Emergent Communication with Stack-Based Agents

Daichi Kato (daichi5967@is.s.u-tokyo.ac.jp)
Ryo Ueda (ryoryoueda@is.s.u-tokyo.ac.jp)
Jason Naradowsky (naradowsky@is.s.u-tokyo.ac.jp)
Yusuke Miyao (yusuke@is.s.u-tokyo.ac.jp)
The University of Tokyo, Japan

Abstract

Emergent communication (EC) is the field that seeks to understand the mechanisms behind the emergence and evolution of natural language. In EC, the de facto standard has been using sequential architectures that have not explicitly incorporated the "tree-structured hierarchy" inherent in human language. This study utilizes a stack-based model called RL-SPINN, which learns tree structures through reinforcement learning without ground-truth parsing data, and acquires sentence representations according to these structures. We use this model as the basis for the understanding agents and investigate the extent to which the inductive bias of an architecture that explicitly utilizes tree structures affects the emergent language. The experimental results show that the emergent language generated by our model exhibits higher communication accuracy than those generated by other baselines in some settings. This work is the first to focus on the tree-structured hierarchy of language and suggests new directions for future research in EC.

Keywords: emergent communication; stack-based agents; tree-structured hierarchy of language

Introduction

Human language is one of the essential features of humans and a vital tool to convey information such as culture, knowledge, and understanding of the world, yet the question of how language itself emerged and developed remains shrouded in mystery. One possible approach that may shed light on the origins of language is the computational study of the circumstances and pressures where it may arise, known as **emergent communication (EC)** (Lazaridou & Baroni, 2020). In the EC field, agents composed of neural networks simulate human conversation, and the message protocols that arise between them are considered a language, called **emergent language**. EC investigates how the emergent language is similar to or different from human natural language and explores the conditions necessary for the emergent language to acquire characteristics similar to natural language.

In this paper, we focus on the hierarchical nature of human language and propose to utilize a stack-based model that respects hierarchy as the basis of the agent architecture. In previous work, agent architectures have commonly been based on standard models that are widely used in the general machine learning literature (Lazaridou, Hermann, Tuyls, & Clark, 2018; Chaabouni, Kharitonov, Bouchacourt, Dupoux, & Baroni, 2020), such as Simple RNN (Elman, 1990), LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), and Transformer (Vaswani et al., 2017). However, such de facto standard architectures do not necessarily have an explicit inductive bias to prefer hierarchical structures. The importance of hierarchical grammatical structures in human language has been discussed for decades (Chomsky, 1957), and it has recently been pointed out that it is crucial to modify the architectures explicitly handle syntactic structures for modeling human-like sentence processing (Linzen, 2020). For instance, RNNG (Dyer, Kuncoro, Ballesteros, & Smith, 2016), a model that explicitly deals with hierarchical structures, has been shown to outperform conventional sequential RNNs in grasping grammar (Kuncoro et al., 2018; Wilcox, Qian, Futrell, Ballesteros, & Levy, 2019). Moreover, both psycholinguistic (Hale, Dyer, Kuncoro, & Brennan, 2018) and neurophysiological (Nelson et al., 2017) perspectives acknowledge the plausibility of hierarchical syntactic structures in human cognitive functions. Thus, looking beyond the field of EC, it is known that the mechanism of tree-structured syntactic understanding is essential from the standpoint of human language comprehension.

Given this background, it is worth investigating the extent to which agents composed based on a stack-based model affect the emergent language. We utilize the model called RL-SPINN (Yogatama, Blunsom, Dyer, Grefenstette, & Ling, 2017), which learns tree structures with reinforcement learning. This approach allows us to explore models that include inductive bias to account for human cognitive functional aspects. In EC, the framework often used to get the objective is the **signaling game** (Lewis, 1969). The signaling game involves two types of agents: the sender and the receiver, both of which aim to cooperate and communicate information via their emergent language. In this study, we applied RL-SPINN as the basic architecture for the receiver to construct meaning representations from messages while retaining the other architectures as they are.

As an evaluation metric, we adopt **communication accuracy (ComAcc)**, which refers to the proportion of successful games for unseen data during training. Yogatama et al. (2017) demonstrated that RL-SPINN learns tree structures with a bias towards better capturing the meaning of natural language sentences, resulting in improved generalization performance. Similarly, we hypothesized that RL-SPINN-based receivers would learn tree structures that better capture the meaning of messages from the sender, allowing for more flexible construction of meaning representations for each message, thus

In L. K. Samuelson, S. L. Frank, M. Toneva, A. Mackey, & E. Hazeltine (Eds.), *Proceedings of the 46th Annual Conference of the Cognitive Science Society.*

enhancing communication accuracy. The experimental results show that our model demonstrates high ComAcc in a setting that is conjectured to be more similar to the semantic space treated by human language. This work represents the first study in EC that uses agents mimicking human tree-structured understanding, potentially suggesting new topics in the field.

Background

This section provides an overview of the stack-based model used in our work as a tree-structured mechanism for the receiver agent to understand messages and the framework typically used in EC.

Tree-Structured Variations of LSTM

In this study, we adopt RL-SPINN (Yogatama et al., 2017), which can learn tree structures using reinforcement learning, as the basis of the receiver agent. In EC, where languages vary with each execution, it is impractical to generate ground-truth parsing data for each of them. RL-SPINN's ability to learn tree structures without relying on parsing data makes it an ideal basis for a receiver.

Tree-LSTM The approach of using sentence tree structures to generate sentence meaning representations has been used in computational models (Goller & Küchler, 1996; Socher, Lin, Ng, & Manning, 2011). Tai, Socher, and Manning (2015) also recognized the potential value in the relationship between sentence meaning and tree-structured syntactic features. They attempted to extend the conventional sequential LSTM (Hochreiter & Schmidhuber, 1997) to a tree-structured model called **Tree-LSTM**.

While the conventional LSTM processes tokens sequentially from left to right to build meaning representations, Tree-LSTM builds meaning representations according to the additional input parse trees. Given a corresponding parse tree with each input sentence, Tree-LSTM constructs its representation hierarchically according to the parse structure. Tai et al. (2015) experimentally showed that Tree-LSTM outperformed conventional LSTM in two tasks of semantic understanding of natural language: prediction of semantic relatedness (SemEval-2014 Task 1 (Marelli et al., 2014)) and sentiment classification (Stanford Sentiment Treebank (Socher et al., 2013)). Furthermore, subsequent studies have also confirmed that tree-structured models outperform conventional models (Li, Luong, Jurafsky, & Hovy, 2015; Bowman, Manning, & Potts, 2015).

SPINN Tree-LSTM has the drawback of relying on an external parser since it requires inputting parsed results for each sentence. To mitigate this issue, Bowman et al. (2016) proposed a stack-based model called **SPINN** (**S**tack-augmented **P**arser-**I**nterpreter **N**eural **N**etwork), shown in Figure 1, which can determine the tree structure while simultaneously creating the sentence representation. The parser in SPINN is based on and motivated by the shift-reduce parsing. The shift-reduce parser generates a sequence of two types of actions, `SHIFT` and `REDUCE`, to parse a sentence. The shift-reduce parser has a **stack** and a **buffer**, where initially the stack is empty and the buffer contains all words of the sentence. `SHIFT` represents the action of piling the buffer's foremost element to the top of the stack, and `REDUCE` indicates the action of combining the stack's top two elements and returning to the top of the stack. For instance, given a sentence $\boldsymbol{x} = (a, young, man, sleeps)$, the output action sequence $\boldsymbol{a}$ becomes $\boldsymbol{a}$ = `SSSRRSR`, where `S`, `R` represents an abbreviation for `SHIFT`, `REDUCE`, respectively. This sequence parses the sentence into the tree structure $((a\ (young\ man))\ sleeps)$.

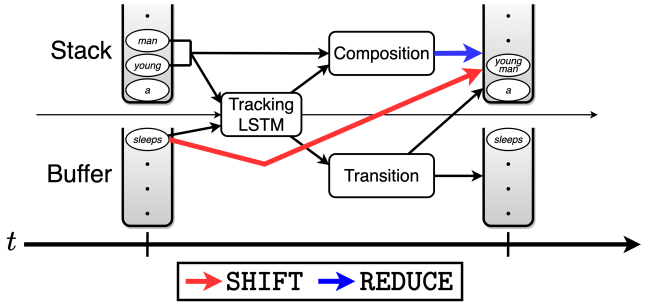


Figure 1: Schematic diagram of SPINN. This figure shows the behavior when `REDUCE` action occurs to compose *young* and *man*. The red arrows are the paths taken when `SHIFT` action is processed, the blue arrows when `REDUCE` action, and the black arrows when both actions.

As in the shift-reduce parser, SPINN incorporates an internal stack and buffer, which manage arrays whose length is equal to the maximum size of sentences. These arrays contain elements that are combinations of two vectors of dimension D: the hidden vector $\boldsymbol{h}$ and the memory representation $\boldsymbol{c}$. In the initial state, the stack is empty, and the buffer contains the elements of a sentence, each embedded into a $2 \cdot D$-dimensional space.

In addition to stack and buffer, SPINN has a component called **tracking LSTM**. This component consists of a conventional LSTM, which takes as input the stack's top two elements and the buffer's foremost element. The D_{tracking}-dimensional hidden vector of this component $\boldsymbol{h}_{\text{tracking}}$ plays two roles: composition and transition.

Composition. During a `REDUCE` operation, the top two pairs of a hidden vector and a memory representation in the stack, denoted as $\langle \boldsymbol{h}_s^1, \boldsymbol{c}_s^1 \rangle$ and $\langle \boldsymbol{h}_s^2, \boldsymbol{c}_s^2 \rangle$, are composed into one pair of $\langle \boldsymbol{h}_s, \boldsymbol{c}_s \rangle$ according to the following eq. (1) and then placed on top of the stack:

$$\begin{bmatrix} \boldsymbol{i} \\ \boldsymbol{f}_l \\ \boldsymbol{f}_r \\ \boldsymbol{o} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(W_{\text{comp}} \begin{bmatrix} \boldsymbol{h}_s^1 \\ \boldsymbol{h}_s^2 \\ \boldsymbol{h}_{\text{tracking}} \end{bmatrix} + \boldsymbol{b}_{\text{comp}} \right) \quad (1)$$

$$\boldsymbol{c}_s = \boldsymbol{f}_l \odot \boldsymbol{c}_s^2 + \boldsymbol{f}_r \odot \boldsymbol{c}_s^1 + \boldsymbol{i} \odot \boldsymbol{g}$$

$$\boldsymbol{h}_s = \boldsymbol{o} \odot \boldsymbol{c}_s$$

where $W_{\text{comp}}, \boldsymbol{b}_{\text{comp}}$ are learnable parameters, σ denotes the sigmoid activation function, and $\odot$ signifies the elementwise product. Ultimately, one pair of $\langle \boldsymbol{h}_s^{\text{last}}, \boldsymbol{c}_s^{\text{last}} \rangle$ remains in the stack, with the buffer being emptied. The hidden vector $\boldsymbol{h}_s^{\text{last}}$ of the remaining pair in the stack becomes the final vector representation of the message.

Transition. The shift-reduce classifier within SPINN calculates the probability of determining actions as follows:

$$\boldsymbol{p}_a = \text{Softmax}(W_{\text{trans}} \boldsymbol{h}_{\text{tracking}} + \boldsymbol{b}_{\text{trans}})$$

During training SPINN, the classifier is trained using ground-truth parsing data. At the time of testing, the internal shift-reduce parser constructs the tree, eliminating the need to prepare a pre-parsed tree as input and removing the dependency on external parsers.

RL-SPINN While SPINN does not require an external parser during testing, it still needs syntactic parsing data during training. To deal with this issue, Yogatama et al. (2017) proposed **RL-SPINN** (**R**einforcement **L**earning **SPINN**). RL-SPINN does not require input syntactic parsing results even during training since its shift-reduce parser learns tree structures via reinforcement learning to optimize the performance on downstream tasks. They showed that the trees induced by this stack-based model captured structures such as noun and verb phrases. We hypothesized that the advantage of this model, which can generate sentence representations leveraging sentence structures without human-annotated data, would positively impact resulting emergent languages.

Signaling Game

It is typical to base the process on the framework of the **signaling game** (Lewis, 1969) in EC.[1] Within the game, there are two types of agents – the sender and the receiver – each with different roles. Let us denote the input set as I and the message set as $\mathcal{M}$. The game proceeds as follows:

1. The sender gets an input $\boldsymbol{i} \in I$, randomly sampled from I.
2. The sender generates a message $\boldsymbol{m} \in \mathcal{M}$ based on the input $\boldsymbol{i}$, and passes it to the receiver.
3. The receiver, based solely on the message $\boldsymbol{m}$, guesses the input that the sender might have gotten and generates an output $\hat{\boldsymbol{i}}$.
4. The success of the game is determined by comparing $\boldsymbol{i}$ and $\hat{\boldsymbol{i}}$; if they match, the game is considered a success; otherwise, it is a failure.

As the sender and receiver repeatedly play the game and learn from it, they become able to achieve high accuracy. At this point, the message $\boldsymbol{m}$ can be said to be a sequence with some common meaning for both agents. In EC, the set of learned sequences is considered language due to its ability to "indicate things", an important aspect of human natural language.

[1] A variant of the signaling game, called *referential game*, is also frequently adopted in EC (Lazaridou et al., 2018; Dessì, Kharitonov, & Baroni, 2021; Ri, Ueda, & Naradowsky, 2023).

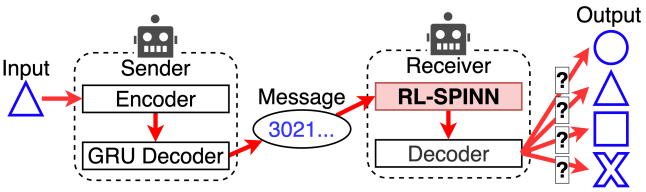


Figure 2: Diagram of the signaling game with RL-SPINN.

Experimental Methods

This section describes an outline of our experimental methods.[2]

Input Set & Message Set Settings

As in many EC works (Ren, Guo, Labeau, Cohen, & Kirby, 2020; Resnick, Gupta, Foerster, Dai, & Cho, 2020; Chaabouni et al., 2020), we use the attribute-value setting for the input set I. Each input $\boldsymbol{i} \in I$ possesses n_{att} attributes, each of which can take on n_{val} distinct values. The size of the input set is given by $|I| = n_{\text{att}}^{n_{\text{val}}}$. For instance, under the configuration of $n_{\text{att}} = 2$ and $n_{\text{val}} = 3$, an example input could be $\boldsymbol{i}_{\text{sample}} = \{\text{att\_1} : 2, \text{att\_2} : 0\}$. Drawing an analogy to the real world, att_1 represents *color* with 0 for *red*, 1 for *blue*, and 2 for *yellow*, while att_2 signifies *shape* with 0 as *circle*, 1 as *square*, and 2 as *triangle*. In this case, $\boldsymbol{i}_{\text{sample}}$ corresponds to a *yellow circle*.

From an implementation perspective, each input $\boldsymbol{i}$ is represented as a concatenation of n_{val}-dimensional one-hot vectors for each of the n_{att} attributes. Consequently, the input $\boldsymbol{i}$ is expressed as a vector of dimension $n_{\text{dim}} = n_{\text{att}} \times n_{\text{val}}$.

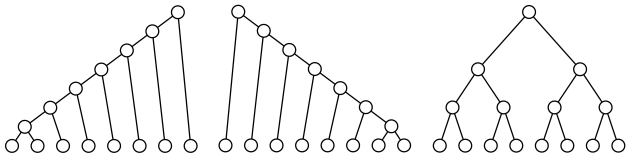
Furthermore, any message $\boldsymbol{m} \in \mathcal{M}$ can be composed as follows. Let there be a finite vocabulary set $\mathcal{V}$; we concatenate a token $v \in \mathcal{V}$ that the sender emits at each time step. Generation of $\boldsymbol{m}$ terminates when the specific token v_{EOS} representing EOS (End Of Sequence) is output, or the length of $\boldsymbol{m}$ reaches the maximum message length κ.

Architectures

We utilize RL-SPINN as a basis of the receiver, as illustrated in Figure 2, using senders and receivers whose details are described below. Note that layer normalization (Ba, Kiros, & Hinton, 2016) is incorporated into both agents in this work.

Sender In our architecture for the sender, we employ the same design as used by Chaabouni et al. (2020). The encoder of the sender transforms the n_{dim}-dimensional vector representation of the input into an $n_{S,\text{hidden}}$-dimensional hidden vector. This output hidden vector is then utilized as the initial vector for the subsequent GRU. This GRU decoder sequentially outputs one token $v \in \mathcal{V}$ at a time. During training, tokens are chosen through sampling from a categorical

[2] The code for our models and experiments is available at https://github.com/porink0424/EmeCom_with_Stack-Based_Agents.



(a) left (b) right (c) complete-binary

Figure 3: Diagram of the tree structures handled by baselines.

distribution with $|\mathcal{V}|$ categories. In contrast, tokens are selected greedily in testing, i.e., by employing a deterministic approach that determines the most likely token at each step.

Receiver with RL-SPINN Upon receiving a message from the sender, the receiver must infer the input that the sender gets. Initially, RL-SPINN generates an $n_{R,\text{hidden}}$-dimensional semantic representation from the message. Note that what Bowman et al. (2016) refer to as D is denoted as $n_{R,\text{hidden}}$ in this paper. After the semantic representation is generated by RL-SPINN, the decoder of the receiver transforms this vector representation into an n_{dim}-dimensional vector, which is then output as the receiver's prediction.

Baselines

In this study, we prepare four baselines: **left-branching**, **right-branching**, **complete-binary-branching**, **random-branching**. Only the generation of shift-reduce actions is manipulated, while the other implementations are consistent.

Left-branching and right-branching deterministically select actions to generate entirely left/right-branching trees. For instance, for a message of length 8, the action sequences generated by left-branching and right-branching, denoted as $\boldsymbol{a}_{LB}$ and $\boldsymbol{a}_{RB}$, would be as follows:

$$\boldsymbol{a}_{LB} = \texttt{SSRSRSRSRSRSRSR},\ \ \boldsymbol{a}_{RB} = \texttt{SSSSSSSSRRRRRRR}.$$

$\boldsymbol{a}_{LB}$ and $\boldsymbol{a}_{RB}$ make trees in Figure 3a and Figure 3b, respectively. Note that the left-branching baseline is almost identical to the conventional LSTM model since tokens are processed in order from left to right.

Complete-binary-branching attempts to generate completely symmetrical trees as much as possible. For example, for a message of length 8, the action sequence would be:

$$\boldsymbol{a}_{PBB} = \texttt{SSRSSRRSSRSSRRR},$$

resulting in forming a tree as Figure 3c. Specifically, complete-binary-branching follows algorithm 1, which determines the shift-reduce actions even for messages whose lengths are not a power of 2 to be as close to complete-binary-branching as possible.

Random-branching is a baseline where actions are sampled uniformly at random, if both `SHIFT` and `REDUCE` are possible. Note that `REDUCE` is impossible if less than two pairs have been pushed on the stack. Likewise, `SHIFT` is no longer possible after the buffer becomes empty.

Algorithm 1 Calculate action sequences in complete-binary-branching baseline

Input: *messageLen*
$A \leftarrow []$
$shiftCount \leftarrow 0$
while $len(A) < 2 \times messageLen - 1$ **do**
  A.append(`SHIFT`)
  $shiftCount \leftarrow shiftCount + 1$
  $tmp \leftarrow shiftCount$
  while $tmp \bmod 2 = 0$ **do**
    A.append(`REDUCE`)
    $tmp \leftarrow tmp \div 2$
  end while
end while
Output: $A[: 2 \times messageLen - 1]$

Optimization

The parameters to be optimized in our model are the sender's parameters $\boldsymbol{\phi}$ and the receiver's parameters $\boldsymbol{\theta}$. Additionally, our model involves the following four types of probability distributions:

- $p_{\text{input}}(\boldsymbol{i})$, the probability of sampling an input $\boldsymbol{i}$ from the input set I.
- $S_{\boldsymbol{\phi}}(\boldsymbol{m} \mid \boldsymbol{i})$, the conditional probability that the sender generates a message $\boldsymbol{m} \in \mathcal{M}$, given $\boldsymbol{i}$.
- $P_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{m})$, the conditional probability that the shift-reduce parser within RL-SPINN generates a sequence of shift-reduce actions $\boldsymbol{a}$, given $\boldsymbol{m}$.
- $R_{\boldsymbol{\theta}}(\hat{\boldsymbol{i}} \mid \boldsymbol{m}, \boldsymbol{a})$, the conditional probability that the receiver generates an inferred output $\hat{\boldsymbol{i}} \in I$, given $\boldsymbol{m}$ and $\boldsymbol{a}$.

Our goal is to minimize the expected value of the cross-entropy loss between the input and the receiver's inferred output, as described below:

$$\mathbb{E}_{\boldsymbol{i}\sim p_{\text{input}}(\boldsymbol{i}),\boldsymbol{m}\sim S_{\boldsymbol{\phi}}(\boldsymbol{m}|\boldsymbol{i}),\boldsymbol{a}\sim P_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{m})}\left[\mathcal{L}\left(\boldsymbol{i}, R_{\boldsymbol{\theta}}(\boldsymbol{i} \mid \boldsymbol{m}, \boldsymbol{a})\right)\right].$$

The gradient of this expected value is:

$$\begin{aligned}&\nabla_{\boldsymbol{\phi}\cup\boldsymbol{\theta}}\mathbb{E}[\mathcal{L}\left(\boldsymbol{i}, R_{\boldsymbol{\theta}}(\boldsymbol{i} \mid \boldsymbol{m}, \boldsymbol{a})\right)] \\ =&\mathbb{E}\left[\nabla_{\boldsymbol{\phi}\cup\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{i}, R_{\boldsymbol{\theta}}(\boldsymbol{i} \mid \boldsymbol{m}, \boldsymbol{a})\right)\right. \\ &\left. + \mathcal{L}\left(\boldsymbol{i}, R_{\boldsymbol{\theta}}(\boldsymbol{i} \mid \boldsymbol{m}, \boldsymbol{a})\right) \nabla_{\boldsymbol{\phi}\cup\boldsymbol{\theta}} \log\left(S_{\boldsymbol{\phi}}(\boldsymbol{m} \mid \boldsymbol{i}) P_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{m})\right)\right].\end{aligned} \tag{2}$$

Now, we define the function f as follows:

$$\begin{aligned}f :=&\mathcal{L}\left(\boldsymbol{i}, R_{\boldsymbol{\theta}}(\boldsymbol{i} \mid \boldsymbol{m}, \boldsymbol{a})\right) \\ &+ \left(\left\{\mathcal{L}\left(\boldsymbol{i}, R_{\boldsymbol{\theta}}(\boldsymbol{i} \mid \boldsymbol{m}, \boldsymbol{a})\right)\right\} - b\right) \log\left(S_{\boldsymbol{\phi}}(\boldsymbol{m} \mid \boldsymbol{i}) P_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{m})\right),\end{aligned}$$

where $\{\cdot\}$ denotes the stop-gradient symbol, and b represents a baseline.[3] In our experiments, the mean-baseline is employed as the baseline. We leverage that the gradient of f

[3]Note that the term "baseline" here does not refer to our baselines, but rather in the context of reinforcement learning.

equals to eq. (2) and use f as a loss function. As the form of f implies, the sender and the parser in the receiver are optimized with the policy gradient method and the receiver's prediction with standard backpropagation.

Furthermore, to promote the exploration in generating messages and shift-reduce actions and to facilitate learning, an entropy regularizer (Williams & Peng, 1991) is used to maintain high entropy. In summary, the loss function used in our implementation is as follows:

$$f - \eta_S \cdot \mathcal{E}_S - \eta_R \cdot \mathcal{E}_R.$$

Here, $\mathcal{E}_S$ and $\mathcal{E}_R$ represent the average entropy during the sender's message sampling and the receiver's action sampling, respectively. η_S and η_R are positive coefficients. Model parameters are trained using Adam (Kingma & Ba, 2015), with a learning rate of γ and an L2 regularization coefficient λ.

Hyperparameter Settings

In the experiments, we use $(n_{\text{att}}, n_{\text{val}})$ configurations of $(2,64), (3,16), (4,8), (6,4)$, as used in Ueda, Ishii, and Miyao (2023). In all these settings, the size of the dataset remains constant at $|I| = 4096$. The entire dataset is split into a $9:1$ ratio, designated as training and test data. The vocabulary size $|\mathcal{V}|$ is set to 4, and the maximum message length κ is set to 8. We set the batch size for each iteration at 5120 and conduct 5000 iterations in a single experiment. The other hyperparameters are set as Table 1.

$n_{S,\text{hidden}}$, $n_{R,\text{hidden}}$, D_{tracking}	500, 500, 300
η_S, η_R[4]	0.5, 0.01
γ, λ	0.0001, 0.0001

Table 1: Hyperparameter settings in the experiments.

Evaluation Metrics

In our study, we measure **communication accuracy (ComAcc)** as an evaluation metric. During each iteration, we calculate the accuracy of the signaling game on the test data. The highest accuracy achieved among all iterations is considered ComAcc. We hypothesize that if the model learns tree structures that better capture the meanings of messages, allowing for flexible vector representation generation for each message, then the ComAcc of our model will be higher than other baselines.

Results and Discussions

Our model obtains higher ComAcc in settings with more attributes.

We run the experiments with 24 distinct random seeds. The results of ComAcc are shown in Figure 4. This section compares and discusses our model with three baselines: left-branching, right-branching, and complete-binary-branching. The results of the random-branching baseline are discussed separately later.

As Figure 4 illustrates, when n_{att} is small and n_{val} is large, the ComAcc score of our model performs not decisively better or worse than the three other baselines. Conversely, in the setting with the highest number of attributes, i.e., $(n_{\text{att}}, n_{\text{val}}) = (6,4)$, the ComAcc score of our model is high.

When n_{val} is large, it can be insufficient only to use a few tokens to express a value for each single attribute. Instead, combining more tokens becomes necessary. This is akin to creating a long word from multiple letters in human language. The situation is such that the hierarchical structure cannot be utilized to a great extent, where our model could not show a high ComAcc. On the other hand, as n_{att} increases, each value can be represented by fewer tokens, but more semantic components (i.e., attributes) have to be conveyed. This is analogous to combining multiple words to understand the overall meaning of the sentence. In such scenarios, our model is considered to show higher ComAcc since it can flexibly change the way to compose tokens for capturing the overall meaning of messages.

We conjecture that RL-SPINN can approximate human sentence processing better in that it learns how to combine tokens to yield the corresponding meaning with more than 2 or 3 semantic components. As an analogous example in linguistics (Aarts, 1997; Carnie, 2007; Radford, 1988; Fillmore, Bach, & Harms, 1968), human language sentences may contain multiple semantic components (e.g., *thematic roles*) such as *agent*, *patient*, *goal*, *source*, and *locative*. In contrast, the baselines with the predefined composition orders may perform to some degree when capturing word-like, relatively flat structures. Experiments with a richer semantics setting may explain this argument more clearly. This aspect should be addressed as part of future work.

$(n_{\text{att}}, n_{\text{val}})$	average tree depth	average message length
$(2,64)$	5.872	7.126
$(3,16)$	5.910	7.215
$(4,8)$	6.162	7.408
$(6,4)$	5.957	7.345

Table 2: Average tree depth and average message length in our model. Values are rounded to the fourth decimal place.

Trees learned by our model closely resemble sequential structures.

We calculate the average of message lengths and depths of trees[5] learned by RL-SPINN, which are obtained from each run at the iteration where the accuracy was adopted as ComAcc. The averages are shown in Table 2. In the setting where $(n_{\text{att}}, n_{\text{val}}) = (2,64)$, the average message length

[4]This hyperparameter value is determined as a result of adjustments within the choices of $[0.5, 0.1, 0.05, 0.01]$, ensuring that the receiver's entropy value does not diverge but ultimately converges around 0 for all configurations $(n_{\text{att}}, n_{\text{val}}) = (2,64), (3,16), (4,8), (6,4)$.

[5]The depth of a tree is defined as the length of the longest path from the root to a leaf node.

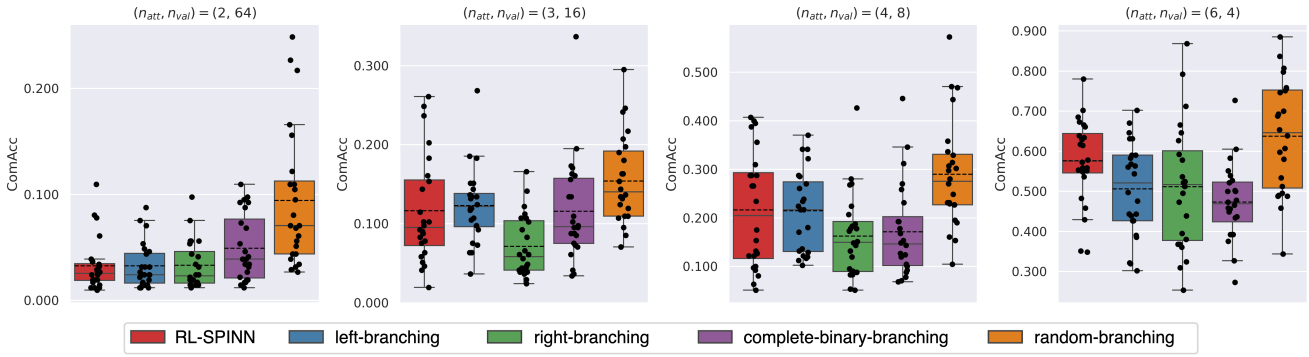


Figure 4: Communication accuracy (ComAcc) in our experiment. The four figures represent the results for $(n_{\text{att}}, n_{\text{val}}) = (2,64), (3,16), (4,8), (6,4)$, respectively, from left to right. Regarding the horizontal axis within each figure, the far left shows the results of our model, while the remaining four represent, from left to right, the results of the left-branching, right-branching, complete-binary-branching, and random-branching baselines. The vertical axis indicates the values of ComAcc. The dotted lines inside the box plots represent the average ComAcc.

is 7.126. It implies that the average tree depth would be $7.126 - 1 = 6.126$ if the tree structures were completely biased towards either left or right-branching. In fact, the average tree depth is 5.872, which is close to the depth expected in a completely biased tree structure. This pattern is also observed in other parameter settings, suggesting that the tree structures learned by RL-SPINN are very similar to those used in left-branching or right-branching.

Random-branching baseline unduly outperforms others.

ComAcc of our model is inferior to the random-branching baseline. Even during testing, the random-branching baseline determines the tree structure randomly for any message, virtually ignoring the order of messages. It is close to the bag-of-words model, which tries to understand messages almost without regard to their order. Although such a language understanding method is far removed from human language comprehension, the unduly high ComAcc of the random model can be attributed to the following two issues in the experimental setup.

First, our experiments adopt the attribute-value format, the de facto standard for input sets in EC. However, the corresponding semantic spaces may not be sufficiently large, and the information can be adequately conveyed even with a bag-of-words approach. In contrast, the information conveyed by humans in reality is more complex and should be incomprehensible without the order of words.

Second, our experiment does not model the incremental nature of human sentence processing. In human language comprehension, the succeeding characters and words are predictable to some extent, given the preceding contexts. Conversely, their unpredictability, called *surprisal* (Levy, 2008), is believed to be a cognitive processing cost for human listeners and can be a crucial factor for forming the word order. Our setting and most previous EC work ignore such a surperisal-theoretic perspective, undesirably allowing the agents to use order-agnostic sequences. Adding the surprisal term to the objective would be an important future direction.[6]

Conclusion

In this work, we leverage a stack-based model called RL-SPINN, which can learn tree structures using reinforcement learning without ground-truth parsing data, as a basis of the receiver agents motivated by the hierarchical nature of human languages. The experiments confirm that our model can generate a language with higher communication accuracy on test data than those of several baselines in a setting that is presumably closer to the semantic space handled by human language.

This research has the following limitations. Firstly, in our experiments, we use an environment setting where the ComAcc of the random-branching baseline, which deviates from human language comprehension, becomes unjustifiably high. Future work could include devising input sets with more complex semantics and incorporating objectives that motivate predicting future tokens, as humans do in incremental language comprehension. Another limitation is that the evaluation is limited only to assessing ComAcc. Experiments and discussions on other metrics are desired. Lastly, our experiment introduces tree-structure inductive bias only on the receiver side. We assume there are two main reasons: the lack of a well-developed syntactic theory on the sender side and the anticipated complexity in implementation and training. Introducing a similar mechanism on the sender side is also a consideration.

This research represents the first study in EC that employs agents mimicking human tree-structured comprehension, potentially suggesting new topics in the field.

[6]Contemporary work (Ueda & Taniguchi, 2023) adopted a variational Bayesian approach to naturally incorporate the surprisal.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP23KJ0768. We would like to express our gratitude to the anonymous reviewers for their helpful feedback and constructive critique.

References

Aarts, B. (1997). *English syntax and argumentation*. St. Martin's Press.

Ba, L. J., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *CoRR*, *abs/1607.06450*.

Bowman, S. R., Gauthier, J., Rastogi, A., Gupta, R., Manning, C. D., & Potts, C. (2016). A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th annual meeting of the association for computational linguistics, ACL 2016, august 7-12, 2016, berlin, germany, volume 1: Long papers.* The Association for Computer Linguistics.

Bowman, S. R., Manning, C. D., & Potts, C. (2015). Tree-structured composition in neural networks without tree-structured architectures. In T. R. Besold, A. S. d'Avila Garcez, G. F. Marcus, & R. Miikkulainen (Eds.), *Proceedings of the NIPS workshop on cognitive computation: Integrating neural and symbolic approaches co-located with the 29th annual conference on neural information processing systems (NIPS 2015), montreal, canada, december 11-12, 2015* (Vol. 1583). CEUR-WS.org.

Carnie, A. (2007). *Syntax: A generative introduction*. Malden, MA: Blackwell.

Chaabouni, R., Kharitonov, E., Bouchacourt, D., Dupoux, E., & Baroni, M. (2020). Compositionality and generalization in emergent languages. In D. Jurafsky, J. Chai, N. Schluter, & J. R. Tetreault (Eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics, ACL 2020, online, july 5-10, 2020* (pp. 4427–4442). Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing, EMNLP 2014, october 25-29, 2014, doha, qatar, A meeting of sigdat, a special interest group of the ACL* (pp. 1724–1734). ACL.

Chomsky, N. (1957). *Syntactic structures*. Berlin, Boston: De Gruyter Mouton.

Dessì, R., Kharitonov, E., & Baroni, M. (2021). Interpretable agent communication from scratch (with a generic visual processor emerging on the side). In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems 34: Annual conference on neural information processing systems 2021, neurips 2021, december 6-14, 2021, virtual* (pp. 26937–26949).

Dyer, C., Kuncoro, A., Ballesteros, M., & Smith, N. A. (2016). Recurrent neural network grammars. In K. Knight, A. Nenkova, & O. Rambow (Eds.), *NAACL HLT 2016, the 2016 conference of the north american chapter of the association for computational linguistics: Human language technologies, san diego california, usa, june 12-17, 2016* (pp. 199–209). The Association for Computational Linguistics.

Elman, J. L. (1990). Finding structure in time. *Cogn. Sci.*, *14*(2), 179–211.

Fillmore, C., Bach, E., & Harms, R. (1968). *The case for case*. Holt, Rinehart, and Winston.

Goller, C., & Küchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of international conference on neural networks (icnn'96), washington, dc, usa, june 3-6, 1996* (pp. 347–352). IEEE.

Hale, J. T., Dyer, C., Kuncoro, A., & Brennan, J. (2018). Finding syntax in human encephalography with beam search. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th annual meeting of the association for computational linguistics, ACL 2018, melbourne, australia, july 15-20, 2018, volume 1: Long papers* (pp. 2727–2736). Association for Computational Linguistics.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings.*

Kuncoro, A., Dyer, C., Hale, J., Yogatama, D., Clark, S., & Blunsom, P. (2018). Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th annual meeting of the association for computational linguistics, ACL 2018, melbourne, australia, july 15-20, 2018, volume 1: Long papers* (pp. 1426–1436). Association for Computational Linguistics.

Lazaridou, A., & Baroni, M. (2020). Emergent multi-agent communication in the deep learning era. *CoRR*, *abs/2006.02419*.

Lazaridou, A., Hermann, K. M., Tuyls, K., & Clark, S. (2018). Emergence of linguistic communication from referential games with symbolic and pixel input. In *6th international conference on learning representations, ICLR 2018, vancouver, bc, canada, april 30 - may 3, 2018, conference track proceedings.* OpenReview.net.

Levy, R. (2008). Expectation-based syntactic comprehension. *Cognition*, *106*(3), 1126-1177.

Lewis, D. K. (1969). *Convention: A philosophical study*. Wiley-Blackwell.

Li, J., Luong, T., Jurafsky, D., & Hovy, E. H. (2015). When are tree structures necessary for deep learning of representations? In L. Màrquez, C. Callison-Burch, J. Su,

D. Pighin, & Y. Marton (Eds.), *Proceedings of the 2015 conference on empirical methods in natural language processing, EMNLP 2015, lisbon, portugal, september 17-21, 2015* (pp. 2304–2314). The Association for Computational Linguistics.

Linzen, T. (2020). How can we accelerate progress towards human-like linguistic generalization? In D. Jurafsky, J. Chai, N. Schluter, & J. R. Tetreault (Eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics, ACL 2020, online, july 5-10, 2020* (pp. 5210–5217). Association for Computational Linguistics.

Marelli, M., Bentivogli, L., Baroni, M., Bernardi, R., Menini, S., & Zamparelli, R. (2014). Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In P. Nakov & T. Zesch (Eds.), *Proceedings of the 8th international workshop on semantic evaluation, semeval@coling 2014, dublin, ireland, august 23-24, 2014* (pp. 1–8). The Association for Computer Linguistics.

Nelson, M. J., El Karoui, I., Giber, K., Yang, X., Cohen, L., Koopman, H., ... others (2017). Neurophysiological dynamics of phrase-structure building during sentence processing. *Proceedings of the National Academy of Sciences*, *114*(18), E3669–E3678.

Radford, A. (1988). *Transformational grammar: A first course* (Vol. 1). Cambridge University Press.

Ren, Y., Guo, S., Labeau, M., Cohen, S. B., & Kirby, S. (2020). Compositional languages emerge in a neural iterated learning model. In *8th international conference on learning representations, ICLR 2020, addis ababa, ethiopia, april 26-30, 2020*. OpenReview.net.

Resnick, C., Gupta, A., Foerster, J. N., Dai, A. M., & Cho, K. (2020). Capacity, bandwidth, and compositionality in emergent language learning. In A. E. F. Seghrouchni, G. Sukthankar, B. An, & N. Yorke-Smith (Eds.), *Proceedings of the 19th international conference on autonomous agents and multiagent systems, AAMAS '20, auckland, new zealand, may 9-13, 2020* (pp. 1125–1133). International Foundation for Autonomous Agents and Multiagent Systems.

Ri, R., Ueda, R., & Naradowsky, J. (2023). Emergent communication with attention. *CoRR*, *abs/2305.10920*.

Socher, R., Lin, C. C., Ng, A. Y., & Manning, C. D. (2011). Parsing natural scenes and natural language with recursive neural networks. In L. Getoor & T. Scheffer (Eds.), *Proceedings of the 28th international conference on machine learning, ICML 2011, bellevue, washington, usa, june 28 - july 2, 2011* (pp. 129–136). Omnipress.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing, EMNLP 2013, 18-21 october 2013, grand hyatt seattle, seattle, washington, usa, A meeting of sigdat, a special interest group of the ACL* (pp. 1631–1642). ACL.

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing of the asian federation of natural language processing, ACL 2015, july 26-31, 2015, beijing, china, volume 1: Long papers* (pp. 1556–1566). The Association for Computer Linguistics.

Ueda, R., Ishii, T., & Miyao, Y. (2023). On the word boundaries of emergent languages based on harris's articulation scheme. In *The eleventh international conference on learning representations, ICLR 2023, kigali, rwanda, may 1-5, 2023*. OpenReview.net.

Ueda, R., & Taniguchi, T. (2023). Lewis's signaling game as beta-vae for natural word lengths and segments. *CoRR*, *abs/2311.04453*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems 30: Annual conference on neural information processing systems 2017, december 4-9, 2017, long beach, ca, USA* (pp. 5998–6008).

Wilcox, E., Qian, P., Futrell, R., Ballesteros, M., & Levy, R. (2019). Structural supervision improves learning of non-local grammatical dependencies. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, NAACL-HLT 2019, minneapolis, mn, usa, june 2-7, 2019, volume 1 (long and short papers)* (pp. 3302–3312). Association for Computational Linguistics.

Williams, R. J., & Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, *3*(3), 241–268.

Yogatama, D., Blunsom, P., Dyer, C., Grefenstette, E., & Ling, W. (2017). Learning to compose words into sentences with reinforcement learning. In *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings*. OpenReview.net.