INTERNAL-CONSISTENCY CONSTRAINTS FOR EMERGENT COMMUNICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

When communicating, humans rely on internally-consistent language representations. That is, as speakers, we expect listeners to behave the same way we do when we listen. This work proposes several methods for encouraging such internal consistency in dialog agents in an emergent communication setting. We consider two hypotheses about the effect of internal-consistency constraints: 1) that they improve agents' ability to refer to unseen referents, and 2) that they improve agents' ability to generalize across communicative roles (e.g. performing as a speaker despite only being trained as a listener). While we do not find evidence in favor of the former, our results show significant support for the latter.

1 INTRODUCTION

Emergent communication is the study of how linguistic protocols evolve when agents are tasked with cooperating with one another. For example, agents might be engaged in a simple object retrieval task, and must learn to communicate with one another in order to indicate which items they want (Lazaridou et al., 2018). To date, work of this type has each agent assume a specific conversational role. For example, an agent might be trained either to speak or to listen but not both (Lazaridou et al., 2018; Kharitonov et al.), or an agent might learn to speak using a vocabulary that is disjoint from the vocabulary it is capable of comprehending as a listener–e.g. speaking only to ask questions (*"what color?"*) and listening only to comprehend the answer (*"blue"*) (Kottur et al., 2017; Das et al., 2017).

These assumptions are misaligned with how we think about human communication, and with the way we'd like computational models to operate in practice. As humans, not only can we shift effortlessly between roles, we further understand that there is inherent symmetry between these roles: we expect others to speak (or listen) similarly to the way we do, and we know that others have the same expectations of us.

In this paper, we test the hypothesis that dialog agents which explicitly incorporate the symmetry between themselves and their communicative partners learn more generalizable representations of language than those which do not. We introduce three modifications to the agents which are meant to enforce that they abide by the "golden rule": speak/listen as you would want to be spoken/listened to. Specifically, these modifications include self-play training objectives, shared embedding spaces, and symmetric decoding and encoding mechanisms that share parameters. We test two hypotheses about the effect of the proposed modifications on emergent communication protocols:

1. Internal-consistency constraints improve agents' ability to generalize to unseen items–e.g. training on *"red square"* and *"blue circle"* and then testing on *"blue square"*.
2. Internal-consistency constraints improve agents' ability to generalize across communicative roles–e.g. training on *"blue"* only as a listener, but having to refer using *"blue"* as a speaker at test time.

We evaluate the effect of each of the proposed modifications using two different reference game datasets and two underlying model architectures, an RNN-based model used by Lazaridou et al. (2018) as well as a Transformer-based model. We find no evidence to support that internal-consistency improves generalization to unseen items (Hypothesis 1), but significant evidence that these proposed constraints enable models to generalize learned language representa-

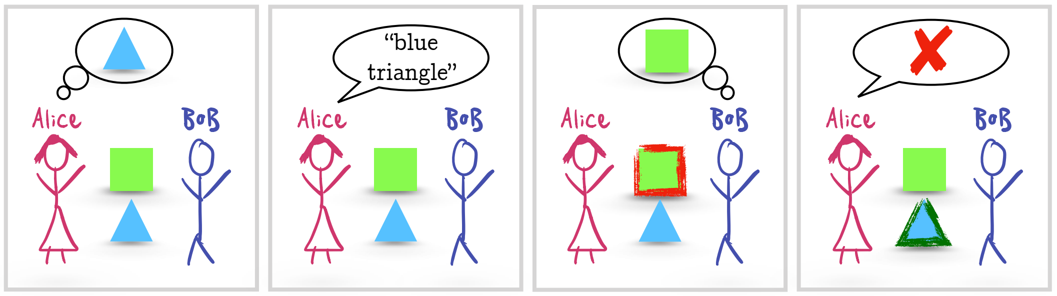


Figure 1: **Reference game.** Alice wants the blue triangle, and asks Bob to get it for her. He selects the green square, and Alice tells him "No" and shows him the item she wanted.

tions across communicative roles (Hypothesis 2), even in the case of where the agent receives no direct training in the target (test) role. All of our code and data is available at `bit.ly/internal-consistency-emergent-communication`.

2 Emergent Communication via Reference Games

Following past work on emergent communication (Lazaridou et al., 2018), we train and evaluate our models in the context of a cooperative *reference game* (depicted in Figure 1). A basic instantiation of a reference game consists of two agents, a set of items (each represented as a vector of `attribute:value` pairs), and a symbolic vocabulary. One agent (the "speaker") has a target item in mind, and must communicate to the other agent (the "listener") which item that is. The game plays out as a simple two-turn action sequence. In turn 1, the speaker sends a message to the listener using a set of symbols from the vocabulary. In turn 2, the listener selects the item that they believe to be the target, based on the message. Afterward, both agents receive a reward if the listener's choice was correct, and no reward otherwise. After training on such a game, we are primarily interested in the properties and the generalizability of the resulting *lexicon*: the mapping from symbols in the vocabulary onto items and/or their attributes.

Implemented naively, agents converge to a trivial lexicon which assigns a unique symbolic name to each item in the training set, e.g., *word1*=[`shape:circle, color:red`] (Kottur et al., 2017). Thus, the challenge in emergent communication is to build agents which learn a lexicon that maps symbols to attributes or other primitive concepts, such that the agents can generalize to novel combinations of attributes at test time. For example, if trained on a set of items such as {[`shape:circle, color:red`], [`shape:square, color:blue`]}, an ideal lexicon would map each attribute-value pair to a unique word (for example, *word1*=`blue`, *word2*=`circle`, etc) such that the agents could successfully refer to an unseen combination ([`shape:circle, color:blue`]) at test time. Prior work has explored a number of methods for enabling models to learn more generalizable lexicons, including constraints on vocabulary size or memory (Kottur et al., 2017), and information minimization (Kharitonov et al., 2019).

Like past work, this paper focuses on finding inductive biases that result in lexicons which can generalize to novel items at test time. In addition, however, we focus on a different type of generalization: generalization across communicative roles. Prior work has tended to treat these roles as distinct, such that an agent may learn to comprehend *"blue"* and *"circle"* perfectly as a listener, and yet have no knowledge of how to refer to [`shape:circle, color:blue`] as a speaker without being retrained from scratch in this role. Thus, in this work, we test whether models that are trained in both roles (either with explicit supervision, or only via self-play) and which strive to be internally-consistent in their behavior across roles, can learn lexicons that not only generalize to novel items but also which can be transferred across roles without the need for retraining.

Notation used in this work. We assume the space of possible references is parameterized by the number of attributes n_f that describe each item (e.g. `color, shape`) and the number of values n_v each attribute can take (e.g. `color`$\in${`red, blue`}, `shape`$\in${`circle, square`}). Each item o is represented by bag-of-features vector $\boldsymbol{o} \in \{0,1\}^N$ where $N = n_f \cdot n_v$. Each index $\boldsymbol{o}_i$ is 1 if d expresses the corresponding feature value. The speaker produces a message with symbols from a vocabulary $\mathcal{V}$ with length L. In all our experiments $|\mathcal{V}| = 100$ and $L = 10$ for a proper comparison to previous work (Lazaridou et al., 2018), as this was their best-performing setting. Symbols in $\mathcal{V}$ are represented as 1-hot vectors.

In each round of the reference game, we construct $\langle \boldsymbol{C}, \boldsymbol{r}, r \rangle$ where $\boldsymbol{C}$ is the context (set of item column vectors stacked into a matrix), $\boldsymbol{r}$ is a vector representing the referent, and r is the index of the referent in the context. We uniformly sample $k - 1$ items as distractors to form the context $\boldsymbol{C} = \{\boldsymbol{o}_1, \ldots \boldsymbol{o}_{k-1}\} \cup \{\boldsymbol{r}\}$. For each round, the context is sampled randomly (even across epochs).

3 Models

We begin with a general architecture and training objective to underly all of our models (Sections 3.1 and 3.2). We then introduce three modifications which can be used to encourage internally-consistent representations: a self-play training objective, a shared embedding space, and a symmetric decoding and encoding mechanism with shared parameters (Section 3.3). Refer to Appendices A.1, A.2 or source code[1] for full implementation details.

3.1 Architecture

In our experiments, agents may both speak and listen. Below, we detail a general architecture that encapsulates all the modules used in the different roles. Note that, in each experiment, there are two distinct agents, one of which is acting as a speaker and the other of which is acting as a listener. We never share parameters *between agents*. That is, when we discuss parameter sharing, we are referring to sharing of parameters between roles *within a single agent*.

3.1.1 Speaker and Listener Modules.

All of our implemented agents contain the following four modules. When speaking, the embeddings and the decoder are used (Figure 2-a); when listening, the embeddings, encoder, and pointing module are used (Figure 2-b). We vary the architecture used for the encoders and decoders, as described in Section 3.1.2.

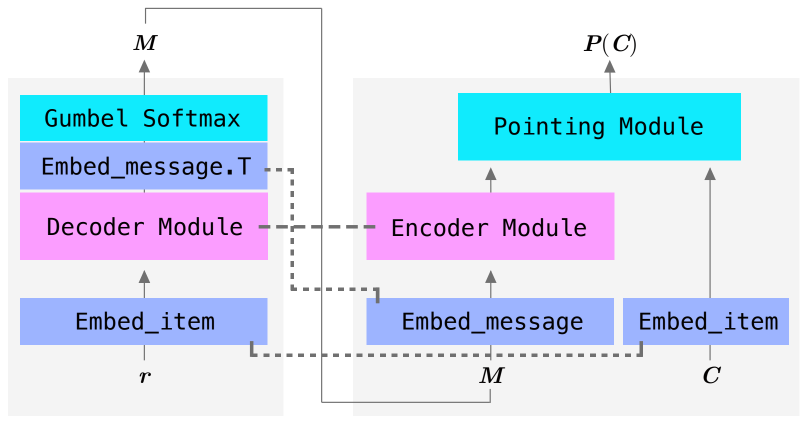
1. **Embedding Matrices** $E_{item} \in \mathbb{R}^{N \times D}, E_{message} \in \mathbb{R}^{|\mathcal{V}| \times D}$. $E_*(x)$ is a linear mapping: $E(\boldsymbol{X}) = E \times \boldsymbol{X}^\intercal$.
2. **Decoder Module**. This module consumes the embedded referent $E_{item}(\boldsymbol{r})$ and produces a discrete message $\boldsymbol{M} \in \mathcal{V}^L$.
3. **Encoder Module**. The encoder consumes the embedded messages $E_{message}(\boldsymbol{M}) \in \mathbb{R}^{W \times D}$ and then produces a representation of the referent $\hat{\boldsymbol{r}} \in \mathbb{R}^D$.
4. **Non-parametric Pointing Module**. This produces a distribution over the items in context by matrix multiplying $\hat{\boldsymbol{r}}$ with the embedded context $E_{item}(\boldsymbol{C})$.

The decoders emit one symbol at a time, auto-regressively producing fixed-length messages. The messages are discretized with the straight-through Gumbel Softmax (Jang et al., 2016) as in Mordatch & Abbeel (2018). This converts a distribution to a one-hot vector while permitting gradients to flow through the operation and enables the agents to be optimized without reinforcement methods. An alternative approach rewards successful classifications and updates the agents separately using REINFORCE, as in Lazaridou et al. (2018). Note that, unlike REINFORCE, Gumbel assumes that listeners receive supervised feedback (i.e. the speaker not only provides response of correct/incorrect, but also points to the intended item). As this assumption is a reasonable for a realistic reference game setting, we choose Gumbel to avoid the problems of instability that comes with using REINFORCE.

3.1.2 Encoders and Decoders

We test both a model with recurrent encoders and decoders and a model with transformer encoders and decoders. See Appendix A for implementation details and hyperparameter selections.

[1] `bit.ly/internal-consistency-emergent-communication`



(a) Speaker Modules. (b) Listener Modules.

Figure 2: **General architecture.** This architecture underlies each model we use; only the implementation of the Decoder and Encoder modules vary between models. In the baseline models no parameters shared within an agent. In shared embedding models, the embeddings (purple) are shared across roles. In symmetric models, the encoder and decoder (pink) are shared across both roles. The blue modules are non-parametric.

Recurrent Model. The recurrent model uses a LSTM (Hochreiter & Schmidhuber, 1997) decoder when speaking and a LSTM encoder when listening, as in (Lazaridou et al., 2018).

Transformer Model. The transformer model (Vaswani et al., 2017) uses a Transformer Decoder when speaking and a Transformer Encoder to encode when listening, as in (Kharitonov et al.).

3.2 Training Objectives

Each agent uses optimizable parameters θ to speak and to listen. Speaking is a function $\mathscr{S} : \boldsymbol{r}, \theta \rightarrow \boldsymbol{M}$ and listening is a function $\mathscr{L} : \boldsymbol{C}, \boldsymbol{M}, \theta \rightarrow P(\boldsymbol{C})$, where $\boldsymbol{M} \in \mathcal{V}^L \subset \{0,1\}^{L \times |\mathcal{V}|}$ is a discrete-valued message with length L, and $P(\boldsymbol{C})$ is a distribution over the items in context. We optimize the parameters θ_A, θ_B of agents A, B over a dataset D to select each referent r from among the distractors in its context $\boldsymbol{C}$. To do this, we minimize the negative log likelihoods of selecting the correct referent in context:

$$\mathcal{L}_{A \rightarrow B} = \sum_{\langle \boldsymbol{C}, \boldsymbol{r}, r \rangle \in D} - \log \mathscr{L}(\boldsymbol{C}, \mathscr{S}(\boldsymbol{r}; \theta_A)\,;\, \theta_B)_r, \tag{1}$$

In loss $\mathcal{L}_{A \rightarrow B}$ Agent A is speaking and Agent B is listening. In our experiments, the speaker modules (Figure 2-a) and the listener modules (Figure 2-b) instantiate the function $\mathscr{S}$ and $\mathscr{L}$ respectively.

3.3 Imposing Internal-Consistency Constraints

We investigate three different internal-consistency constraints, each of which further constrain the model, encouraging internally-consistent representations. Baseline agents consists of two separate sets of parameters, one for listening and one for speaking. For example, the baseline recurrent model consists of two recurrent models. This corresponds to the scenario where agents either speak or listen, but not both (Lazaridou et al., 2018). Similarly, in (Das et al., 2017; Kottur et al., 2017) the agents both speak and listen as part of a two-step communication, but they use distinct modules for each role.

Self-Play Objective. We introduce a self-play loss for agents A:

$$\mathcal{L}_{A\circlearrowleft} = \sum_{\langle \boldsymbol{C}, \boldsymbol{R}, r \rangle \in D} - \log \mathscr{L}(\boldsymbol{C}, \mathscr{S}(\boldsymbol{R}; \theta_A)\,;\, \theta_A)_r, \tag{2}$$

Specifically, in loss $\mathcal{L}_{A\circlearrowleft}$ agent A takes both roles and plays against itself, encouraging it to speak/listen to others the same way it speaks/listens to itself. When we use the self-play training objective, we use it for both agents.

Shared Embeddings. Shared embedding agent uses the same item embeddings and the same message embeddings when listening and speaking. We set shared embedding agents to always use the self-play objective, because otherwise its equivalent to the baseline agent.

Symmetric Encoding and Decoding. Symmetric agents use the same parameters (but different mechanisms) to decode a message when speaking as it does to encoding a message when listening. We set symmetric agents to always use the self-play objective and shared embeddings.

Symmetric Recurrent Model. A LSTM cell is shared between the encoder and decoder. Otherwise, the recurrent model is unchanged.

Symmetric Transformer Model. Transformer Encoders and Transformer Decoders have different structures, so to share parameters between them, we have change either how the transformer agent speaks or how it listens. We opt to replace the Transformer Decoder with Transformer Encoder, and use it to decode messages in-place when speaking. See A.3 for implementation details.

4 Datasets

Our evaluation is based on the simple reference game[2] as described in Section 2, played across two datasets. The datasets, summarized in Table 1, target different aspects of lexical reference. The first, Visual Attributes for Concepts (Concepts) Silberer et al. (2013), is derived from annotated images of actual items (animals, tools, etc). Thus, the data contains realistic co-occurance patterns: groups of attributes like `has-head`, `has-mouth`, and `has-snout` appears together several times, whereas `has-seeds`, `has-mouth`, `made-of-metal` never co-occur. The intuition is that a good lexicon will exploit the structure by developing words which refer to groups of frequently co-occurring attributes (e.g. *"mammal"*) and will describe unseen referents in terms of these primitive concepts. The second dataset, Shapes, is one we create ourselves in order to contrast with the Concepts data. In Shapes, items intuitively correspond to abstract shapes which are exactly describable by their set of attributes (e.g. `blue`, `shaded`, and `hexagon`). All the attributes are independent and there is no co-occurence structure to exploit, so a good lexicon should ideally provide a means for uniquely specifying each attribute's value independently of the others.

Name	**Features** (n_f)	**Values** (n_v)	**Context Size** (k)	**Train**	**Val**	**Test**
Shapes	10	50	5	1000	200	100
Concepts	597	1	5	375	50	85

Table 1: Dataset statistics.

5 Results

We present experimental results aimed at testing the hypotheses previously stated in Section 1. To provide intuition throughout, we frame experiments in terms of two agents, "Alice" (Agent A) and "Bob" (Agent B), who are taking turns asking for and fetching toys.

5.1 Hypothesis 1: Generalization to new items

5.1.1 Setup

We first test whether any of the proposed internal-consistency constraints improve the agents' ability to generalize to novel items–i.e. items which consist of unseen combinations of features. Here,

[2]We use a "context-hidden" game: the context is visible to the listener but not the speaker. This is to ensure that the speaker does not have access to additional attributes (e.g. position of item within the context) that could enable the agents to find degenerate solutions to the game.

we focus on the performance when models are trained and tested *within the same communicative role*. This corresponds to the setting that has typically been used in prior work on emergent communication: Alice always speaks in order to ask for toys, Bob always responds by fetching them, and the pair's success is evaluated in terms of Alice's ability to describe new toys such that Bob correctly fetches them. For evaluation, we hold out a subset of the value combinations from each dataset to use for testing. For example, the agents might be trained to refer to {[`shape:circle, color:red`], [`shape:square, color:blue`]} and then tested on its ability to refer to [`shape:circle, color:blue`]. We compute both validation and test accuracies.[3] The validation items are unseen during training but are used to inform early stopping, whereas the test set is comprised of completely novel items. Unless otherwise noted, numbers reported reflect test accuracy.

5.1.2 Comparison of model architectures

Before introducing our internal-consistency constraints, we first evaluate our various architectures in this vanilla setting. In these experiments, we do not yet use the self-play objective nor do the agents ever act in the opposing roles. Thus, these results serve chiefly to calibrate differences between model architectures. If we see differences between the test conditions, these should be attributed to general architectural advantages associated with implementing these internal-consistency constraints, rather than interpreted as evidence for/against internal-consistency *per se*. Table 2 shows the results. Overall, the transformer architecture outperforms the recurrent architecture, and the transformer outperforms the previously best-reported result for this task (which is only available on the Concepts data). We observe a no clear trend associated with the shared embedding module (sometimes it helps, sometimes it hurts), but do see a slight positive effect associated with the symmetric encoding and decoding mechanism.

	Shapes		Concepts	
	Val	Test	Val	Test
Random (Baseline)	20.0	20.0	20.0	20.0
Prior SOTA (Lazaridou et al., 2018)	-	-	-	81.6
Recurrent Model	56.8 ± 0.8	49.0 ± 1.9	67.9 ± 1.6	70.1 ± 2.6
Transformer Model	86.5 ± 1.5	89.5 ± 1.2	86.2 ± 1.7	87.0 ± 0.7
Symmetric Transformer Model	86.7 ± 1.2	89.3 ± 1.4	92.7 ± 0.7	89.3 ± 1.6

Table 2: Performance when agents are trained and tested in a single role, before introducing any internal-consistency constraints. Results reflect effects attributable to architectural changes associated with implementing our proposed internal-consistency constraints, rather than effects of internal-consistency *per se*. These scores are mean accuracy with 95% confidence range averaged over 5 runs over different test set samplings (the distractors change).

5.1.3 Effects of Internal-Consistency Constraints.

We next look at whether there is an advantage to using the the internal consistency constraints, even when agents remain in fixed conversational roles. Intuitively, this corresponds to a scenario in which both Alice and Bob are capable of performing either role (speaking or listening), but nonetheless, they only ever interact within the same routine: Alice only every speaks and Bob only ever listens and fetches items in response. However, both Alice and Bob can imagine how they would behave if they were to assume the opposite role, and thus, via self-play, each can enforce internal consistency between their own actions as speaker (listener) and the way they imagine they would respond as listener (speaker). In this manner, although Alice and Bob only ever receive direct feedback in one role, they can still impose internally consistent behavior across both roles. It is conceivable that doing so might improve performance even though each remains in a fixed role.

[3]We additionally tried computing Tree Reconstruction Error as introduced by Andreas (2019). See Appendix A.5 for implementation details. However, we did not find an interesting differences across conditions. Therefore, for compactness, we omit these results.

Table 3 shows the effect of adding the self-play objective in the fixed-role setting, across architectures and datasets. The trends are mixed: it appears the additional signal only noises the baseline and symmetric models, whereas the shared embeddings models are able to leverage it effectively. Thus, the effect is not clear enough to establish conclusively that the internal-consistency constraints help the agents generalize in this fixed-role setting, and in fact it may hurt.

	Baseline	+SelfPlay		+Shared Emb.		+Symmetric	
	%	%	Δ	%	Δ	%	Δ
RNN, Shapes	49.0	52.6 ± 0.8	$+3.6$	69.0 ± 2.4	$+20.0$	20.1 ± 1.5	-28.9
RNN, Concepts	70.1	68.6 ± 2.2	-1.5	81.4 ± 1.9	$+11.3$	20.6 ± 1.3	-49.5
Trans, Shapes	89.5	78.7 ± 1.7	-10.8	86.3 ± 1.0	-3.2	87.4 ± 0.4	-2.1
Trans, Concepts	87.0	85.3 ± 1.4	-1.7	89.0 ± 1.3	$+2.0$	90.1 ± 0.4	$+3.1$

Table 3: Performance on task of referring to/fetching unseen items for baseline model compared against models with the internal-consistency constraints. To highlight the difference of each constraint compared to the baseline performance, each delta compares the performance of the modified model to the baseline model, *not necessarily the previous column*. In this setting, in which agents are tested in the same role in which they were trained, we see no clear advantage to enforcing internal consistency via self-play. These scores are mean accuracy with 95% confidence interval averaged over 5 runs over different test set samplings (the distractors change).

5.2 Hypothesis 2: Generalization to new roles

5.2.1 Setup

We now look at whether internal-consistency improves the agents' ability to generalize linguistic knowledge across roles. For example, if an agent is trained to understand the word *"blue"* as a listener, does that agent simultaneously learn when to produce the word *"blue"* as a speaker? Intuitively, we can picture the following scenario: imagine Alice is speaking to Bob, and asks for the *"truck"*. Bob hands her the doll, and Alice replies negatively, indicating that what she actually wanted was the truck. Now, without additional direct supervision, when Bob wants the truck, will he know do use the word *"truck"*? We consider two versions of this setting, involving different levels of supervision as described below. Note that, throughout this section, we still ensure (as above) that test items are disjoint from training items.

Training in one role. Our first experimental setting assumes that Alice and Bob each only receive direct training in one role, e.g. Alice only ever speaks to Bob, so Alice only receives feedback on how she is performing as a speaker, and Bob on how he is performing as a listener. However, both Alice and Bob are able to practice in the opposite role via self play. This setup is analogous to the experiment just discussed in Section 5.1.3. However, unlike before, Alice and Bob will be tested in the roles opposite of those in which they were trained. That is, if Alice was trained as a speaker, then she will be tested as a listener (on her ability to correctly identify items to which Bob refers).

Training in both roles. In our second experimental setting, we assume Alice and Bob enjoy a healthy friendship, in which both take turns speaking and listening to each other, and thus both receive direct supervision in both roles. However, they do not necessarily receive equal training on every vocabulary item. Rather, there are some contexts in which Alice only speaks and other contexts in which she only listens. Intuitively, this corresponds to a scenario in which Alice speaks exclusively about food (while Bob listens), while Bob speaks exclusively about toys (while Alice listens). We are interested in testing how well Alice is able to speak about toys at test time.

We use the SHAPES dataset[4] to create two training splits, each having the same attributes but covering disjoint sets of values. For example, the first training split (`train-1`) might have `color∈{blue, red, yellow}` whereas the second training split (`train-2`) has `color∈{green, orange, purple}`. We use `train-1` to train Alice as speaker and Bob

[4]We cannot construct analogous splits with CONCEPTS since the train set is small.

as listener and `train-2` to train them in the reverse roles. We then report performance with Alice as listener and Bob as speaker using a test set that uses the same attribute values as `train-1`.

5.2.2 RESULTS

Our results for both training conditions are shown in Table 4. The baseline model (which includes no internal consistency constraints) performs, unsurprisingly, at chance. Adding the self-play objective gives dramatic improvements across the board. This result may not be too surprising: by adding the self-play objective, both agents gain the ability to practice in both roles over the space of possible attributes and values. Even so, it has a promising practical interpretation. Namely, if we consider the setting in which a model has access to only a small amount of interaction (e.g. a human may be willing to train a robot via pointing and naming items, but not patient enough to train it by responding to the robot's noisy commands), the ability to massively augment performance via self-play is a significant one.

On top of the self-play objective, we see that enforcing shared-embedding spaces yields further significant performance gains (in the range of 30 percentage points in some cases). Adding the symmetric encoding and decoding mechanism on top of self-play and shared embeddings seems to hurt in general. The performance decrease associated with symmetric decoding and encoding is especially detrimental for the recurrent model in the single-role training setting. Further analysis is required to determine exactly why this is the case.

	Baseline	+Self Play		+Shared Emb.		+Symmetric	
One Role	%	%	Δ	%	Δ	%	Δ
RNN, Shapes	19.8 ± 1.3	50.7 ± 1.3	+30.7	70.7 ± 1.8	+50.9	20.1 ± 1.9	−49.9
RNN, Concepts	21.3 ± 2.3	62.9 ± 0.9	+42.9	81.7 ± 0.8	+18.8	19.6 ± 1.5	−62.1
Trans, Shapes	19.5 ± 0.8	82.1 ± 1.5	+62.6	79.5 ± 0.8	−2.6	83.9 ± 1.8	+4.4
Trans, Concepts	19.6 ± 2.9	52.8 ± 1.7	+32.6	80.3 ± 1.1	+27.5	77.3 ± 2.1	−3.0
Both Roles	%	%	Δ	%	Δ	%	Δ
RNN, Shapes	20.4 ± 1.0	40.1 ± 0.6	+19.7	18.8 ± 1.5	−21.3	69.4 ± 1.9	+50.6
Trans, Concepts	20.9 ± 2.5	61.7 ± 2.2	+40.8	70.3 ± 2.0	+8.6	74.7 ± 1.5	+4.4

Table 4: Performance for task that requires agents to generalize representations across roles–e.g. training on the word *"blue"* only as a listener, but then having to produce *"blue"* as a speaker at test time. "One Role" refers to setting in which agents only ever receive direct feedback in a single role (i.e. their training on the other roles is only in the form of self play). "Both Roles" refers to setting when agents receive direct feedback in both roles, but only see the test vocabulary in the role opposite that in which they are tested. To highlight the additive differences between the internal-consistency constraints, each delta compares the performance of the current column compared to the previous column, *not necessarily the baseline model.* These scores are mean accuracy with 95% confidence range averaged over 5 runs over different test set samplings (the distractors change).

5.3 TAKEAWAYS

Overall, in the case where agents can be trained directly in the role for which they are to be tested, we do not see clear evidence that adding internal-consistency constraints improves the ability of agents to generalize to new items. That is, if Alice is already being trained to speak to Bob, there is no systematic advantage to additionally training by talking to herself. However, internal consistency constrains yield significant performance gains in cases where agents have limited ability to train in a given role. Such a situation may arise, for example, in the case where a human is willing to train a robot by speaking (giving commands), but unwilling to train it by listening (following commands given by the robot). Specifically, models which are equipped with self-play training objectives and shared embedding spaces show clearly superior ability to generalize learned representations across roles, perform about as well as if they had been trained on the target role directly.

6 Related Work

Work in emergent communication (Das et al., 2017; Kottur et al., 2017; Lazaridou et al., 2018) analyzes ways in which computational agents develop a shared protocol via reference games and Lewis Signaling games (Lewis, 2008). Kottur et al. (2017) presented results showing that computational models do not learn compositional protocols by default. Instead, the agents tend to develop brittle protocols which perform well during training but have trouble generalizing to novel situations. Several approaches have been proposed which could encourage models to learn more generalizable representations of language, including compression (Kirby et al., 2015) and efficiency (Gibson et al., 2019), memory constraints (Kottur et al., 2017), and pragmatic constraints (Tomlin & Pavlick, 2018). All of the work just mentioned, like our work, studies the emergence of language assuming agents have access to symbolic representations of referents and their attributes. Other related work looks more specifically at the question of how such representations of attributes are learned from sensory data. Lazaridou et al. (2018) uses a setting very similar to our work, but uses raw pixel data of images rather that vectors of symbolic attributes-value pairs. Das et al. (2017) similarly use raw pixel data to explore the application of emergence protocols for visual question answering. Other work looks out how language understanding arises via goal-oriented cooperation, but in contexts where the agent only listens and follows instructions, rather than fully communicating with the human teacher (Wang et al., 2016; Hill et al., 2017). To our knowledge, our work is the first to explore the effect of internal-consistency on language learning, and to evaluate agents' ability to generalize learned representations across communicative roles.

Our work also relates to a broader body of work on speaker-listener models, specifically pragmatically-informed models in which speakers reason recursively about listeners' beliefs (and vice-versa) (Frank & Goodman, 2012; Goodman & Frank, 2016). Such models have been used in applications such image captioning (Andreas & Klein, 2016; Yu et al., 2017; Monroe & Potts, 2015), and robotics (Vogel & Jurafsky, 2010; Vogel et al., 2013; Fried et al., 2018), as well as in linguistics and psychology in order to explain complex linguistic inferences (Tessler & Goodman, 2016; Monroe et al., 2017). Conceptually, our proposed internal consistency constraints share something in common with these neural speaker-listener models developed outside of emergent communication. However, again, past work has tended to assume that a speaker's mental model of their listener is not necessarily consistent–in fact, it is often assumed explicitly to be inconsistent (Frank & Goodman, 2012)–with the way the speaker themself would behave as a listener. We note, however, that our proposed model architecture (because it lacks the recursion typical in other pragmatics models) are likely unable to handle the types of higher-level inferences (e.g. implicatures) targeted by the mentioned prior work on computational pragmatics, though this is an interesting avenue to explore.

Finally, our work builds off other models that have leveraged self-play for playing competitive games like chess and go have received significant recognition, namely AlphaZero (Silver et al., 2018). An interesting future line of work is to use the ideas introduced by World Models (Ha & Schmidhuber, 2018) to allow each model to hallucinate future interactions, stepping beyond selfplay alone.

7 Conclusion

We propose three methods for encouraging dialog agents to follow "the golden rule": speak/listen to others as you would expect to be spoken/listened to. We evaluate the proposed methods in an emergent communication setting. We find that the internal-consistency constraints did not systematically improve models' generalization to novel items, but that both self-play objective and shared embeddings significantly improve performance when agents are tested on roles they were not specifically trained for. In fact, when trained in one role and tested on another, these internal-consistency constraints allow the agents to perform about as well as if they had been trained regularly.

References

Jacob Andreas. Measuring compositionality in representation learning. *arXiv preprint arXiv:1902.07181*, 2019.

Jacob Andreas and Dan Klein. Reasoning about pragmatics with neural listeners and speakers. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp.

1173–1182, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1125`.

Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2951–2960, 2017.

Michael C Frank and Noah D Goodman. Predicting pragmatic reasoning in language games. *Science*, 336(6084):998–998, 2012.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. *arXiv preprint arXiv:1806.02724*, 2018.

Edward Gibson, Richard Futrell, Steven T Piandadosi, Isabelle Dautriche, Kyle Mahowald, Leon Bergen, and Roger Levy. How efficiency shapes human language. *Trends in cognitive sciences*, 2019.

Noah D Goodman and Michael C Frank. Pragmatic language interpretation as probabilistic inference. *Trends in Cognitive Sciences*, 20(11):818–829, 2016.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Felix Hill, Karl Moritz Hermann, Phil Blunsom, and Stephen Clark. Understanding grounded language learning agents. *arXiv preprint arXiv:1710.09867*, 2017.

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

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

Eugene Kharitonov, Rahma Chaabouni, Diane Bouchacourt, and Marco Baroni. EGG: a toolkit for research on Emergence of lanGuage in Games. *arXiv preprint arXiv:1907.00852*.

Eugene Kharitonov, Rahma Chaabouni, Diane Bouchacourt, and Marco Baroni. Information minimization in emergent languages. *arXiv preprint arXiv:1905.13687*, 2019.

Simon Kirby, Monica Tamariz, Hannah Cornish, and Kenny Smith. Compression and communication in the cultural evolution of linguistic structure. *Cognition*, 141:87–102, 2015.

Satwik Kottur, José Moura, Stefan Lee, and Dhruv Batra. Natural language does not emerge 'naturally' in multi-agent dialog. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2962–2967, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1321. URL `https://www.aclweb.org/anthology/D17-1321`.

Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls, and Stephen Clark. Emergence of linguistic communication from referential games with symbolic and pixel input. *arXiv preprint arXiv:1804.03984*, 2018.

David Lewis. *Convention: A philosophical study*. John Wiley & Sons, 2008.

Will Monroe and Christopher Potts. Learning in the rational speech acts model. *arXiv preprint arXiv:1510.06807*, 2015.

Will Monroe, Robert X. D. Hawkins, N. D. Goodman, and Christopher Potts. Colors in context: A pragmatic neural model for grounded language understanding. *Transactions of the Association for Computational Linguistics*, 2017.

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

Carina Silberer, Vittorio Ferrari, and Mirella Lapata. Models of semantic representation with visual attributes. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 572–582, 2013.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Michael Henry Tessler and Noah D Goodman. A pragmatic theory of generic language. *arXiv preprint arXiv:1608.02926*, 2016.

Nicholas Tomlin and Ellie Pavlick. Incremental pragmatics and emergent communication. In *Emergent Communication Workshop at NeurIPS*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 806–814. Association for Computational Linguistics, 2010.

Adam Vogel, Max Bodoia, Christopher Potts, and Daniel Jurafsky. Emergence of gricean maxims from multi-agent decision theory. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1072–1081, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N13-1127`.

Sida I. Wang, Percy Liang, and Christopher D. Manning. Learning language games through interaction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2368–2378, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1224. URL `https://www.aclweb.org/anthology/P16-1224`.

Licheng Yu, Hao Tan, Mohit Bansal, and Tamara L Berg. A joint speakerlistener-reinforcer model for referring expressions. In *Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

A Implementation Details

We use the deep learning framework Pytorch[5] (v1.2.0) to implement our models (and Python 3.7.3). For reproducibility, all random seeds (random, numpy, torch) are arbitrarily set to 42.

A.1 Recurrent Implementation

The recurrent model decodes and encodes message as follows: to generate a message, the first input is the embedding of a `SOS` start-of-sentence symbol and the initial hidden state is set as the embedded referent (and the cell memory is all zeroes). From here, at each step, the outputted hidden state ($\in \mathbb{R}^D$) is projected by the transposed word embeddings ($E^{\intercal}_{message} \in \mathbb{R}^{D\times|V|}$), and the next word is sampled from this resulting distribution across the vocabulary. Moving forward, the next input is the embedding of the sampled word, and the hidden state and cell memory are set those emitted at the previous step. As mentioned before, we produce words until the maximum length is reached.

When encoding a message, a learned embedding is set to the first hidden state, and the input at each time step is the corresponding embedded word. The last hidden state is set as the encoding. In the symmetric variant of this model, the LSTM cell used for encoding and decoding is the same.

[5]https://pytorch.org/docs/master/torch.html

A.2 Transformer Implementation

The transformer model decodes and encodes message as follows: to generate a message describing the referent auto-regressively, all the embeddings of the words produced so far $\boldsymbol{M}$ and the embedding of a `NEXT` symbol are concatenated together into a matrix $\boldsymbol{X}$ (Eq. 3). Next, the transformer decoder consumes this matrix and the referent embedding and produces a contextualized representation of the input matrix (Eq. 4). The last column vector $\tilde{\boldsymbol{X}}_{:,W}$, which corresponds to the `NEXT` embedding, is the internal representation of the next word. The next word $\boldsymbol{m}$ is sampled from the projection of this representation with the transposed word embedding. More words are produced in this way until the maximum length message is formed. Producing the $i + 1th$ word (so $\boldsymbol{M}$ consists of the first i words), works as follows:

$$\boldsymbol{X} = [E_{message}(\boldsymbol{M}); E_{\texttt{NEXT}}] \in \mathbb{R}^{(i+1)\times D}, \tag{3}$$

$$\tilde{\boldsymbol{X}} = \texttt{TransformerDecoder}(\boldsymbol{X}, E_{item}(\boldsymbol{r})) \in \mathbb{R}^{(W+1)\times D}. \tag{4}$$

$$\boldsymbol{M}_{i+1} = \texttt{GumbelSoftmax}(\tilde{\boldsymbol{X}}_{:,i} \times E^{\intercal}_{message}) \tag{5}$$

To encode incoming messages when listening, all the embeddings of the words in the message plus the embedding of a `ITEM` symbol are concatenated together into a matrix $\boldsymbol{X}$ (Eq. 6). Then, the transformer encoder is used to produce a contextualized embedding $\tilde{\boldsymbol{X}}$ (Eq. 7). The last column vector $\tilde{\boldsymbol{X}}_{:,W}$, which corresponds to the `ITEM` embedding, is set as the message encoding. Note, W is the number of words in each message (and the length of $\boldsymbol{M}$).

$$\boldsymbol{X} = [E_{message}(\boldsymbol{M}); E_{\texttt{ITEM}}] \in \mathbb{R}^{(W+1)\times D}, \tag{6}$$

$$\tilde{\boldsymbol{X}} = \texttt{TransformerEncoder}(\boldsymbol{X}) \in \mathbb{R}^{(W+1)\times D}. \tag{7}$$

$$\hat{\boldsymbol{r}} = \tilde{\boldsymbol{X}}_{:,W} \tag{8}$$

A.3 Symmetric Transformer Implementation

The Symmetric Transformer uses the same mechanism for encoding messages when listening as the default transformer model (described directly above). However, it uses a Transformer Encoder when speaking instead of a Transformer Decode. When speaking, to produce the next symbol, the embeddings of the i words produced so far, the referent embedding, and the embedding of a `NEXT` symbol are concatenated together into a matrix $\boldsymbol{X}$ (Eq. 9). The Transformer Encoder then maps $\boldsymbol{X}$ to a contextualized representation $\tilde{\boldsymbol{X}}$ (Eq. 10). Finally, the column vector in $\boldsymbol{X}$ that corresponds to `NEXT`, is used to sample the next symbol:

$$\boldsymbol{X} = [E_{item}(\boldsymbol{r}); E_{message}(\boldsymbol{M}); E_{\texttt{ITEM}}], \qquad \boldsymbol{X} \in \mathbb{R}^{(i+2)\times D}, \tag{9}$$

$$\tilde{\boldsymbol{X}} = \texttt{TransformerDecoder}(\boldsymbol{X}), \qquad \tilde{\boldsymbol{X}} \in \mathbb{R}^{(i+2)\times D}, \tag{10}$$

$$\boldsymbol{M}_{i+1} = \texttt{GumbelSoftmax}(\tilde{\boldsymbol{X}}_{:,i+1} \times E^{\intercal}_{message}), \qquad \boldsymbol{M}_{i+1} \in \mathcal{V} \subset \{0,1\}^{|V|}. \tag{11}$$

A.4 Hyper Parameters

Search strategy	uniform sampling (25 samples)
Hyperparameter	**Search Space**
optimizer	RMSProp
early stopping patience	100
batch size	64
number of layers	1
hidden dimensionality	choice[16, 32, 64, 128]
learning rate	choice[0.0001, 0.001, 0.01]
scheduler	choice[None, ReduceLROnPlateau, CyclicLR]

Table 5: Recurrent Model Hyperparameter Search Results.

Search strategy	uniform sampling (25 samples)
Hyperparameter	**Search Space**
optimizer	RMSProp
early stopping patience	100
batch size	64
number of layers	1
hidden dimensionality	choice[16, 32, 64, 128]
number of attention heads	choice[1, 2, 8]
dropout	choice[0., 0.2, 0.5]
learning rate	choice[0.0001, 0.001, 0.01]
scheduler	choice[None, ReduceLROnPlateau, CyclicLR]

Table 6: Transformer Model Hyperparameter Search Space.

We uniformly sampled 25 hyperparameter configurations for each model architecture, experiment, and dataset split. In every case, we fixed the hidden size dimensionality and embedding dimensionality to be the same. We searched over three different learning rate schedulers: (1) None (or no scheduler, (2) ReduceLROnPlateau with a patience of 25, reduction factor of 0.1, and uses validation accuracy as its measure of progress, and (3) CyclicLR rising from 0.00001 to the given learning rate over 500 batches and then declines towards 0.00001 for the rest of training (10,000 batches). This is similar to the Noam Update in Vaswani et al. (2017). To save space, we relegate the hyper-parameter selections in our code `bit.ly/internal-consistency-emergent-communication` – see `/lib/hyperparamters.py`.

A.5 Tree Reconstruction Error

TRE takes two important hyperparameters, an error function and a composition function. We select the same choices as the author, for what amounts to the same task (producing a discrete message) as detailed in the original paper. This method requires structured feature representations, so we assume that the features in each item are entirely right-branching. The composition function is learned, and set the number of update steps to 1000.

The original implementation is at `https://github.com/jacobandreas/tre`, and our modification is at `bit.ly/internal-consistency-emergent-communication` – see `/lib/compositionality.py`.