Bayesian Few Shot Learning of Compositional Instructions

Rogerio A. Guimaraes Jr. (Undergraduate)
rjunior@mit.edu

Adib Hasan (Undergraduate)
notadib@mit.edu

Abstract

One of the most important human skills in generalizing language is compositionality: once a human learn a new word, they can combine it with other previously known words and generate new sentences that are easily understood. For instance, if a person learns the meaning of *selfie*, he/she is very likely to understand *mirror selfie*, *animal selfie* etc. However, a modern deep learning model would require vast amount of training data to be able to do so. Recently, (Lake, Linzen, & Baroni, 2019) have been able to develop a task that quantitatively showed an implication of this crucial difference between humans and machines. In this paper we shall develop a Bayesian Inference model that will approximate the human data.

Keywords: Bayesian Inference, NLP, Compositional Tasks.

Introduction

Humans are naturally good at generalization from small training data. One critical aspect of it comes from compositionality: humans learn new words from few examples (few shot learning), and also combine new knowledge with prior information, such as, known words, to generate seemingly infinite number of new sequence and patterns that can be easily understood. For instance, *selfie* can be generalized to *mirror selfie* or *animal selfie*.

Recently, there has been a breakthrough in the field of deep learning and natural language processing. However, the most successful models require vast amount of training data, which is contrary to humans, and still fail to generalize beyond the simplest patterns that any human can identify. (Lake et al., 2019)

In order to qualitatively measure this behavior, it is necessary to design tests. However, a simple test such as above (*selfie* to *mirror selfie*) would give unfair advantage to human subjects, since they would understand context of English words, which machine learning models do not have.

To keep test results as neutral as possible (Lake et al., 2019) have designed a test in an artificial language, where every word correspond to either a color or a function of nearby words. This tests attempt to mimic how humans compose known words to compose and understand unknown sequences. They showed that humans are capable of composing and understating sequences that goes beyond provided demonstration. They also showed qualitatively that modern RNN models are unable to do any such generalization.

In this paper, we propose a Bayesian inference model that mimics the thought-process of human subjects and obtains comparable and sometimes better accuracy in the provided tasks.

Related Work

Lake et al., 2019 have performed three experiments to understand instruction learning by humans. The instructions composed of some sequence of artificial words such as *dax, zup, fep, kiki* etc. Each instruction could be interpreted as some number of collinear, colored dots.

Experiment 1: Few-shot instruction learning

This experiment was performed on thirty US based participants. Prior to the experiment, the participants were informed that the study investigated how people learn input-output associations, and that they would be asked to learn a set of commands and their corresponding outputs.

The experiment consisted of four stages. In the study phase of the first three stages, the participant learned individual functions from just two examples each. In the study phase of the final stage, participants learned to interpret composition of these functions from four examples.

During the first three stages, the study instructions always included the four primitives and two examples of the relevant function, presented together on the screen. For the last stage, the entire set of study instructions was provided together in order to probe composition. (Lake et al., 2019)

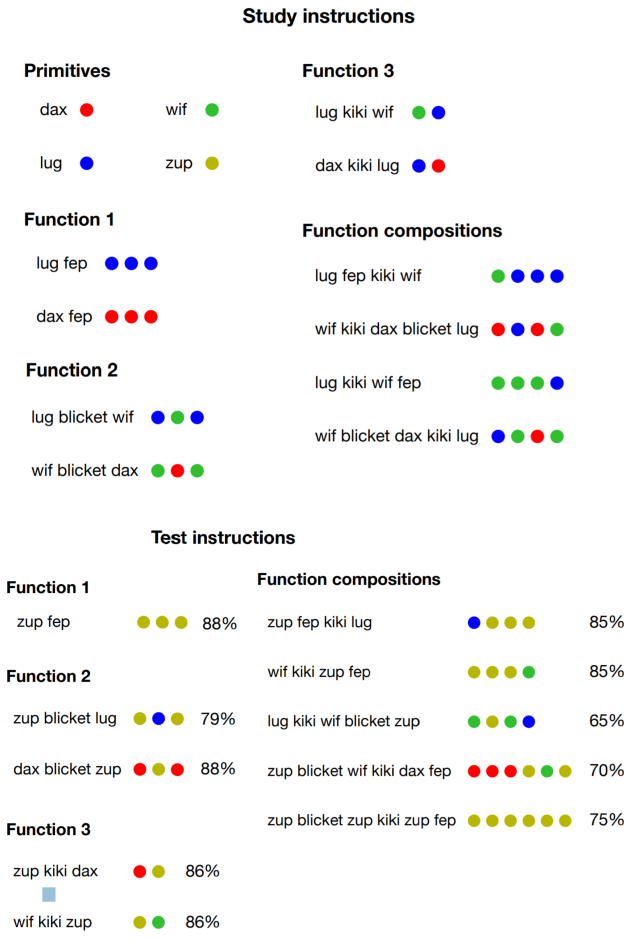


Figure 1: Few-shot learning of instructions in Experiment 1. Participants learned to execute novel instructions by producing a sequence of colored dots. (Lake et al., 2019)

Each of the primitives mapped to one colored dot. The meaning of the functions are the following: Function 1 (*fep*) takes in the preceding primitive as argument and repeats it three times. Function 2 (*blicket*) takes both the primitive before and after it as argument and maps it to a deterministic alternating sequence. For instance, *lug blicket wif* maps to *blue green blue* dot as shown in figure 1. Function 3 (*kiki*) takes in the primitive before and after it and flips their order. The authors also took necessary steps to ensure the test subjects were paying attention and the order in which the functions were presented did not induce a bias. It was observed that $80-90\%$ human subjects were able to understand the single functions and $65-85\%$ were able to understand the function compositions.

The authors also trained various RNN models with the same training data and observed that no model generalized better than 2.5% on average.

Experiment 2: Inductive biases in instruction learning

In previous studies, it has been observed that when human subjects are presented with one familiar object and one unfamiliar object (Markman & Wachtel, 1988), or even two unfamiliar objects (Diesendruck & Markson, 2001) with instructions like "show me the zup", the subjects correspond the unfamiliar object with "zup" as opposed to an alias for the known object (Lake et al., 2019), which is an example of mutual exclusivity bias. In short, humans have biases they follow when trying to parse an ambiguous sentence, and Experiment 2 was designed to better understand that.

This experiment is designed to identify the inductive biases that may have influenced the previous experiment. In particular, the author designed quantitative tests to measure one-to-one, iconic concatenation and mutual exclusivity of the test subjects. The authors of (Lake et al., 2019) defined the these terms as the following:

One-to-one: The assumption that each instruction corresponds to exactly one output symbol.
Iconic concatenation: A preference for maintaining the order of the input symbols in the order of the output symbols.
Mutual exclusivity: This is a classical learning bias that involves refraining from assigning aliases to known items, and in turn assigning one label/word for each item.

The experiment itself is another instruction to color mapping task. However, this time, the instructions were intentionally ambiguous and allowed more than one possible generalization. The choices the subjects made to come up with the generalizations were tested tor the inductive bias.

The test subjects of this experiment were 28 US based individuals. Each individual was asked about fourteen independent trials that measured inductive bias in various scenarios. In each trial, the subject was provided with multiple training examples and was asked to predict the output of only one sequence. The authors ensured that the test conditions were as similar as possible to experiment 1.

The Figure 2 summarizes the the findings from this experiment. Strong evidence for the above three biases were found. While the effect of mutual exclusivity was robust, it was sensitive to context. In particular, people were wiling to override the effect of mutual exclusivity if enough counterexamples were shown.

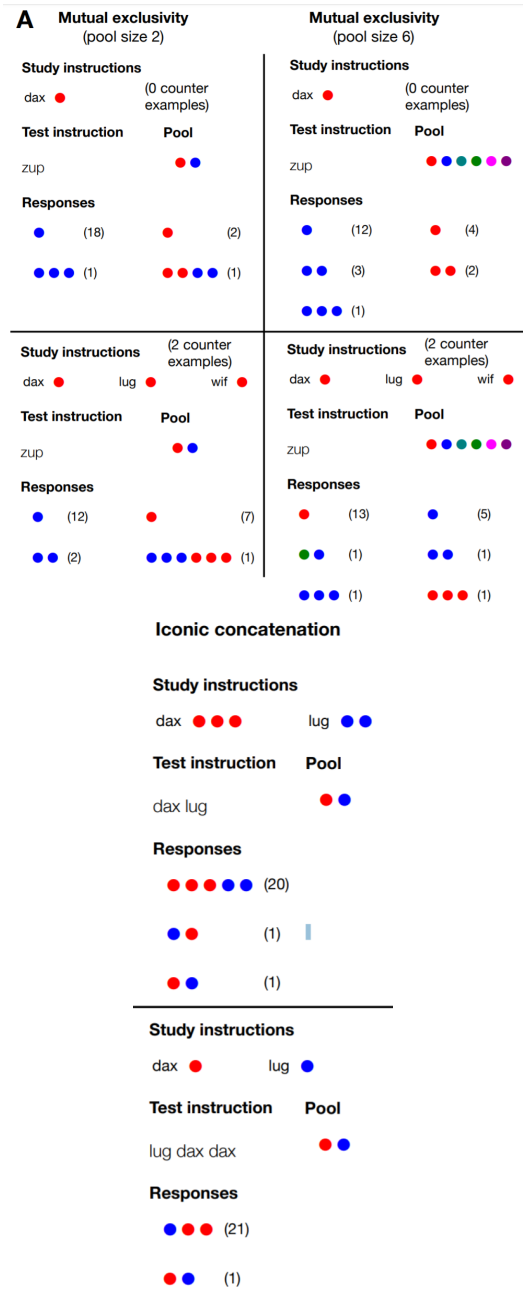


Figure 2: Inductive biases observed in compositional instructional learning experiment. In each case, participants were provided with several study examples and then asked to generate one text example, which was intentionally ambiguous. The results show clear biases towards one-to-one, iconic concatenation, and mutual exclusivity. (Lake et al., 2019)

Experiment 3: Inductive biases in free-form response

In this experiment, participants responded to novel instructions without receiving any demonstrations, e.g., making plausible guesses for the outputs of instructions *fep*, *fep fep* and *fep wif* and how they relate to one another. This design offers the purest examination of people's assumptions since they have no relevant evidence about how to respond.

Methods

We propose the following Bayesian Inference Models. The first model will reproduce the results from experiment 1 and the second model will reproduce the results from experiment 2&3.

Generalized Function Model

This model will be used to reproduce the results of experiment 1. The motivation behind this model was to understand how humans would think about the data. When provided with data, a test subject would understand:

- The functions and primitives are deterministic. Furthermore, Each primitive word map to one colored dot.
- The functions output some combinations of the colored dots mapped from its arguments' primitives.
- The order and length of this combination might vary for each function.
- The behavior of a function might even be different for different words. For instance, *dax fep* outputs three red dots, but *zup fep* might output four yellow dots.

Under these assumptions, we created a concept of generalized functions. To match with the experiments, function 1 takes one argument and multiplies the colored dot its argument points to by an integer between 1 and 4 (inclusive).

Similarly, function 2 and 3 take two parameters each and return some combination of the colored dots its arguments point to. The length of the combination is between 1 and 4. A colored dot may appear more than once, and some colored dot may not appear at all.

All the functions above are allowed to return a different compositions for each value of the argument.

The functions above represent a large number of possible ways to map each sequence. Given the study examples, the model must learn in each case to tune the functions to get the matching outputs. If the inference is correct, we expect the model to learn the true behaviors of the functions (which are not known by the model).

During the composition, it can be seen from the study data that the functions have a precedence order. To account for that, we add a random precedence weight to each function and parse every input string accordingly. This becomes another parameter that the model can learn from training data.

Setup

We implemented both models in WebPPL. (Goodman & Stuhlmüller, 2014)

Implementation of the model for Experiment 1 (M1)

The M1 model has a parser and an evaluator. The parser takes a sequence of instructions as the input. Each individual instruction in the sequence is called a *word*. Each word is mapped to either a function keyword or a primitive keyword. There are three function keywords, namely *function1, function2 and function3*, to match the study dataset and one primitive keyword called *noun*. At first it is decided via a biased coin flipping ($p = 57\%$) whether a word should be a noun or function. If it is a noun, then it is assigned to one of the four available colors with equal probability. If the word is a function, it is mapped to one of *function1, function2 and function3* with equal probability.

```
// map each word to either a primitive (noun)
// or a function
var word_to_term = mem(function(word) {
  var isNoun = flip(0.57)
  var type = isNoun? 'noun' : 'complement'
  //var meaning = isNoun ? sample(Categorical({vs: ['R', 'B', 'G', 'Y']}))
  if (isNoun){
    var times = sample(Categorical({ps: [0.98, 0.01, 0.01], vs: [1, 2, 3]}))
    //var color = sample(Categorical({vs: ['R', 'B', 'G', 'Y']}))
    var color = sample(Categorical({vs: ['R', 'B']}))
    var meaning = times == 1 ? color : (times == 2 ? color + color : color + color + color)
    return {'type': type, 'meaning' : meaning}
  } else {
    var meaning = sample(Categorical({vs: [ 'function1', 'function2', 'function3']}))
    return {'type': type, 'meaning' : meaning}
  }
})
```

Figure 3: Probabilistic function that assigns each word of the input sentence to a term. A term is something with type (primitive or function, called noun and complement in the code in reference to language) and a meaning, which determines its behavior and interpretation. This is a memorized function so that words retain their meanings.

After that, a deterministic function is generated for each function keyword using two randomized function generators. One of them takes one argument, and the other takes two arguments. The function generators work as follows:

First, the generators decide whether the generated function would behave differently for different values of its parameter(s) via a biased coinflip. For instance, *dax fep* returns three red dots, but *zup fep* might return four yellow dots.

If the returned function would behave differently according to coin flip (8% chance), then a random output length between 1 and 4 is chosen for each possible color of its parameter(s). Next, a random combination of parameters of the above length is returned as output. If the behavior is same for all values of it parameters (92% chance), then the process is the same except just one random length between 1 and 4 is selected. As an example, suppose the function generator will return the function $f(x,y)$. Also assume the sampled length is 3, and the random combination is xxy. Once the output is decided, a deterministic $f(x,y)$ is then returned accordingly.

Also note that the function generators memorize the function keywords and will always return same function for the same keyword. Furthermore, no random sampling is done inside the returned function. They are done before constructing this function. Thus, the function that is returned is always deterministic.

Next, the evaluator assigns precedence weight to each function keyword from a categorical distribution. More than one function keyword may have same precedence. The model needs to learn the right precedence order from the data in order to learn the function. Primitive keyword is assumed to have the lowest weight. This is necessary, because applying a function to some parameters returns a sequence of primitives. If primitives had higher precedence than a function, then that function could never been applied.

Finally, we evaluate the instruction sequence (Figure 5. In order to do so, we find the word with highest priority in the sequence and apply the function corresponding to its id. We recursively keep applying functions until the id of the every term of the remaining sequence becomes *primitive*.

The initial mapping that the parser returns can be illegal, for instance it can have only functions without valid parameters. Hence, error handling is performed and the final output of the evaluator is either a sequence of colors or *ERROR*, which signals that the parsing was incorrect. In order to get correct behavior, we condition the output not to be an *ERROR*.

In order to provide the model with study data we use an MCMC based inference model. The model is conditioned on the colors, but for the examples of function outputs and compositions, we use observe on a Bernoulli distribution with $p = 0.95$. This is because the probability space of the model is so large that MCMC is unable to initialize the trace because of the conditions.

```
// evaluate a parsed instruction
var evaluate = function(segment) {
  if(!isValid(segment)){
    return 'ERROR'
  }
  var priority = highest_priority(segment, 0)

  if (priority == 0){
    var concatenation = merge_nouns(segment)
    return {'type': 'noun', 'meaning': concatenation}
  }

  var next_segment = iterate(segment, priority, 0)
  if(next_segment == 'ERROR'){
    return 'ERROR'
  }
  return evaluate(next_segment)
}
```

Figure 4: Function that evaluates the output of a sequence of terms, primitives and functions. First, the function checks for trivial validity of the sequence of terms (it cannot begin or end in functions that require parameters on both sides). Than, it uses a recursive function (since webppl has no loops) to find the term with highest priority in the sequence. Primitives have the lowest priority, zero, thus if the highest priority of a sequence is zero, it only has primitives. Thus, the function concatenates the meaning of the primitives and returns the result. If there are still functions to be applied, the evaluator calls *iterate*, a function that find the term in the sequence with given priority (which we pass the highest) and applies the function defined by that term to the sequence. In the end, the function recursively call itself to repeat the whole cycle. Recursion stops when all functions have been applied and only primitives are remaining. Notice that the code also handles errors for incoherent sequences, which is necessary since the meaning of words might be arbitrary without enough training data.

The main obstacle in designing the M1 model was evaluating the sequence by maintaining precedence order. Since WebPPL has very rudimentary support in this regard, we ended up writing a language parser in WebPPL from the scratch.

```
// translate a sentence of instruction to colored dots
var translate = function(sentence) {
  var segment = parse(sentence)
  var value = evaluate(segment)
  if(value == 'ERROR'){
    return value
  } else{
    return value['meaning']
  }
}
```

Figure 5: Code of translate for an easier understanding of the final dynamics of the code. Translate is the function directly called in a sequence of words to determine the output it produces. First, it parses the sentence (converts a sequence of words to a sequence of terms by calling *parse*, which applies *word_to_term* to all words). Then, it evaluates the sequence of terms to obtain the output sequence generated by the given sentence

Implementation of the model for Experiments 2 and 3 (M2)

The model we used to simulate experiment 2 was basically the same as in the previous experiment, but with two small modifications to account for the bias, for the words representing multiple symbols, and for the errors.

Multiple Symbols In this experiment, as we can see in the third column of Figure 2 (under the other two columns), the study instructions show that ”dax” corresponds to three red dots and ”lug” corresponds to two blue dots. In the previous model, this would not be possible because a sentence of a single word had to be a primitive, and primitives could only mean a single dot of an specific color.

To make the model suitable capable of understanding that a word can mean multiple dots, we changed the random initialization of a primitive. We now sample an integer length from a categorical distribution, all of the same color also sampled from a categorical distribution. We fine tuned the weights to better represent the data, and the optimal weights were 0.98 on length 1 and 0.01 on lengths 2 and 3.

Random Valid Error Another difference was the introduction of random valid error. When the model converts each word to their meanings, it uses the correct meaning with 0.95 probability and a wrong with 0.05 probability.

This is done by converting a word to a another random unseen word with 0.05 probability. In this case, it does not make any difference for words the model could not extract the meaning from the data, but forces the model to sometimes "forget" meanings it had previously learned in the past.

The errors are "valid" because we still condition the entire translation to being a valid translation.

Mutual Exclusivity Principle This bias could be introduced in the model using the function **factor**. With it, we could assign mode weight to interpretations of the words that mapped different words to different meanings.

When we calculated the translation of a sentence, we also count how many "repetitions" a certain translation is implying, which is the number of different words mapping to the same as some other word. We used the code shown in Figure 6 to count this number of repetitions.

Then, we use **factor** to increase the log likelihood of a given translation by:

$$\frac{C_{ME}}{repetitions + 1} \tag{1}$$

In which C_{ME} is a constant. Notice that this formulation accounts for both behaviors present in the data regarding the Mutual Exclusivity Principle:

1. Translations with a smaller number of repetitions will have greater likelihood because, as the number of repetitions grow, that quantity is strictly decreasing.

2. Training data showing contradictory examples of the mutual exclusivity principle lowers the bias the principle induces. Since the difference between $\frac{1}{n}$ and $\frac{1}{n+1}$ decreases as n decreases, the bias differences decrease if the minimum number of possible repetitions increases. When we condition translations on more and more training data with repetitions, we are imposing larger minimum numbers of repetitions.

Results

Experiment 1

For the function input, we ran each model 5 times with 100,000 samples each. And for the function compositions, we ran the model 3 times with 200,000 samples each. In this case, the model takes roughly 15-20 minutes for each run, hence we were unable to run each test 5 times. Our findings are presented in Table 1.

```
var allWords = (['dax', 'lug'])
var allTerms = map(word_to_term, allWords)

var equiv = function(i, j) {return allWords[i] != allWords[j] && isEqual(allTerms[i], allTerms[j])}
var length = allWords.length
var idxs = mapN(function(x) { return x }, length)
var repeatedGroups = groupBy(equiv, idxs)
var repetitions = length - repeatedGroups.length

factor(4.5/(repetitions+1))
```

Figure 6: Code snippet showing how we implemented the bias of the mutual exclusivity principle in our model. We first need to write all words used in training and testing in *allWords*. Then we translate it to the true meanings of the words decided by the model and use *groupBy* to group different words with same meanings.
Counting the number of such groups will let is easily calculate the number of repetitions, based on the total number of words present in the training and test data.

Table 1: Comparison of human accuracy (pcnt.) and the M1 model accuracy (pcnt.) for various test sequences in experiment 1.

Test sequence	**human**	**M1**
zup fep	88	93.29 ± 0.79
zup blicket lug	79	74.94 ± 2.48
dax blicket zup	88	72.25 ± 4.41
zup kiki dax	86	85.07 ± 4.17
wif kiki zup	86	84.01 ± 0.90
zup kiki fep lug	85	95.65 ± 6.61
wif kiki zup fep	85	95.63 ± 1.13
lug kiki wif blicket zup	65	93.16 ± 3.40
zup blicket wid kiki dax fep	70	85.25 ± 6.33
zup blicket zup kiki zup fep	75	92.05 ± 6.88

Figure 7 and 8 shows some of the probability distributions we found for our model. In can be seen that the probability distributions are very easily interpretable and even the incorrect outputs might be found very easily from human subjects.

Furthermore, it can be seen that the M1 model approximately matches the human accuracy for the individual functions, and does better than human subjects in compositions. It must be noted that there precedence weight of the functions in the data was not hard-coded in the model. It rather learned it on its own.

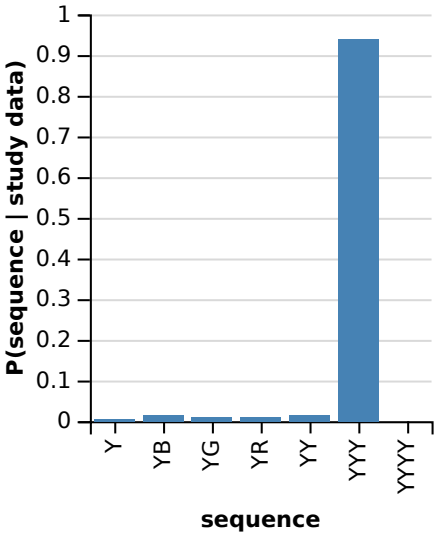


Figure 7: Probability distribution generated by M1 for the sequence *zup fep* showing over 90% probability for the output to be *yellow yellow yellow*. The letters *R, B, G, Y* correspond to red, blue, green and yellow, respectively. The other non-zero bars represent the outputs represent *fep* as a function with various length between 1 to 4, and *fep* as a primitive.

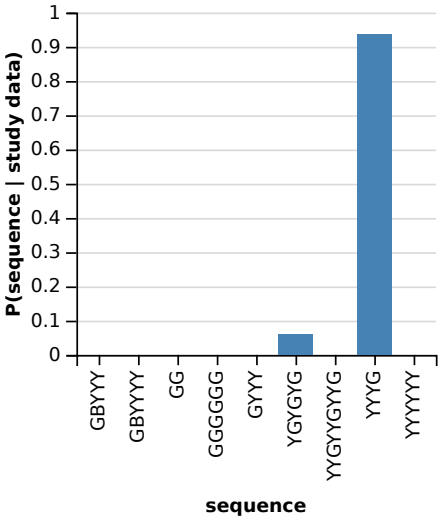


Figure 8: Probability distribution generated by M1 for the sequence *wif kiki zup fep* showing over 90% probability for the output to be *yellow yellow yellow green*. The letters *R, B, G, Y* correspond to red, blue, green and yellow, respectively. The other notable non-zero output is *yellow green yellow green yellow green*, which can occur when the functions are correctly learned, but applied in the wrong order.

Experiment 2

For experiment 2 we could only satisfactorily reproduce results from the first and third columns. We used the M2 model with 100,000 samples using MCMC. Here is the data obtained by the model compared to the human data found in the original study:

Table 2: Comparison of human most predicted outcome (pcnt.) and the M2 model most predicted outcome (pcnt.) for various test sequences in experiment 2. The most predicted outcome was the same for humans and the model in all cases (and can be seen in Figure 2, with differences only in percentage of that answer with respect to all answers. obs.: c.e. stands for counter examples

Test sequence	**human**	**M2**
Mutual exclusivity (0 c.e.)	81	86
Mutual exclusivity (2 c.e.)	55	57
Iconic concatenation (0 c.e.)	91	83
Iconic concatenation (2 c.e.)	95	91

Experiment 3

The data showed by the authors of the original experiments for experiment 3 are just two examples of responses from the subjects. It was only intended to show the biases existing, and previously mentioned, when assigning new words to meaning.

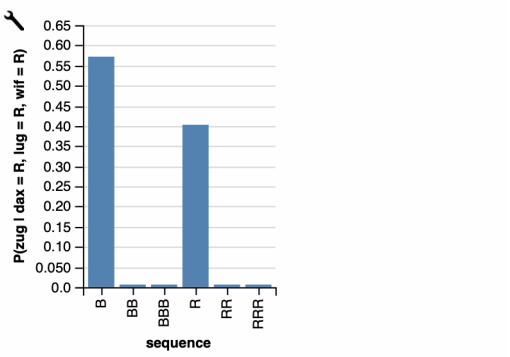
The model was very similar to the first example, and Table 3 shows the percentage of outcomes of the model that were similar to what the human subject produced.

Language Model Package

We also consider that the development of the probabilistic language model itself is also one of the greatest contributions of our work. The final version of the model in webppl will be submitted alongside this paper.

The model can learn from a sequence of strings to another and can be used in the following way:

1. In function *makeModel*, assign sentence to the sequence you want the model to predict
2. In the distribution *dist*, assign *allWords* to a list of all elements used in either training or test sequences




```
P(zug = B | dax =  R, lug = R, wif = R) = 0.5728

P(zug = R | dax =  R, lug = R, wif = R) = 0.4039

P(zug = BB | dax =  R, lug = R, wif = R) = 0.005
```

Figure 9: Probability distribution generated by M2 for the sequence *zup*, given that *dax, lug*, and *wif* are red dots, showing over around 60% probability for the output to be *blue*, and around 40% chance of being red, which is extremely similar to human data

Table 3: Comparison of human most predicted outcome (pcnt.) and the M2 model most predicted outcome (pcnt.) for various test sequences in experiment 3. The most predicted outcome was the same for humans and the model in all cases (and can be seen in Figure 2, with differences only in percentage of that answer with respect to all answers. Each step was calculated conditioning on the previous choices of the model

Test sequence	**M2**
fep	98
fep fep	96
zup fep	75
fep wif	65
fep dax fep	37
kiki dax fep	77
fep dax kiki	98

3. In the distribution *dist*, provide the training data is conditions. If sequence s_1 generates sequence $s2$, provide: condition(translate(s_1) == s2)

The model allows tuning of the following parameters, among others:

- bias of on type of each word (primitive or function)
- frequency of random error
- bias of Mutual Exclusivity Principle
- atomic pieces allowed for the construction of the output sequence, and the model bias for using each of them
- hypothesis class for the generation of any given function

We want to publish our model as a webppl package and we believe it can assist others in further investigating the use Bayesian models for seq2seq problems, especially in language domain.

Collaboration Policy

Both author worked equally on implementing and testing the models. Specific contribution would be the following:

Adib Hasan

1. Implemented probabilistic general functions for model 1.
2. Added code for error handing for the models.
3. Identified and implemented relaxation conditions for randomized output.
4. Tuned the hyperparameters for M1.

Rogerio Junior

1. Implemented the compositional language parser that is able to read words and convert to output sequences following the types and hierarchy defined by the probabilistic general functions.
2. Augmented the model of experiment 1 (M1) to generalize to experiments 2 and 3 (M2)
3. Tuned the hyperparameters for M2.

Acknowledgments

We would like to thank prof. Joshua Tenenbaum and stuff members of 6.804 for their continuous support throughout the semester.

References

Diesendruck, G., & Markson, L. (2001, 10). Children's avoidance of lexical overlap: A pragmatic account. developmental psychology, 37(5), 630-641. *Developmental psychology*, *37*, 630-41. doi: 10.1037//0012-1649.37.5.630

Goodman, N. D., & Stuhlmüller, A. (2014). *The Design and Implementation of Probabilistic Programming Languages.* `http://dippl.org`. (Accessed: 2019-12-15)

Lake, B. M., Linzen, T., & Baroni, M. (2019). *Human few-shot learning of compositional instructions.*

Markman, E. M., & Wachtel, G. F. (1988). Children's use of mutual exclusivity to constrain the meanings of words. *Cognitive Psychology*, *20*.