size 1×60 to generate $\mathbf{U} \cdot \mathcal{S}$, a vector of size 1×50. The vector $\mathbf{U} \cdot \mathcal{S}$ is the second input to A_S. In our experiments, four other inputs to this agent would be generated ($\mathbf{W_z} \cdot \vec{h_0}, \mathbf{W_r} \cdot \vec{h_0}, \mathbf{U_z} \cdot \mathcal{S}$, and $\mathbf{U_r} \cdot \mathcal{S}$). We omit these, though, for clarity. The inputs $\mathbf{W} \cdot \vec{h_0}$ and $\mathbf{U} \cdot \mathcal{S}$ are then processed by the program tree of A_S to generate new hidden state $\vec{h_1}$ of size 1×50. We multiply

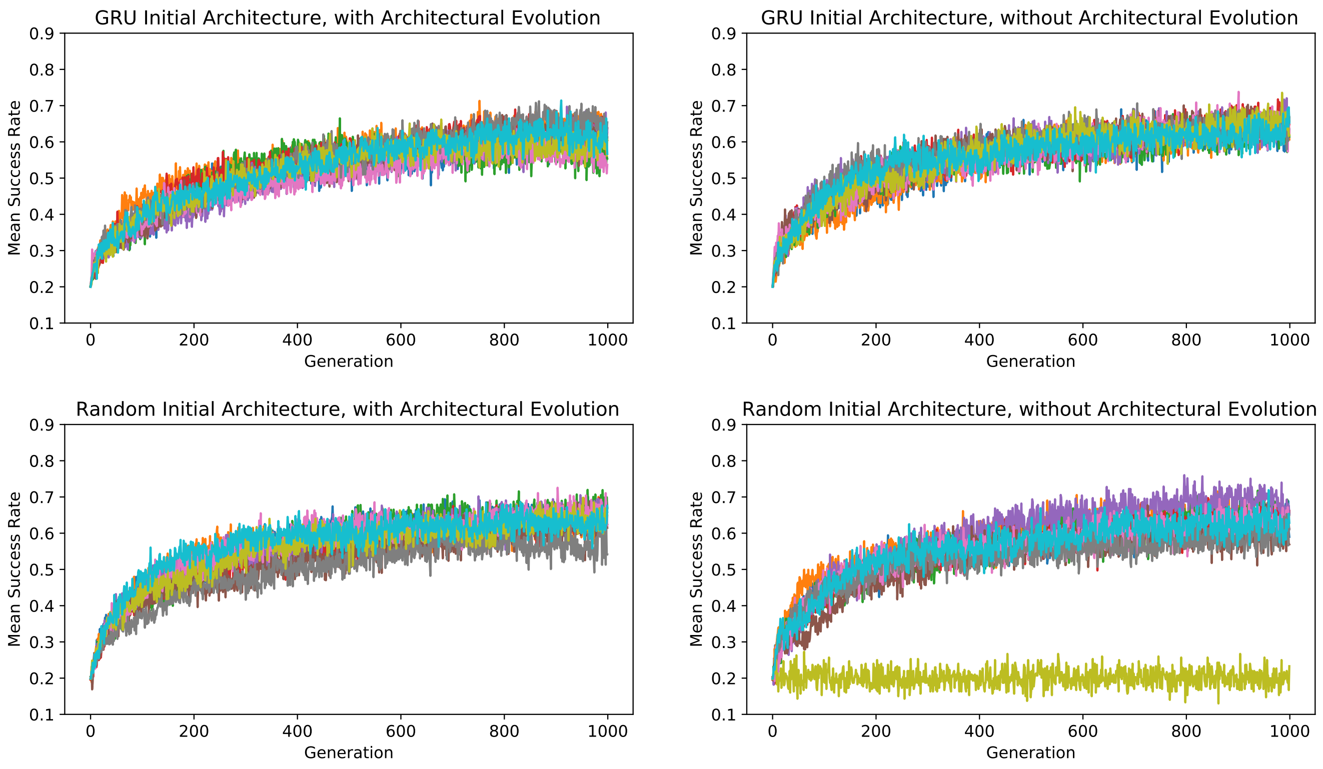

Figure 5.3: Mean generational performance on each generation during training, with maximum message length L set to 2.

$\vec{h_1}$ with matrix $\mathbf{V}$ to produce vector $\vec{m_1}$ of size 1×60, which will be the first element of message $\vec{m}$. The vectors $\vec{h_1}$ and $\vec{m_1}$ make up the next inputs to $\mathbf{W}$ and $\mathbf{U}$, respectively.

5.4 Chosen Hyperparameters

In this section, we first present the hyperparameters which we chose for all evolution runs. We then present the hyperparameters which were chosen to be different for runs with maximum message length $L = 2$ and $L = 5$. All hyperparameters were chosen through preliminary experiments.

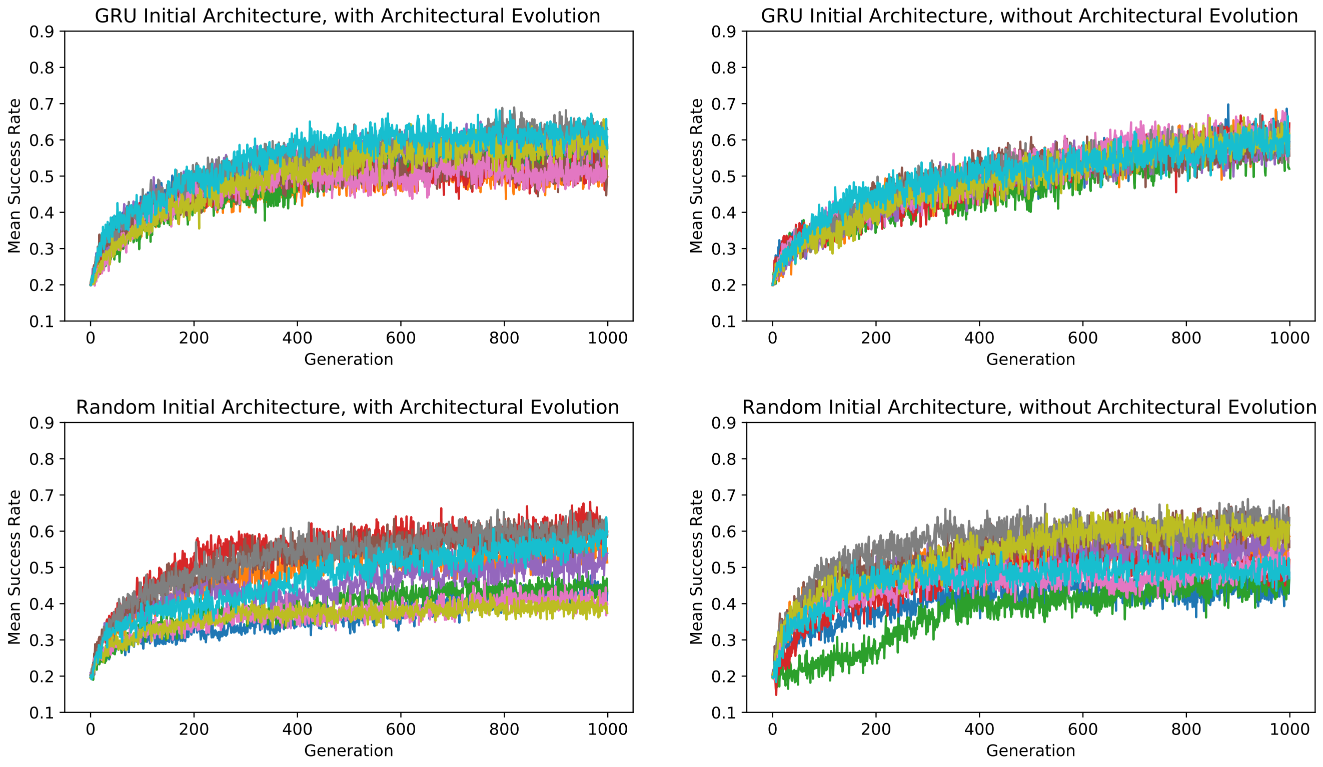

Figure 5.4: Mean generational performance on each generation during training, with maximum message length L set to 5.

Table 5.2 presents the hyperparameters that we chose to be the same for all evolution runs. The last two hyperparameters, p_s and $\vec{p_e}$, are applicable only to the experimental conditions which involve architectural evolution (conditions **RT+E** and **GRU+E**).

For runs with $L = 2$, we set population size n to 2000 and number of reproducing agents n_r to 60. For runs with $L = 5$, we set n to 1000 and n_r to 40. We reduced the population size for runs with $L = 5$ because increasing L increases computational cost. Reducing n from 2000 to 1000 kept the runtime of evolution runs with $L = 5$ similar to evolution runs with $L = 2$ (Table 5.5).

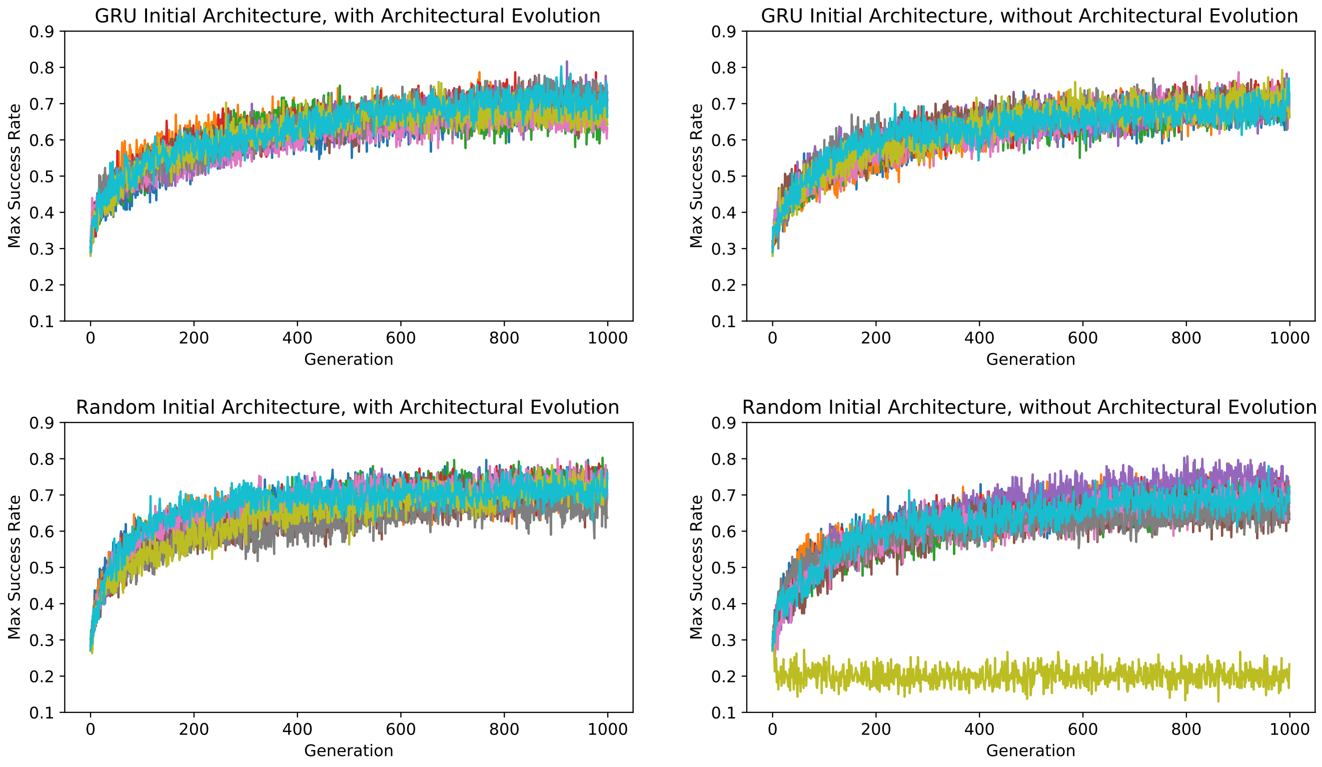

Figure 5.5: Maximum generational performance on each generation during training, with maximum message length L set to 2.

5.5 Statistical Methods

In this chapter we will be comparing the performance of experimental conditions. To ensure that these comparisons are meaningful, we will be using a z-test. We chose to use a z-test as agent success on a given round of the repeated referential game is binary (1 if the listener chooses correctly, 0 otherwise).

For each experimental condition and each L, we tested three agent pairs on a test set (one pair from each evolution run). Each agent pair was evaluated on 10000 samples from its evolution run's respective test set. So, for any given experimental condition and L, there were 30000 rounds of the referential game played with concepts drawn from a test set. For

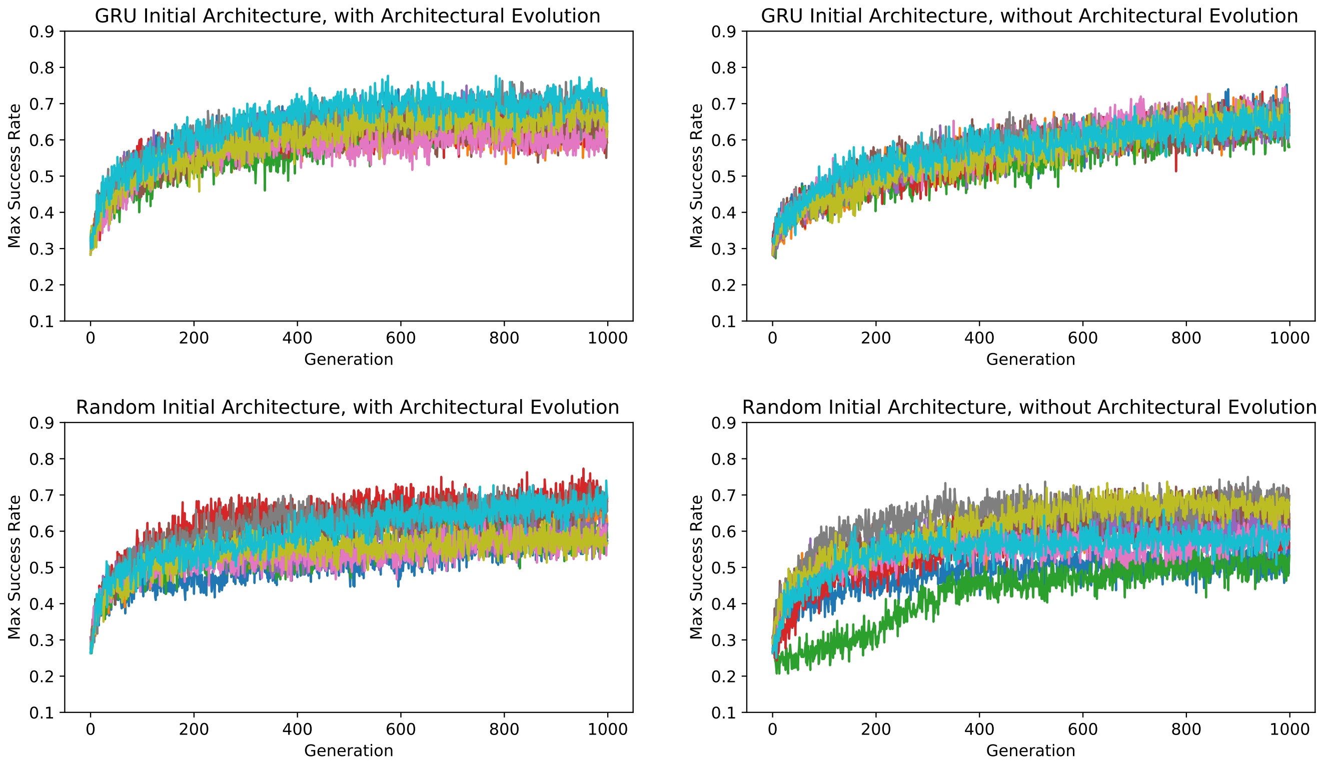

Figure 5.6: Maximum generational performance on each generation during training, with maximum message length L set to 5.

each experimental condition and L, we generated a data set consisting of that experimental condition's 30000 test set samples, and a binary number indicating the agent pair's outcome on that sample. For example, on round $\rho = 10$, an entry in this data set would look like $(\vec{v_{10}}, \vec{d^1_{10}}, \vec{d^2_{10}}, \vec{d^3_{10}}, \vec{d^4_{10}}, 1)$, where $\vec{v_{10}}$ is the target, the $\vec{d^i_{10}}$ are distractors, and the 1 indicates that the listener chose correctly.

To compare pairs of experimental conditions for a given L, we compare their respective data sets containing test set samples and outcomes. To do this we use a two-tailed z-test and consider the difference statistically significant if $p < 0.05$. When comparing experimental conditions to baseline random chance, we use a one-tailed z-test. One shortcoming of this approach is that the z-test ignores variance between runs. This is because the test

Table 5.2: Hyperparameters which were chosen to be identical for all evolution runs. Hyperparameters p_s and $\vec{p_e}$ are applicable only to experimental conditions **RT+E** and **GRU+E**.

Hyperparameter	**Value**
Total number of generations n_g	1000
Number of repeated referential game rounds T	300
Number of extended repeated referential game rounds T_x	3000
Number of repeated referential game rounds for testing T_τ	10000
Frequency of extended referential game q	40
Number of distractors k	4
Weight perturbation scale σ	0.008
Architectural mutation probability p_s	0.5
Individual mutation probabilities $\vec{p_e}$	$(0.15, 0.3, 0.3, 0.15, 0.1)$

set performance of all three agent pairs tested for a given experimental condition and L is amalgamated into a single data set despite the performance of different agents having potentially different variance.

5.6 Results and Discussion

In this section, we present the results of our experiment. We show the test set performance achieved for all experimental conditions along with statistical analysis to demonstrate that the results are significant. We discuss our results along with their implications.

The test set performance of each experimental condition is presented in Table 5.3 for maximum message length $L = 2$ and in Table 5.4 for $L = 5$. We executed 10 evolution runs for each experimental condition with $L = 2$ and $L = 5$. For both tables, the first four rows show the mean test set performance and standard deviation of the agent pairs chosen from that condition to be tested, for each of the 10 evolution runs executed (as described in Section 5.3). These agents were all tested on 10000 samples consisting of previously unseen concepts drawn from a test set. The fifth row of each table shows the test performance of the single agent pair tested by (Lazaridou et al., 2018).

Table 5.3: Mean and standard deviation of success rates in the referential game when referring to objects drawn from a test set for each experimental condition with maximum message length 2. Means and standard deviations are for 10 trials. All pair-wise comparisions of experimental conditions revealed significant differences (two-tailed z-test, $p < 0.05$).

Experimental Condition	Success Rate
RT+E	$55.2\% \pm 3.6\%$
RT	$50.2\% \pm 10.6\%$
GRU+E	$52.5\% \pm 2.8\%$
GRU	$55.9\% \pm 4.2\%$
Lazaridou et al. (2018)	74.2%

5.6.1 Comparison to Baseline Chance

Given that we used $k = 4$ distractors, if agents guessed randomly then their expected success rate would be 20% (the baseline). All experimental conditions performed significantly better than baseline chance (one-tailed z-test, $p = 10^{-16}$ for all conditions). This shows that gradient-free evolutionary computation can be used to train deep neural networks to develop shared languages from scratch, and that these languages can generalize to previously unseen objects.

5.6.2 Comparison to Previous Work

We now compare the performance of our gradient-free evolutionary approach to previous results from work using gradient-based deep reinforcement learning (Lazaridou et al., 2018). We first note, though, that this comparison has flaws. Lazaridou et al. (2018) presented only the performance of a single best agent pair, while we presented the performance of 3 agent pairs for each combination of experimental condition with L. Additionally, no information regarding train/test set splits or computational resource utilization is given in Lazaridou et al. (2018). Agent pairs tested by Lazaridou et al. (2018) achieved a test-set success rate of 74.2% when $L = 2$ and 76.8% when $L = 5$. The best mean performance of any of our experimental conditions with $L = 2$ was 55.9%, with condition **GRU**. Comparing the 74.2% seen in previous work to the mean performance of **GRU**, we see a decrease of 18.3%.

Table 5.4: Mean and standard deviation of success rates in the referential game when referring to objects drawn from a test set for each experimental condition with maximum message length 5. Means and standard deviations are for 10 trials. All pair-wise comparisions of experimental conditions revealed significant differences (two-tailed z-test, $p < 0.05$).

Experimental Condition	Success Rate
RT+E	$49.8\% \pm 5.3\%$
RT	$48.0\% \pm 6.5\%$
GRU+E	$51.4\% \pm 4.4\%$
GRU	$53.9\% \pm 3.3\%$
Lazaridou et al. (2018)	76.8%

Table 5.5: CPU days used for training, averaged over 10 evolution runs for each of the experimental conditions, for $L = 2$ and $L = 5$.

	Experimental condition			
Maximum message length	**RT+E**	**RT**	**GRU+E**	**GRU**
$L = 2$	59.6	32.9	65.4	34.8
$L = 5$	46.5	29.7	55.05	29.7

Similarly, with $L = 5$, comparing 76.8% to the mean performance of condition **GRU**, we see a decrease of 22.9%.

5.6.3 Comparisons Between Experimental Conditions

For both $L = 2$ and $L = 5$, all differences between experimental conditions were shown to be significant (two-tailed z-test, $p < 0.05$). For the cases when $L = 2$ and $L = 5$, condition **GRU** had the highest mean test set performance. This shows that we were not able to evolve the architecture of agents which performed the given task better than a well-known contemporary architecture. We believe that the primary explanation for this fact is the simplicity of the architectural evolution algorithm which we used. We elaborate on this explanation further in Chapter 6.

Condition **RT+E** outperformed condition **RT** when $L = 2$ and $L = 5$. This suggests that architectural evolution can be beneficial for cooperative communication tasks, as evolved architectures were able to significantly outperform their fixed counterparts.

Table 5.6: Selected messages of maximum length 2 with associated concepts generated by a high performing speaker. Concepts are drawn from a test set consisting of previously unseen concepts.

Message	Concepts
[10,5]	woodpecker, eagle, chicken, goose
[10,6]	parakeet
[10,10]	duck, toad, penguin
[10,33]	lobster
[28,12]	tractor
[28,22]	taxi
[28,34]	scooter, subway
[53,1]	shield, otter
[53,10]	buffalo, pig, fawn, bison, sheep, bull, ox
[53,35]	dagger, iguana
[53,36]	dog

For both $L = 2$ and $L = 5$, condition **GRU+E** performed worse than condition **GRU**. This is consistent with previous work (Stanley and Miikkulainen, 2002) which showed that architectural evolution is more effective when the initial agent architectures used are small. Diminished performance was shown by Stanley and Miikkulainen (2002) when agent architectures were initialized to larger, hand-crafted networks.

5.6.4 Learned Languages

In this section we present a qualitative analysis of the languages that agents learned.

One desirable natural-language property of emergent languages is *compositionality*, in which elements of a language are constructed from simpler elements of that language. For example, sentences in English are sequences of words which are simpler than the sentence itself. Compositionality is desirable because it allows elements of a language to be combined in arbitrarily many ways so as to produce new elements of that language. The languages generated by our agents exhibited evidence of compositionality. This evidence is best seen in Table 5.6, which shows a selection of messages along with their associated concepts for a high-performing speaker-listener pair with $L = 2$. Rather than concepts being randomly associated with messages, a pattern can be seen in which this speaker grouped concepts

Table 5.7: Selected messages of maximum length 5 with associated concepts generated by a poorly performing speaker. Concepts are drawn from a test set consisting of previously unseen concepts.

Message	**Concepts**
[27, 12, 3, 12, 30]	garage
[27, 20, 12, 20, 12]	fox, rat, pig, porcupine, fawn, iguana, python, sheep, dog, ox
[27, 20, 12, 20, 27]	toad
[27, 20, 17, 20, 12]	bureau
[27, 20, 17, 20, 27]	cello
[27, 20, 27, 20, 4]	spear
[27, 20, 27, 20, 12]	yam
[27, 20, 27, 20, 13]	pyramid
[27, 20, 27, 20, 17]	bookcase
[27, 20, 55, 34, 55]	bed

into categories by assigning the same prefix to similar concepts. For example, messages beginning with 10 referred to birds, and messages beginning with 53 referred to four-legged animals as well as some weapons.

As described by Lazaridou et al. (2018), further evidence of compositionality is *productivity*, in which new messages can be made up *on the fly*. That is, when a speaker can create new messages to describe previously unseen objects which can be understood by the listener. Our agents exhibited productivity, as can be seen with the messages $[10, 6]$, $[28, 22]$ and $[2, 2]$ which refer, respectively, to the test set concepts parakeet, taxi, and mittens. These three messages were not used for any concepts in this speaker's training set but were consistent with the structure of messages used by the speaker to refer to training set concepts. That is, the speaker used messages beginning with 10, 28, and 2, respectively, to refer to training set concepts which were birds, vehicles, and clothing.

Table 5.7 sheds light on the diminished performance seen with experimental condition **RT** when L was increased to 5. This table shows all of the messages beginning with 27 and their associated test-set concepts for a speaker which was chosen from one of experimental condition **RT**'s evolution runs. While the agent did learn to group some similar concepts ([27, 20, 12, 20, 12] refers to four-legged animals), there is minimal evidence of compositionality. These messages, despite being structurally similar, refer to dissimilar concepts. The communication strategy learned by this agent seemed to randomly assign messages to concepts. This suggests that agent architectures with no specific engineering struggle to develop compositional languages when the space of possible messages is large.

Chapter 6

Future Work

In this section, we suggest extensions and modifications to our work which could further improve performance, as well as analyses which could provide additional insight into the languages and agents which emerged. In addition, we discuss other communication environments to which this methodology could be applied that may further shine light on the usefulness of evolutionary computation for emergent communication.

6.1 Future Research Directions

Due to time constraints, we were only able to execute ten evolution runs for each combination of experimental condition and L. Future work should replicate these results with a larger number of evolution runs (e.g., 50 runs per combination of experimental condition and L).

While this thesis demonstrated potential applicability of architectural evolution to emergent communication, as evidenced by the improved performance of RNNs with randomized architecture when allowed to evolve, no evolved architecture outperformed a fixed Gated Recurrent Unit. Previous work on evolving neural networks has shown that this is likely because mutation which could eventually lead to improved performance often renders offspring temporarily less viable, as the mutation needs time to mature. This has been addressed by using NEAT (Stanley and Miikkulainen, 2002; Rawal and Miikkulainen, 2018) to divide the population into species and temporarily protect innovation. Utilizing aspects of NEAT could improve the usefulness of architectural mutation for communication. Additionally, using more complex reproduction strategies such as crossover could further improve performance.

As this was preliminary work in applying evolutionary computation to emergent communication, only a single environment was explored - namely, a repeated referential game. This environment was chosen because it is complex enough for interesting languages to evolve and simple enough for these languages to be easily detected. The referential game has limitations, though, when used to evaluate the effectiveness of an algorithm for training agents to communicate. Learning is one-shot and does not involve long horizons. In addition, agents need only learn to communicate so as to be successful - there are no other elements of the environment which need be learned as well, or factors which make communication more difficult. Given that evolutionary algorithms have been shown to be effective when faced with tasks which have misleading gradients and long horizons, other environments may better illustrate the usefulness of evolutionary computation for communication. In particular, artificial-life (Langton, 1997; Bulitko et al., 2018) and other multi-agent grid environments (Mordatch and Abbeel, 2018; Lowe et al., 2017) could provide additional insight into the efficacy of the techniques explored in this thesis.

In this thesis, we presented only a visual inspection of emerged languages and no inspection of architectures which emerged from evolution. Previous work (Lazaridou et al., 2018) has used tests inspired by linguistics to explore natural language properties of emerged languages. Additionally, when using architectural mutation, the emerged architectures have been studied and shown to have interesting properties (Rawal and Miikkulainen, 2018), such as LSTM-like components (Hochreiter and Schmidhuber, 1997a). In future work, it would be interesting to investigate what natural-language properties emerge in languages developed with evolutionary computation rather than deep reinforcement learning. Additionally, it could be fruitful to further explore the structures which emerge, as these could provide insight into what aspects of a deep neural network's architecture make it effective at learning to communicate.

We trained only pairs of agents to communicate, and different pairs of agents were separate in the sense that offspring of speaker-listener pairs remained paired. As such, the languages of two different speaker-listener pairs could be completely different and it would not affect performance. In future work, it would be interesting to explore whether or not the techniques developed in this thesis can be applied to evolve larger populations of agents, all of which share a single language.

While this thesis explored communication amongst deep neural networks, another open problem is development of mechanisms for machines to communicate with humans. One difficulty, though, is that deep neural networks must currently pass many thousands of messages to each other for meaningful language to develop, which is infeasible when a machine is communicating with a human. One possible solution to this is the use of human proxies (Bulitko et al., 2019). Rather than having a machine communicate directly with a human, some human responses could be collected and modelled. Machines could then communicate with this model as a proxy for true human interaction. Future work will explore utilization of human proxies to make progress towards communication between neural networks and humans.

Chapter 7

Conclusion

In this thesis, we presented a gradient-free evolutionary approach for training pairs of deep neural networks to develop their own languages from scratch and use these languages to communicate and cooperate. To achieve this, we used a genetic algorithm to search for both the weights and architectures of communicating agents. The specific communication task that we trained our agents on was a repeated referential game.

We showed that our approach - while achieving weaker results than those previously demonstrated with gradient-based reinforcement learning - performed well above random chance. We demonstrated that the learned languages are compositional and generalize to previously unseen objects. Additionally, we demonstrated that agents whose architecture is evolved can, in some cases, outperform agents with fixed, hand-crafted architectures.

The results of this thesis imply that a gradient-free evolutionary approach can be used to train deep neural networks to communicate and cooperate, which had previously only been achieved with gradient-based supervised and reinforcement learning. This provides compelling motivation for the application of evolutionary computation to future research in cooperative, multi-agent settings.

Bibliography

Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford university press, 1996.

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

Vadim Bulitko, Mac Walters, Morgan Cselinacz, and Matthew R. Brown. Evolving NPC behaviours in A-life with player proxies. In *Proceedings of the Experimental AI in Games (EXAG) Workshop at the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2018.

Vadim Bulitko, Kacy Doucet, Daniel Evans, Hope Docking, Mac Walters, Marilene Oliver, Julian Chow, Shelby Carleton, and Natali Kendal-Freedman. A-life evolution with human proxies. *The 2019 Conference on Artificial Life*, (31):465–466, 2019. doi: 10.1162/isal_a_00204. URL https://www.mitpressjournals.org/doi/abs/10.1162/isal_a_00204.

Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *arXiv preprint arXiv:1804.03980*, 2018.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *In ICML*, pages 160–167. ACM, 2008.

Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer, 2000.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

David B Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons, 2006.

Serhii Havrylov and Ivan Titov. Emergence of language with multi-agent games: learning to communicate with sequences of symbols. In *NIPS*, pages 2149–2159, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997a. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

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

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

William B Langdon. Size fair and homologous tree genetic programming crossovers. In *In GECCO-Volume 2*, pages 1092–1097. Morgan Kaufmann Publishers Inc., 1999.

Christopher G Langton. *Artificial life: An overview.* Mit Press, 1997.

Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. *arXiv preprint arXiv:1612.07182*, 2016.

Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls, and Stephen Clark. Emergence of linguistic communication from referential games with symbolic and pixel input. *CoRR*, abs/1804.03984, 2018. URL http://arxiv.org/abs/1804.03984.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. URL http://arxiv.org/abs/1506.00019.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017. URL http://arxiv.org/abs/1706.02275.

Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017. URL http://arxiv.org/abs/1703.00548.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *CoRR*, abs/1703.04908, 2017. URL http://arxiv.org/abs/1703.04908.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Steven T Piantadosi, Harry Tily, and Edward Gibson. The communicative function of ambiguity in language. *Cognition*, 122(3):280–291, 2012.

Aditya Rawal and Risto Miikkulainen. From nodes to networks: Evolving recurrent neural networks. *CoRR*, abs/1803.04439, 2018. URL http://arxiv.org/abs/1803.04439.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Carina Silberer, Vittorio Ferrari, and Mirella Lapata. Models of semantic representation with visual attributes. In *In ACL (Volume 1: Long Papers)*, volume 1, pages 572–582, 2013.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

J. Sirota, V. Bulitko, M. R. G. Brown, and S. Poo Hernandez. Towards procedurally generated languages for non-playable characters in video games. In *2019 IEEE Conference on Games (CoG)*, pages 1–4, Aug 2019. doi: 10.1109/CIG.2019.8848093.

Joshua Sirota, Vadim Bulitko, Matthew R. G. Brown, and Sergio Poo Hernandez. Evolving recurrent neural networks for emergent communication. In *GECCO*, 2019.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

L. Steels. *The Talking Heads experiment: Origins of words and meanings*. Computational Models of Language Evolution. 2015. ISBN 9783944675428. URL https://books.google.ca/books?id=jtEsCQAAQBAJ.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017. URL http://arxiv.org/abs/1712.06567.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Paul Vogt. On the acquisition and evolution of compositional languages: Sparse input and the productive creativity of children. *Adaptive Behavior*, 13(4):325–346, 2005. doi: 10.1177/105971230501300403. URL https://doi.org/10.1177/105971230501300403.

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

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.