Language emergence in multi-agents communication with a dynamic common ground

Clara Vania

This document summarizes the internship project that I did at FAIR Paris, in collaboration with Diane, Marco, and Eugene. Code is available at: `https://github.com/fairinternal/EGG/tree/common_ground`

1 Introduction

We are interested in understanding how language emerges when deep multi-agents communicate in a human-like environment. In our setup, we have two agents that communicate with each other to solve a particular task. Each agent is equipped with a knowledge base (KB) which represents current state about the world which can dynamically changes over time. That is, as human we have our knowledge about the world which sometimes can change (e.g., weather or hair color). We are interested to study how agents communicate with each other to update their KBs. In this setup, we treat both agents KBs as a common-ground, which stores our knowledge about the world.

2 Game Dynamics

We adopt the Sender-Receiver game, in which one agent acts as the Fact Agent (FA) and the other agent acts as a Question Agent (QA). At each episode, FA will receive a new fact and QA will receive a question which can be about the new fact or any other facts in the KB. FA needs to send the new fact to QA, and upon receiving the new fact, QA needs to update its KB based and answers the question based on the new state of its KB.

Knowledge Base (KB)

The KB itself has the form of a feature vector $\mathbf{f}$ of $|\mathbf{f}|$ where each f_i corresponds to the attribute value of feature i. An attribute can be a property of that object (color of a t-shirt), and we use discrete attribute values. Note that, two different features might share the same attribute value (t-shirt and shoes both have blue color), and two different objects might also share the same attribute-value pair (two people have blue t-shirt).

Agents

To start, first we implement a generic agent, which can act either as a FA or a QA. The current dynamics for the generic agent is as follows:

1. An agent will receive four inputs (more details are below, depending whether the agent is FA or QA): message; fact; question; and a QA/FA flag to determine whether it acts as FA or QA.
2. If the agent is FA, it receives:
    - **message**: a discrete symbol which is then mapped into its embedding, representing message from the other agent. If FA starts the game, at the beginning of the game, it will receives a dummy message (symbol '0').
    - **fact**: an embedding representing the new fact. To produce the new fact, we implement a module that takes an updated KB and the feature slot that is being updated. This way, we push FA to *read* the updated KB and send the new fact (feature, value) to QA. We have three kinds of architectures for this module:
        - *concat100*: concatenation of integer-based KB and 1-hot vector of feature slot.
        - *concat125*: concatenation of continuous KB (we map it using a linear layer) and feature-slot embedding.
        - *concat175*: each feature-value in the updated KB is mapped into its embeddings, and all of them together with the feature-slot embedding is concatenated.

      Note that, based on previous experiments, concat175 always performs better than others and converges faster.
    - **question**: '0' or dummy question, this is mapped into an embedding.
    - **QA/FA flag**: '0' for FA, this is also represented using an embedding.
3. If the agent is QA, it receives:
    - **message**: a discrete symbol, which is then mapped into its embedding. If QA starts the game, at the beginning of the game, it will receives a dummy message (symbol '0').
    - **fact**: an embedding representing the old KB. This is implemented similar as the FA, but instead of composing updated KB and feature slot, we compose the old KB and dummy feature slot (a vector with all zeros).

    - **question**: a symbol representing the question (feature slot in question), can be the same as the fact or different. Mapped into its embedding.
    - **QA/FA flag**: '1' for QA, this is also mapped into an embedding

4. The agent then concatenates the four inputs (we call this as a state), and computes the following:
    - message (only active when the agent is a FA): we feed the state into a linear layer, followed by log-softmax and sampling message
    - answer (only active when the agent is a QA): we feed the state into a linear layer, followed by log-softmax, then argmax, and output a tuple of (question-feature, question-value)

5. We use cross-entropy loss; sampling with REINFORCE is only used for sending messages, and not for getting the answer.

3 Experiments

We start with a simple game, where we use KB of size 2, 3, and 5 and the number of possible values is equal to the number of possible features. We use 8000 samples for training set and 2000 samples for development set. Our game is implemented using EGG toolkit.[1]

3.1 Question is always the same as new fact

To test our implementation, we first run sanity check experiments where we always give question which is the same as the new fact. In this case, QA needs to listen to FA since the new fact might be different with the old fact.

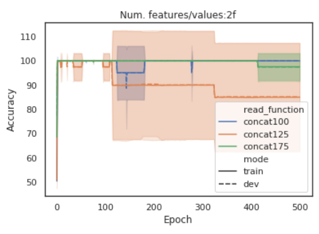
Figure 1 shows learning curves for our experiments with 2, 3, and 5 features. We observe that in all three cases, the agents can communicate quite successfully, with *concat175* shown to be the best function for the agent to "read" its KB.

3.2 Question is not always the same as new fact

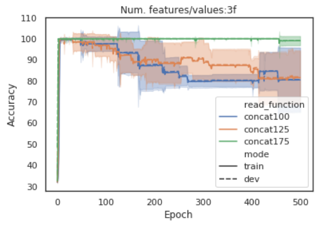
Next, we move to a more difficult setup, where question is not always the same as the fact. To do this, we sample the new fact and the question independently, each from a uniform distribution. Thus, when we use 5 features, there is only 20% chance that question is the same as the new fact.

Figure 2 plots the learning curves for our experiments when question is not always the same as the fact. We observe drops in performance, where

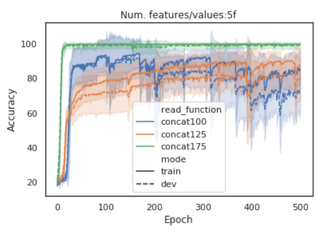
[1]https://github.com/facebookresearch/EGG



(a) 2 features

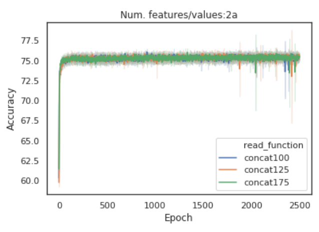


(b) 3 features

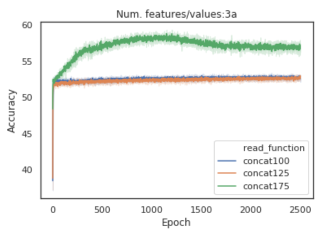


(c) 5 features

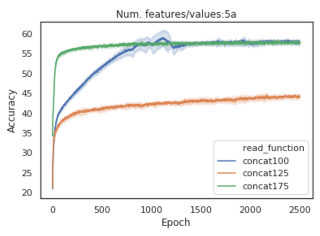
Figure 1: Learning curves of (feature, value) accuracy for 2, 3, and 5 features, *when question is always the same as the new fact.* Shades show standard deviations over 5 different runs.



(a) 2 features



(b) 3 features



(c) 5 features

Figure 2: Learning curves of (feature, value) accuracy for 2, 3, and 5 features, *when question is* ***not*** *always the same as the new fact.* Shades show standard deviations over 5 different runs.

now with 2, 3, and 5 features, the accuracy of the QA drops to 75.6%, 62.6%, and 60.8%, respectively.

4 Error Analysis

To understand our results, we first look at the accuracy of question feature and question value. As seen in Table 1, QA agent can predict the correct question feature with 100% accuracy, and this is expected since this information is given directly to QA. However, it seems that QA is struggle to predict the correct question value.

We then breakdown our results by analyzing the feature and the value of the question and the new fact, which is shown in Table 2. To simplify our analysis, we focus on the game with 2 features. We observe that in this game, QA struggles especially in the case when question is different from the fact, and when they also have different feature values.

	accuracy (%)		
#feat	overall	feature	value
2	75.6	100	75.6
3	62.6	100	62.6
5	60.8	100	60.8

Table 1: Training accuracy when question is not always the same with the new fact.

		overall		$Q_v = F_v$		$Q_v \neq F_v$	
#feat	type	% in training	acc.	%	acc.	%	acc.
2	$Q = F$	49.0	88.3	49.8	88.6	50.2	88.0
	$Q \neq F$	51.0	63.4	50.9	100.0	49.1	25.4
3	$Q = F$	33.8	64.1	32.6	67.2	67.4	62.5
	$Q \neq F$	66.2	61.8	34.6	100.0	65.4	41.6
5	$Q = F$	20.6	51.4	19.6	68.2	80.4	47.3
	$Q \neq F$	79.3	63.3	19.7	64.3	80.3	63.0

Table 2: Training accuracy for the case when we have questions that is not always the same as the fact. Q: question, F: fact, Q_v: question/answer value, F_v: new fact's value, %: percentage in the split data (third column), acc.: question accuracy (feature and value).

4.1 Looking at FA and QA

We investigate both FA and QA abilities to communicate. We first focus on FA and looking at the mutual information between the fact and the messages sent by FA:

- H(fact) = 1.39 — our upper bound if FA doing its job
- MI(message, fact) = 0.86
- MI(message, fact-feature) = 0.19
- MI(message, fact-value) = 0.52

We observe that FA is not always doing its job perfectly, and indeed we found that sometimes FA uses the same symbol to send different facts (more details below). The next question is whether QA listens to FA? Since for question feature it is given directly to QA, we expect that FA's message would be the value of the new fact. To check on QA's ability to listen, we permute messages that FA sends. We found that the accuracy drops from 75.6% to 50.6%, which means that QA does listen to FA.

Message symbol	new KB	new feature	new value	freq
51	0 1	0	0	968
79	1 1	1	1	1014
	1 1	0	1	1005
308	0 1	1	1	1005
591	0 0	0	0	1010
	0 0	1	0	1033
998	1 0	1	0	969
	1 0	0	1	996

Table 3: Mapping from fact to message by FA.

Message symbol	answer feature	answer value	freq
51	0	0	497
	1	0	471
79	0	1	1019
	1	1	1000
308	0	1	518
	1	1	487
591	0	0	981
	1	0	1062
998	0	1	1000
	1	1	965

Table 4: Mapping from message to answer by QA.

4.2 Analysis on messages

We now turn to analysis on the message sent by FA. As a reminder, FA receives fact in terms of new KB and feature slot. We expect that using this information, FA will 'read' the correct value and then send it to QA. We analyze the mapping between fact and message that is sent by FA. In Table 3, we observe that FA uses five symbols to communicate, and except for symbol '998', other symbols can be determine by the new KB and the new value, e.g., '51' is used for KB [0, 1] and new value 0. Next, we look at the message interpreted by QA, what information that the QA access in order to answer the question? From Table 4, we see that given FA message, QA always answer the same value, and for the ambiguous symbol, QA answers it always with '1'.

We then investigate in which case QA fails to answer. One possible reason is because of the ambiguous symbol. We perform ablation study and

symbols	% in data	accuracy
all	100	75.6
w/o ‘998’	75.4	83.6
only ‘998’	24.6	50.9

Table 5: Training accuracy when filtering ambiguous symbol ‘998’.

		overall		$Q_v = F_v$		$Q_v \neq F_v$	
#feat	type	% in training	acc.	%	acc.	%	acc.
all	$Q = F$	49.0	88.3	49.8	88.6	50.2	88.0
	$Q \neq F$	51.0	63.4	50.9	100.0	49.1	25.4
w/o “998”	$Q = F$	49.2	100.0	50.0	100.0	50.0	100.0
	$Q \neq F$	50.8	67.7	67.7	100.0	32.3	0.0
only ”998”	$Q = F$	48.3	51.6	49.1	52.4	50.9	50.9
	$Q \neq F$	51.7	50.2	0.0	-	100.0	50.2

Table 6: when filtering ambiguous symbol ‘998’. Q: question, F: new fact, Q_v: question/answer value, F_v: new fact’s value, %: percentage in the split data (third column), acc.: question accuracy (feature and value).

measure the performance of our model with and without this symbol. We observe that ‘998’ symbol covers around 25% of the data, and filtering this case improves our performance from 75.6% to 83.6% (Table 5). From breakdown analysis in Table 6, filtering ambiguous symbol gives us 0% accuracy in the case when question is different from the fact, and the question value also different with the new fact value. In the other cases, the model achieves perfect accuracies. This results suggest that *QA probably always listen to FA for the given value, and never look at its own KB*. To confirm this, we permute QA’s KB during test and indeed we found that the accuracy did not change.

In Table 6, one might observe that when FA outputs ‘998’, there is never a case when question is different with fact, but they have the same value. To explain this, let us consider the mapping between fact and message on the FA side (Table 7). This particular symbol is used for new KB [1 0] and either fact ‘0,1’ or ‘1,0’, where the first represents the feature and the second represents the value. QA always interpret this as value ‘1’, so it will output, either ‘0,1’ or ‘1,1’. On the other side, given the new KB [1 0] where question is not the same as fact, the correct answer would always either be ‘1,0’ or ‘0,1’. We observe that QA fails because it answers ‘1,1’ instead of ‘1,0’, and for this case given the possible facts, there will never be a case when question is different from the fact but they have the same value.

Besides the ‘998’ symbol, we found 2 other symbols (‘51’ and ‘308’) that

old KB	new KB	fact	sym	gold	ans	freq
0 0	1 0	0,1	998	1,0	1,1	240
1 0	1 0	0,1	998	1,0	1,1	266
1 0	1 0	1,0	998	1,0	1,1	222
1 1	1 0	1,0	998	1,0	1,1	237
0 0	1 0	0,1	998	0,1	0,1	246
1 0	1 0	0,1	998	0,1	0,1	244
1 0	1 0	1,0	998	0,1	0,1	255
1 1	1 0	1,0	998	0,1	0,1	255

Table 7: Incorrect (upper) and correct (bottom) predictions using '998' symbol.

sometimes result in a wrong answer. These symbols represent new KB that has heterogeneous values (both '0' and '1'), and we found that they always fail whenever the new fact value is different with the question value, and this is actually the case when QA needs to look at its own KB.

4.3 How to make QA check its KB?

4.3.1 Changing loss and/or QA architecture

We experiment with two modifications to the current architecture:

- Modify the loss function. In the current architecture, QA outputs a tuple of (question feature, question value). That is, we only provide supervision regarding the correct value. Ideally, to get the correct value, it needs to interpret FA's message, compare the fact feature and the question, and then based on this, output the correct answer. We try to make QA task easier now by also giving it supervision regarding the new fact. So now, QA needs to output two tuples: (fact feature, fact value) and (question feature, question value), where the first tuple is the addition to the current architecture. We call this **full-loss**. For sanity check, we also try for a setup where instead of predicting the answer value, QA just need to output the value of the question in its old KB, i.e., (question feature, old KB value). We call this **sanity-loss**.

- Modify the QA architecture. At the moment, FA has more linear layers to read its KB using the various *concat* function. However, QA doesn't have this since we feed it with a dummy feature. and instead it needs to do feature comparison and KB reading using only a linear layer. We experiment by replacing the dummy feature with question

Loss	QA feature	fact acc.	question acc.
full loss	dummy	100.0 ($\pm$0.0)	75.0 ($\pm$0.5)
	question	100.0 ($\pm$0.0)	75.1 ($\pm$0.4)
sanity loss	dummy	100.0 ($\pm$0.0)	74.9 ($\pm$0.3)
	question	97.5 ($\pm$5.6)	100.0 ($\pm$0.0)

Table 8: Training accuracy for modifying loss function and QA architecture

	train acc.		dev acc.	
num. features	fact	question	fact	question
2	100.0	100.0	100.0	100.0
3	100.0	100.0	100.0	100.0
5	98.2	99.7	76.4	93.1

Table 9: Training and development accuracy for adding one layer to the agent's architecture.

feature. In this case, QA has similar 'power' with the FA to read its KB.

Table 8 reports accuracy when we perform these modifications. Based on the results, it seems that neither the new loss nor the question feature contribute to the improvement of accuracy (top three rows in Table 8).

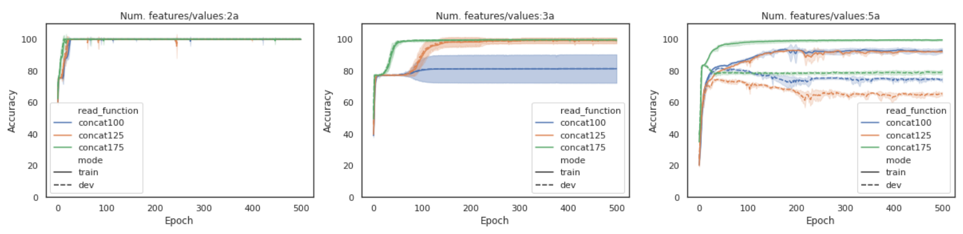
4.3.2 Adding a linear layer to the agent's architecture

Next, we try to modify the agent's architecture by giving it more parameters. Since the difficulty seems come from QA's task comparing the fact from question, we added a linear layer to the agent, from one layer to two with a non-linear layer between them. As seen in Table 9, by adding a linear layer, we now have almost perfect communication accuracy.

Having these results, we want to figure out whether using loss from the question only (one tuple instead of two) can still make the communication between agents works perfectly. So, we again train the model but using only one tuple loss (question feature, question value). Figure 3 shows learning curve during model training and Table 10 presents the final accuracy.

Analysis on messages First, we analyze the communication protocol when the agents succeeded in the game, i.e., average development accuracy over 5 runs achieve > 99.9%. We have these results for the game with 2 and 3 features.

We compute the mutual information between messages and information that are sent by FA. This includes (1) the new updated KB, (2) the fact



(a) 2 features (b) 3 features (c) 5 features

Figure 3: Learning curves of (feature, value) accuracy for 2, 3, and 5 features, after adding a linear layer. Shades show standard deviations over 5 different runs.

	train acc.		dev acc.	
num. features	fact	question	fact	question
2	26.2	100.0	26.2	100.0
3	11.1	100.0	11.2	100.0
5	4.0	99.5	3.8	79.1

Table 10: Training and development accuracy when we add a linear layer.

feature (explicit to FA), and (3) the fact value (implicit to FA). Table 11 presents the statistics for each case which we will explain below. This analysis is conducted on the training data.

- **2 features** For the case of 2 features, we observe that FA always uses four symbols, where each symbol corresponds to a specific KB ('00', '01', '10', or '11'). In other words, instead of sending fact feature and fact value like we hypothesize, FA sends the new updated KB directly to QA. Here, we see that MI(m, KB) = H(KB).

- **3 features** For the case of 3 features, we report two experiments with two different hyperparameters. First, using a bigger model (batch size of 512 and 500 hidden size dimension), denoted with **3-big** in Table 11, we found that FA uses about 29 symbols, where each symbol corresponds to specific fact feature and fact value pair ($\mathrm{MI}(\mathrm{m}, \mathrm{f}) = \mathrm{H}(\mathrm{f}) = \mathrm{H}(\mathrm{v})$). However, manual observation shows that *both updated KB and the fact contribute to the chosen symbol.* An example is shown in the bottom part of Table 12.

  We also check for another experiment when the agents succeeded with 3 features game. This experiment has a slightly lower average development accuracy (98.8%), but 3 out of 5 runs actually achieve 100% accuracy. As seen in Table 11, **3-small**, with a smaller model (batch

Metric	2	3-small	3-big	5
num. symbols	4 (±0.0)	13.2 (±1.79)	29.0 (±2.65)	35 (±7.26)
MI(m, f)	0.00	0.87	1.58	0.41
MI(m, v)	0.51	1.09	1.58	0.27
MI($m, (f, v)$)	1.00	2.38	3.17	1.06
MI(m, KB)	2.00	2.82	2.70	3.55
MI(m, (KB, f))	2.00	3.59	4.43	4.10
H(m)	2.00	3.59	4.43	4.10
H(f) $\equiv$ H(v)	1.00	1.58	1.58	2.32
H(f, v)	2.00	3.17	3.17	4.64
H(KB)	2.00	4.75	4.75	11.3
H(KB, f)	3.00	6.33	6.33	12.50

Table 11: Mutual Information (MI) between messages and information in the training data. Numbers are average over 5 runs (random seeds).

model-size	symbol	new KB	fact feature	fact value
small	341	2**21**	2	1
	341	2**21**	1	2
	341	1**21**	2	1
	341	0**21**	1	2
big	569	1**0**0	1	0
	569	2**0**1	1	0
	569	1**0**1	1	0
	569	0**0**1	1	0

Table 12: Examples of symbol used by models with different capacity.

size of 128 and 250 hidden size dimension), FA uses less number of symbols (13.2). Interestingly, we saw that with a smaller model, the agents develop strategies so that FA can send different feature-value pairs (e.g., '2,1' and '1,2') using the same symbol, whereas with a bigger model, FA always sends the same feature-value pair (e.g., '1,0') with one symbol (Table 12).

On the generalization performance Unlike the 2-feature and 3-feature games, our 5-feature game has unseen examples in its development set. Based on results in Table 10, we see that our trained model could not generalize to these unseen examples. To understand this result, next we analyze our experiments with 5-features game. When we look over 5 different runs, we found 1 run which exceptionally uses way more symbols than the other runs (217 symbols instead of 35 symbols). To better understand the results, we remove this run for now in our analysis.

unseen FA input	seen QA input	% in dev.	acc.
new KB, fact feature	old KB, message, question	6.1	73.0
	old KB	53.2	75.5
	old KB, message	23.6	73.7
	old KB, question	22.1	72.4
	question, message	59.3	76.0
new KB	old KB, message, question	0.5	**90.9**
	old KB	6.8	75.9
	old KB, message	2.9	71.9
	old KB, question	2.6	71.2
	question, message	7.8	76.9

Table 13: Accuracy for cases when FA inputs are unseen but QA inputs are seen.

unseen QA input	seen FA input	% in dev.	acc.
old KB, message, question	new KB, fact feature	33.5	79.6
	new KB	79.6	77.8
old KB	new KB, fact feature	3.0	**88.3**
	new KB	8.3	82.5
old KB, message	new KB, fact feature	18.7	79.4
	new KB	49.6	78.0
old KB, question	new KB, fact feature	23.2	81.4
	new KB	55.3	79.4

Table 14: Accuracy for cases when FA inputs are seen but QA inputs are unseen.

In Table 11, we see that with 5 features, FA uses on average 35 message symbols. We also observe the same pattern across all feature games, where MI(m, (KB, f)) = H(m). That is, FA sends message based on the entire new KB and the fact feature. We observe that for every FA's input that have been seen during training, FA always uses the same symbol. For unseen inputs, FA maps them into symbols, however it's unclear how it choose the symbol (whether it is based on new KB or new fact).

Next, we investigate in which case both agents fail to generalize by looking at the type of information that is unseen by both agents. We divide this based on cases whether FA's input or QA's input are unseen during training. Tables 13 and 14 report accuracy for each of these cases. We do not observe a clear pattern, except that from the highest accuracy achieved, it seems that the combination of both KB and fact/question are difficult to generalize by the agents. We also test in the case when FA's input and QA's input are seen during training (note that this does not mean the combina-

					breakdown acc.	
split	ES	WD	#symbols	overall acc.	Q = F, $Q_v \neq F_v$	others
train	x	x	40	99.7	98.8	99.9
dev	x	x	40	79.8	13.8	92.1
train	v	x	2	84.4	6.4	99.9
dev	v	x	2	83.2	0.6	99.5
train	x	v	1	83.4	0.2	99.9
dev	x	v	1	83.5	0.0	99.8
train	v	v	1	83.4	0.1	100.0
dev	v	v	1	83.6	0.0	99.9

Table 15: Effects of early stopping (ES) and weight decay (WD) on the train and dev performance.

tion of the two are seen during training). This particular case happens 7.1% in the development data, and its accuracy achieves 85.9%, suggesting that our trained model most probably overfits the training data.

Regularization Based on the previous analysis, one thing that we need to address is how to prevent overfitting and push the agents to be able to communicate to achieve perfect communication success (i.e., FA sends fact to QA, QA listens, and answers the question correctly). A simple solution would be using early stopping and/or regularization. We run another set of experiments, where we perform early stopping and weight decay to our model. Table 15 reports our result when adding regularization to the training and development accuracy. In the last column, we further breakdown accuracy on cases when QA needs to listen to FA (that is, when Q == F and $Q_v \neq F_v$), and the other cases. We observe that with regularization, although development accuracy improved, this is because the agents turn to a 'safe choice' strategy which is for QA to just predict output using its own KB (that will give correct answer 80% of the time because of the nature of our dataset, i.e., created via uniform sampling).

5 Conclusions and Future Work

In this project, we have implemented an architecture where agents are equipped by KBs which can dynamically change over time. Our main observation is that the communication that is developed with the current architecture does not really mimic human communication, where agents basically send message that represent entire KB and feature combination, rather than our expected (fact feature, fact value) information. When we experimented with datasets with unseen examples, our agents also generalize poorly, and tend to overfit the training data.

Possible future work:

- Experiment with a balanced training data, to remove the imbalance data effect during training.
- Explore other architectures so that the agents can differentiate their input (KB, fact, or question).
- Use variable-length symbols for messages.
- Multi-episodes game.