Methods in AI Research - Report

Tycho Buisman, Kevin Jetten, Fleur Petit, and Saskia van der Wegen

November 11, 2018

1 Introduction

This report discusses the development of a task-oriented dialogue system. The system aims to operate adequately within the domain of restaurant recommendation. In order to provide this service, the system should use various methods within Artificial Intelligence (AI) research.

Firstly, the dialogue system should be able to classify various speech acts, in order to determine the intention of the user.

Subsequently, the semantic structure of a sentence should be derived by means of combinatory categorical grammar (CCG). This would make it possible to extract user preferences from the user utterance.

Consequently, the knowledge of the system should be updated, and an accurate response should be generated.

In the following sections, we will discuss these various components of the system in detail.

2 Dialog state transition model

A state transition diagram has been derived from a large set of example conversations between a user and a restaurant recommendation system. The result can be seen in Figure 1.

The blue rectangles indicate the actions of the system. The pink rounded rectangles indicate the actions of the user. When the system is in a state that is represented by a diamond, the next action of the system will depend on the answer to the question in the diamond. Some example sentences for each kind of action are shown in Table 1.

As can be seen in Figure 1, the system starts with a welcome message and an information request. It enquires about the user's preferences regarding the location, food type, and price range of the restaurant. The consequent user utterance can be an informative act, providing the required information, or one of the the other speech acts listed in Table 2.

When the system has enough information, it suggests a restaurant, otherwise, it asks for more information. There is one exception to this procedure: when there are no restaurants available that meet the preferences of the user,

Welcome + request	Hello , welcome to the Cambridge restaurant system? You can ask for restaurants by area , price range or food type . How may I help you?
Reply with food/area/price	expensive restaurant in the south part of town
Other	hello welcome
Request food/price/area	What part of town do you have in mind?
	Would you like something in the cheap , moderate , or expensive price range?
No options	Sorry there is no corsica restaurant in the centre of town
Suggest	la tasca is a nice place in the centre of town serving tasty spanish food
Request other restaurant	anything else
Request info restaurant	uh could i have the phone number
Provide info	The phone number of la margherita is 01223 315232 .
Thank you, goodbye	okay cool uh good bye thank you

Table 1: Examples of actions in the state transition diagram

the system will inform the user of this fact. The example data showed that in this case, the user provides alternative preferences.

After the system suggested a restaurant, the user can request contact details of the restaurant, request confirming information, or request an alternative. The system will respond with the requested information. The dialogue ends when the user is content with the suggested restaurant, and the provided details. The user will then close the conversation, possibly by a word of thanks or goodbye.

3 Discussion Machine learning

3.1 The machine learning task

A long-short term memory network (LSTM) was trained in order to categorise user utterance into speech act classes. Each user utterance can belong to one of 15 different speech act classes. The different classes can be seen in Table 2 [3]. In order to achieve this, a model has been constructed with the aid of the Keras package [2], Tensorflow backended [1].

3.2 Data preprocessing

Initially, the training data consisted of a large set of pre-labeled user utterances. These utterances and labels needed to be transformed to numerical data. A neural network needs numerical input and output in order to be able to compute a numerical error for back-propagation.

Furthermore, all user utterance need to be the same length, in order to fit a fixed number of input nodes. We choose a length of 23; the length of the longest sentence in the data set. All sentences that did not reach this length, have been padded with zero's. All other words in the vocabulary have been assigned a random integer.

ack	acknowledgement
affirm	affirmation
bye	ending the dialogue
confirm	confirm understanding
deny	deny
hello	greeting the system
inform	user specification of goal
negate	negation
null	something not understandable for the system
repeat	request to repeat
reqalts	ask for alternative resaurants
reqmore	ask for more information
request	request for specific information
restart	restart the conversation
thankyou	thanking the system

Table 2: Speech acts

The speech acts (Table 2) represent different categories. In the encoding of the speech acts to numerical values, we tried to preserve their categorical character as much as possible.

Integers have certain relations to each other that are not representative of categorical values. That is, integers can have a small difference, or a large difference, they can be bigger than another integer or smaller. This makes them suitable for coding continuous variables, but not so much for categorical values.

Instead of assigning an integer to each speech act, we choose to use dummy coding. We use numerical vectors of length 15. 14 slots of the vectors are filled with zero's. For each individual speech act another slot is filled with a 1.

3.3 Design/lay-out of the LSTM network

3.3.1 The first layer: word embedding

In the previous subsection, we discussed why integers are not best-suited to represent the properties of categorical variables.

How about the words of a user utterance? Important properties such as relations between words and semantics, are lost when each word is mapped to a random integer.

A hint on how to tackle this issue, comes from the distributional hypothesis. The distributional hypothesis states that "words which are similar in meaning occur in similar contexts" [5]. Somehow, we may preserve a bit of the meaning of each word by making the numerical encoding representative of the context each word occurs in. I.e. we somehow want to account for similarities between words, by looking at the words that surround them.

Word embedding provides a way to implement the general idea that is expressed by the distributional hypothesis. We use the Keras [2] function "Embedding" in order to turn positive integers (that represent words) into dense vectors of a fixed size.

Initially, each integer is assigned a random vector. The vectors are adjusted via back propagation. Given the surrounding words, the network has to predict the vector that corresponds to the current word. The difference between the output vector from the network, and the actual vector that corresponds with the current word, is used to adjust the vector. After several iterations, the surrounding words can be used to predict the vector of the current word.

3.3.2 The second layer: LSTM

The second layer in the model is a Long Short Term Memory layer. Just as the name indicates, a Long Short Term Memory Network uses information of previous states in order to update the current state.

In this manner, information from earlier words in the user utterance can be used to predict the label of the sequence, instead of merely using the last word or an n-gram of the last couple of words.

In order to compute activation of the output layer, we use a Rectified Linear Unit (ReLu) activation function. The activation of the output nodes is calculated by means of this function. Characteristic of the ReLu function is that it's output of a node is 0 when the total, weighted, input of the node is a negative number [7].

3.3.3 The third layer: dense layer

The third and last layer is a dense layer. The dense layer connects all the output nodes from the LSTM to the output nodes of the model. The activation function that is used in the Dense layer is "softmax". Softmax is an exponential function that normalises the activation of the output nodes to a value between 0 and 1 [6].

3.4 Number of output nodes

In order to find out which parameters would generate the best results, we tried several different combinations for the number of output nodes for the embedding layer, and the number of output nodes for the LSTM layer.

We tested all different combinations of the arbitrary numbers 128, 64, 32 for the two variables.

For two variables, this results in a list of 9 possible combinations. At first, we tested the different settings on 20 epochs, and looked at which setting yielded the best results. The optimal setting turned out to be 128 output nodes for the embedding layer, and 64 output nodes for the LSTM layer. Figure 2 shows the accuracy and the loss plotted over the number of epochs for this model.

It can be noted that the accuracy does not improve much after 10 epochs. Therefore, we choose to use these parameter settings with 10 epochs for our final model. These settings yielded an accuracy of .9886 on test set, and .9784 on training set.

3.5 Problematic speech acts

Several utterances are classified incorrectly by the model. Most notably are combinations of other speech acts with an "inform" act.

For example "thankyou", "bye", or "hello" followed by "inform" are natural utterances to occur in an information request dialogue. For example in response to the first system utterance: "Welcome to the restaurant system. You can ask for restaurants by price, area and the type of food. What would you like?", a user may want to say something in line with: "Hello. Could I please have Chinese food in the south part of town?".

The system will classify the utterance in accordance with the first act occurring in the utterance, which is "hello". Therefore, it fails to extract the information that is provided in the latter part of the user utterance.

The problem could be solved either during preprocessing the data, or later in parsing the user utterance. In the training set, expressions like the user utterance in the example are classified as "hello" even though they contain a "inform" act as well. An option to solve the issue, would be to change the labels of those utterances in the training set to "inform".

Another possibility is to intervene at a later point in the system, when the sentence is parsed. We could split "inform" utterances, from other utterances, to make sure that useful information is extracted.

4 Combinatory Categorial Grammar

4.1 Background and examples

CCG is a method used to derive the semantic structure of a sentence. By assigning certain types to words, a grammatical valid sentence can be constructed and deconstructed. The type of the word specifies with what kind of types the word can be combined. For example, an adjective (n/n) needs a noun (n) to its right to form a new noun.

In the target-oriented dialogue system, CCG types may aid the system in extracting the necessary information from user utterances. By assigning types to the different words in the sentence and applying rules of combining the types, the whole sentence can be deconstructed. An example is provided in Figure 4. The tree shows the deduction of the original sentence into sub-sentences and finally to the word level. You can see how different word-types and sub-sentences can be combined to create a new type. For example, if the adjective "world" (n/n) is provided a noun (n) to the right, in this case "food", it will create the new noun: "world food".

4.2 Implementation of the deduction algorithm

4.2.1 High level pseudocode

The input text is first checked for words that do not occur in the vocabulary. Words that do not occur are replaced by the first word in the vocabulary with the shortest Levenshtein distance to this word. The distance is computed with the `levenshtein` function from the `pylev` package [4].

An object of the class `Phrase` is initialised for each word. The attribute `text` is defined by the current word in the sentence. The `type` 1 to 3 attributes are defined by the different types of each word.

The `Phrase` object of each word is added to a list of `Phrase` objects. The list of phrases is put as the first element of a queue. While this queue is not empty, we pull the first list of phrases from the queue (`phraseList` in pseudocode) and attempt forward and backward elimination with the types of each `Phrase` object in the list. The list of phrases decreases in length as a result of the eliminations. When the lists of phrases is of length one, the parsing has succeeded.

Algorithm 1 Deduction Algorithm

```
phrases ← empty list
phraseListsQueue ← empty queue
finishedParsing ← empty list
for word in inputText do
    if not in wordAndTypes then
        word ← word with min. levenshtein
    types ← types at index of word in wordAndTypes
    add Phrase(word = word, types = types, ...) to phrases
put phrases in phraseListsQueue
while not empty phraseListsQueue do
    phraseList ← pull first item of phraseListsQueue
    if length of phraseList equals 1 then
        add phraseList to finishedParsing
    for i in length(phraseList) do
        newPhrase ← forwardElimination(phraseList[i + 1], phraseList[i])
        if not empty newPhrase then
            newSentence ← delete phraseList[i + 1] from phraseListsQueue
            newPhrase ← forwardElimination(phraseList[i - 1], phraseList[i])
        if not empty newPhrase then
            phraseList[i] ← newPhrase
            newSentence ← delete phraseList[i - 1] from phraseListsQueue
        add newSentence to phraseListsQueue
```

4.2.2 Forward elimination

The function `forwardElimination` first calls `getForwardRequests` on the current `Phrase` object.

In `getForwardRequests`, for each possible type of a word, anything between brackets at the start of the type is deleted. We only need the types after the slash. If the type has a forward slash, anything at the start can be removed.

Next we look for the forward slash and look at the type after the slash. If this type is between brackets, brackets are deleted.

We then add these "requested types" (i.e. the types after the slash, potential brackets removed) to the `returnTypes` list. This list is returned, together with an array of the initial types.

Algorithm 2 Get forward requested types

```
procedure getForwardRequests(currentPhrase)
    returnTypes ← empty list
    types ← currentPhrase.types
    for i in length(types) do
        returnTypes[i] ← remove anything bracketed at start of types[i]
        if "/" in returnTypes[i] then
            returnTypes[i] ← everything after "/"
            returnTypes[i] ← remove brackets returnType[i]
    return returnTypes, types
```

Consequently, for all requested types, the `forwardElimination` algorithm checks whether it is in the types of the next `Phrase` object. If it is, the new type is defined, and the associated words are concatenated. The new types and the concatenated words are used to define a new `Phrase` object.

The same happens for a type with a backward slash, but then we save the type in *front* of the slash to the `returnTypes` list. Furthermore, we check whether any one of the requested types is in the types of the *previous* `Phrase` object.

Algorithm 3 Forward elimination

```
procedure forwardElimination(nextPhrase, currentPhrase)
    requestedTypes, originalTypes ← forwardRequests(currentPhrase)
    if not empty requestedTypes then
        newTypes ← empty list
        for i in length(requestedTypes) do
            if requestedTypes[i] in nextPhrase.types then
                newTypes[i] ← everything before "/" of originalType[i]
        if not newTypes empty then
            concat ← currentPhrase.word + nextPhrase.word
        return Phrase(word = concat, types = newTypes, leftPhrase = currentPhrase, rightPhrase = nextPhrase, ...)
```

4.2.3 Joint preference in subtree

Subsequently, we check for all the finished trees whether a user preference is present. If a particular user preference is present, it is checked whether it is contained in a `leftPhrase` or a `rightPhrase` in combination with "food", "restaurant", "priced", "part" or another keyword indicating the user preference.

The `leftPhrase` is set to the `currentPhrase`. The `rightPhrase` is the `nextPhrase`. This can be seen in Algorithm 3. After backward elimination, the `left` and `rightPhrase` attributes of the new `Phrase` object are assigned `previousPhrase` and `currentPhrase` respectively.

If a user preference is found in, for example, a `leftPhrase` we check whether it is directly, and only, connected to a preference class. It is important that is connected only to the preference class and not the preference class plus the rest of the sentence. If it is connected only to the preference class, the subtree of this user preference is disjoint.

Figure 3 shows a tree in which the different user preference classes are not disjoint. "cuban" is not directly connected to a leaf containing only "food". Rather, it is connected to leave containing "food" plus the rest of the sentence.

The algorithm will indicate that the "food" preference subtree is not disjoint.

4.3 Possible improvements

4.3.1 Misspelled words

First of all, it takes quite long to calculate the Levenshtein distance from a misspelled word to each word in the vocabulary. Perhaps we could tell the algorithm to stop calculating distance once a distance of 2 has been reached.

Furthermore, the first word with the shortest Levenshtein distance can easily be another word of another type. For example, let say we mistype "in" as "inm". The first word with the shortest Levenshtein distance in the vocabulary is "im". Both "in" and "im" have a Levenshtein distance of 1 from "imn". However, since "im" appears earlier in a alphabetically ordered vocabulary, this is the word we choose to replace "inm" with. This results in a wrong type, and a sentence that can not be parsed.

Of course, we could check every word with the smallest Levenshtein distance, to see if there is one that enables the sentence to parse completely. This would, however, be very time consuming. It may be more useful just to have the system ask "Can you please repeat that?" or state that it does not understand the utterance.

4.3.2 Problematic types

As for now, types with nested brackets do not parse correctly. For example "(np \(s\s))/np" is problematic. This due to the way brackets and bracketed types are handled by the algorithm.

We use the regular expression “^\([^)]*\)” in order to remove anything between brackets at the start of the type. This is removed, because it is not part of the requested type. The regular expression does not work on nested brackets, because the “[^)]*” part matches anything between the brackets that is not a closing bracket.

If a type has nested brackets, there is a closing bracket between the outer brackets. Everything at the start of the type is not removed and the resulting requested type is incorrect.

Luckily, the example sentences do not contain nested brackets. Nonetheless, there are types in the vocabulary that do.

5 Discussion Natural language generation

All steps described above are used to analyse the user input. Additionally, we implemented a natural language generation algorithm, in order to enable the system to respond. This algorithm identifies the speech act of the user, and gives an adequate response to this speech act. In addition, it keeps track of the preferences of the user and the suggested restaurants.

To achieve this, we made a template of a response to each speech act. Those templates can be seen in Table 5. For example, when a user asks to repeat what the system said, the system will repeat his last sentence. In this case, the system has a standard response to a specific speech act.

Sometimes, the user input must be further analysed in order to give an adequate response. The information of the user input must be implemented in the response of the system. For example, when the user wants to inform the system. In such situations, the content of the inform statement is determined by means of a dictionary of words for each of the possible inform categories (food, area, and price). It is then checked, for each of those words, if they occur in the sentence. If they do, the system will know this word represents a preference. It will save the preferences of the user in the order they are suggested and will incorporate this order when suggesting a restaurant.

Similarly, the system will identify utterances that explicitly state a lack of preference in a certain preference class. For example “Any area is fine”. In this case the system recognises the slot as filled, but will not take it into account when searching for a restaurant.

For suggesting a restaurant to the user, the systems makes use of a database that contains all possible restaurants. The system will compare the preferences of the user with the information about each restaurant. It makes a list of possible restaurants that suite the user’s preferences. When this list is small enough, the system will suggest one of those restaurants to the user. When, based on the user preferences, there are still more than 10 possible restaurants, and the system does not yet know all preferences of the user, it will ask for more information. When there are no restaurants at all that meet the preferences, the user is asked to adjust their expectations.

Examples of dialogues with the system can be seen in Table 3 and 4. As can

System	Welcome to the restaurant system you can ask for restaurants by price, area and the type of food. What would you like?
User	*I would like a restaurant in the expensive price range serving American food*
System	Sorry i could not find a restaurant with your preferences, do you have something else you would like?
User	*African food*
System	Bedouin is a restaurant in the expensive price range serving african food.
User	*Anything else?*
System	Sorry there are no other restaurants with your preferences. I will repeat the earlier suggestion. bedouin is a restaurant in the expensive price range serving african food.
User	*What is the phone number?*
System	The phone number is 01223 367660
User	*thank you*
System	You're welcome. Goodbye!

Table 3: Example dialogue with the system.

be seen in the first dialogue the system keeps track of the last suggested restaurant. In addition it is able to change the preferences when no corresponding restaurant is found.

6 General Discussion

6.1 Design and performance

The way in which the number of output nodes is optimised lacks elegance and intelligence. It reflects a very bare understanding of how the network actually works. It is a slow way of checking all possible options. A better way to optimise the hyperparameters settings may be to use reinforcement learning. The reward should be dependent on the accuracy that certain parameter settings yield. Reinforcement learning could be used to predict which parameter settings are most likely to yield the best accuracy, even without checking all possible options.

But even though this could be a quicker solution than testing all possible options, it still does not yield much insight into the influence of each parameter on the accuracy. The best solution may be, to make adequate choices for the parameter settings, based on insight in their influence on the network.

Furthermore, we did not look at the influence of different cut-off lengths for the user utterances in the training set at all. It may be that this would have improved the performance of the system.

The deduction algorithm checks all possible combinations, for all different types, until a tree has been found that parses correctly. This makes it, in some

System	Welcome to the restaurant system you can ask for restaurants by price, area and the type of food. What would you like?
User	*Hi*
System	Hello, what type of food, price range and area are you looking for?
User	*I would like a restaurant in the expensive price range*
System	What kind of area would you like?
User	*any area would be fine*
System	What kind of food would you like?
User	*european*
System	cambridge lodge restaurant is a restaurant in the expensive price range in the west part of town serving european food.
User	*anything else?*
System	eraina is a restaurant in the expensive price range in the centre part of town serving european food.
User	*what is the address?*
System	The address is free school lane city centre c.b 2
User	*thank you*
System	You're welcome. Goodbye!

Table 4: Example dialogue with the system.

cases, a very time consuming algorithm. Many futile options are processed. If there would be a clever way to quickly check whether an option is likely to be futile in advance, this could save a lot of processing time.

Finally, in the natural language generation part, the dialogue is considered finished when a user utterance is classified as "thankyou". This introduces an issue. It may be very natural for a user to start another speech act with a word of thanks. Because any utterance starting with a word of thanks is classified as a "thankyou" speech act, the system will fail to extract any relevant info and closes the dialogue. This could be improved by using other termination conditions then a user uttering a "thankyou" or a "bye".

6.2 Collaboration

Their was some variety in skill sets of each individual group member. Nonetheless, each member contributed at the top of their capabilities, in an attempt to make a successful restaurant recommendation system.

All group members contributed actively to discussions during the group meetings. When it comes to participation, useful insights, and critical notes, every group member contributed equally and wholeheartedly.

The general planning can be seen in the Log (Table 4). The dialogue diagram of part 1 was completed in close collaboration with all the members. The text file with the dialogues was created by each member individually, in order to make sure that everybody got more familiar with python.

The machine learning model was actively discussed together during group

speechact	**reply**
ack	Is there anything else you want to know?
affirm	*Suggest a restaurant*
bye	*Exit the dialogue*
confirm	Yes, indeed. No, *Give correct information*
deny	What are you exactly looking for?
hello	Hello, what type of food, price range and area are you looking for?
inform	*Suggest a restaurant* What kind of ? would you like? Sorry I could not find a restaurant with your preferences, do you have something else you would like?
negate	Is there anything else I can help you with?
null	Sorry, I don't understand what you mean
repeat	*Repeat last sentence*
reqalts	*Suggest other restaurant* Sorry there are no other restaurants with your preferences. I will repeat the earlier suggestions.
reqmore	? is a ? priced restaurant serving ? food in the ? part of town. Its phone number is ? Its address is ?
request	The pricerange of the restaurant is ? The restaurant is in the ? of town. The restaurant serves ? food. The phone number is ? . The address is ? . The ? is unknown.
restart	Welcome to the restaurant system. You can ask for restaurants by price, area and the type of food. What would you like?
thankyou	You're welcome. Goodbye! *exit dialogue*

Table 5: System replies

11–09	First meeting, finished txt. File with all dialogues
13–09	Second meeting, analysed dialogues and created the model together
18–09	Finished part 1, script for data transformation and started on machine learning
25–09	Optimised the machine learning part and started on report
02–10	Finished part 2 and started on part 3
09–10	Finished assigning types to words, started on code for part 3
16–10	Continued working on part 3, started on part 4
23–10	Finished part 3, continued part 4 and report
30–10	Working on part 4 and report
06–11	Working on report

Table 6: Log

meetings. Useful contributions and ideas have been proposed by all group members. Kevin made the script to transform the data and later defined the model. Meanwhile, Tycho and Saskia started on the report. Fleur looked at what would be the best option for the output nodes.

Next up was part 3. All group members contributed equally to assigning types to all the words in the "inform" vocabulary. When all types where assigned we reviewed each others work.

Kevin and Fleur later started on the sentence parsing. This part proved to be challenging. Both spend a lot of time on trying to make their individual parsing algorithm work, both of which worked on the example sentences in the end. Fleur's algorithm was not readable and unnecessary complicated, so we choose to use the more insightful code of Kevin. Kevin extended the parsing algorithm with information extraction and a check of whether a tree is joint or disjoint.

Meanwhile, Tycho and Saskia already started on part 4, the natural language generation. They discussed the architecture of the natural language generation system together. Thereafter, Tycho made the templates, while Saskia implemented the whole pipeline of the system. Those two parts were combined together and Kevin restructured the code, resulting in the final dialogue system.

References

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org/, 2015. Accessed November 6th, 2018.

[2] François Chollet et al. Keras. https://keras.io, 2015. Accessed November 6th, 2018.

[3] Matthew Henderson, Blaise Thomson, and Jason Williams. Dialog state tracking challenge 2 & 3, 2013.

[4] Daniel Lindsley et al. pylev. https://github.com/toastdriven/pylev, 2015. Accessed November 8th, 2018.

[5] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, October 1965.

[6] Marijn Schraagen. Machine learning. https://uu.blackboard.com, September 2018. Lecture slides. Accessed November 6th, 2018.

[7] Avinash Sharma. Understanding Activation Functions in Neural Networks. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0, 2017. Accessed November 6th, 2018.

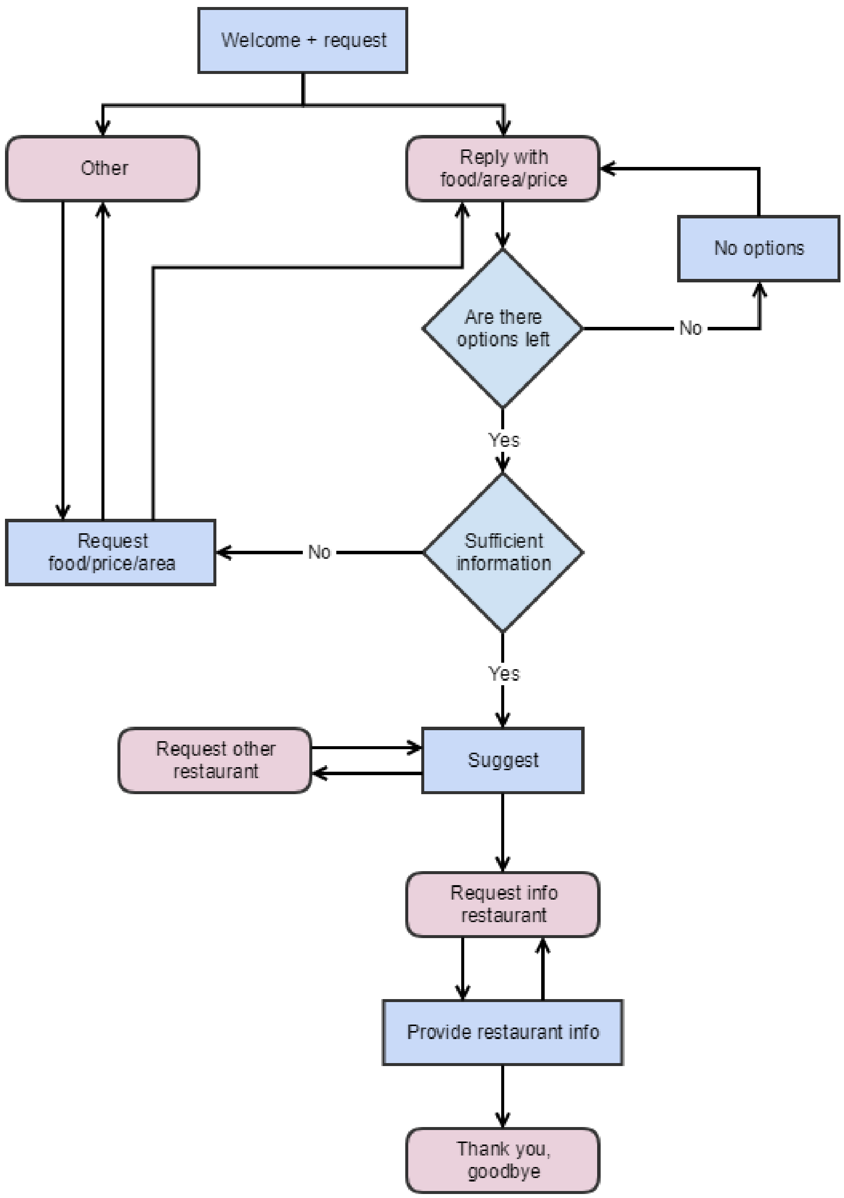

Figure 1: State transition model

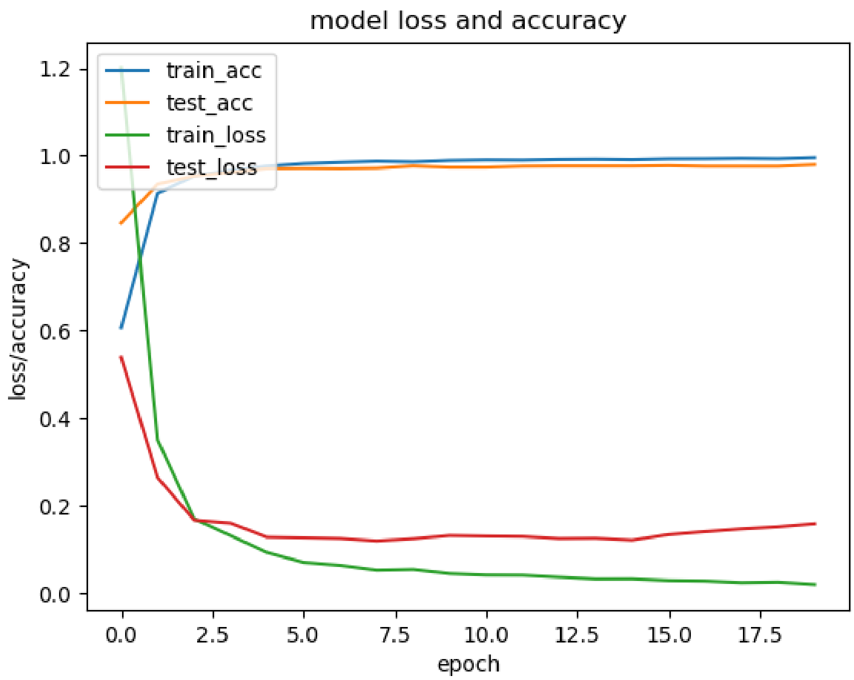

Figure 2: State transition model

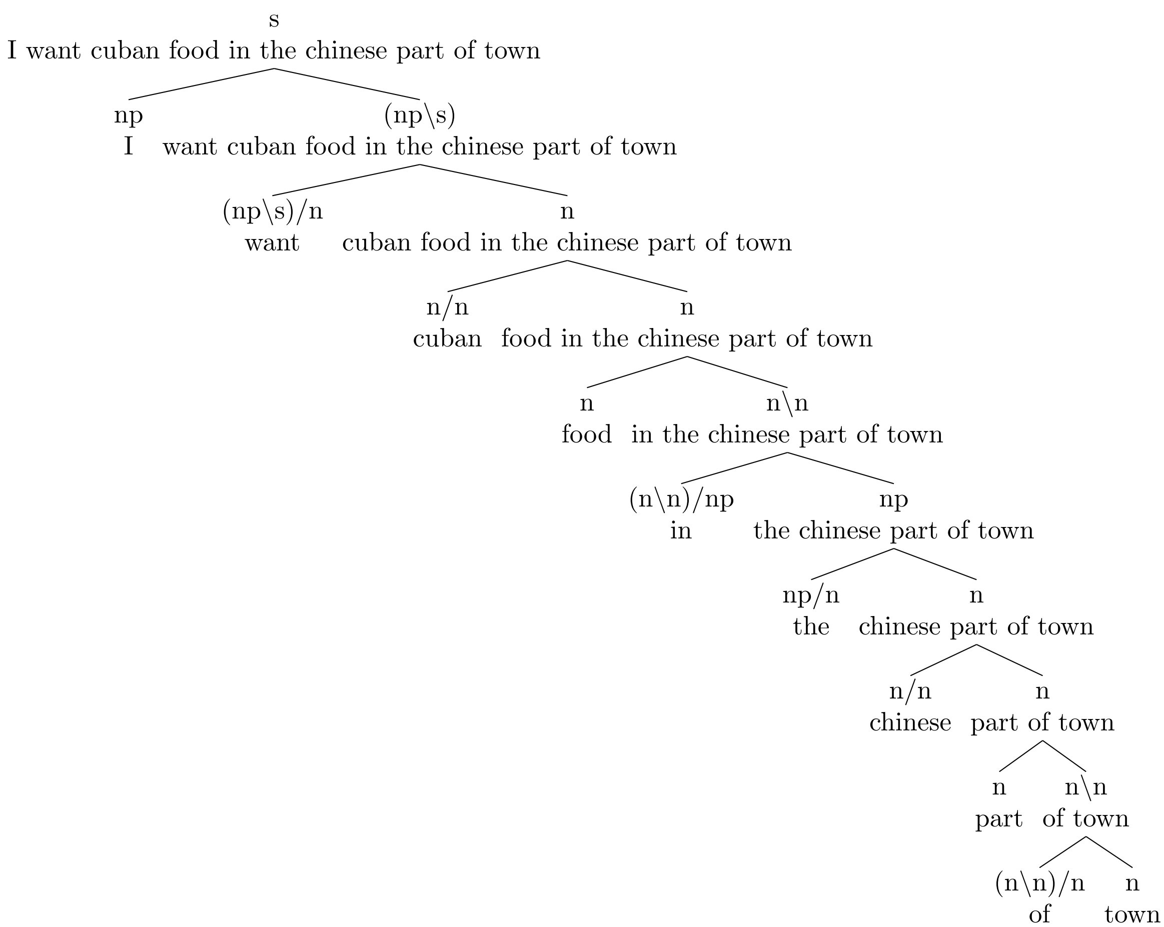

Figure 3: An example of a tree in which preference classes are joint.

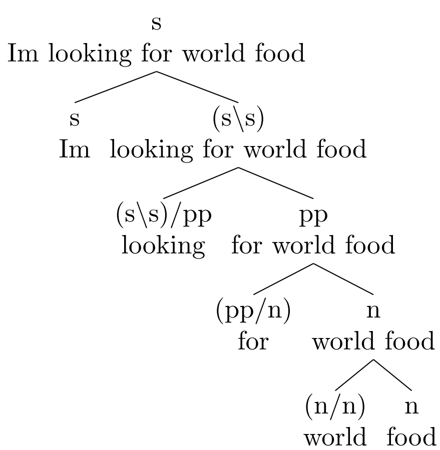

Figure 4: An example of an CCG parse tree.