# Markov Chains in Chat Bots

Kyle Tuckey
5/19/2017

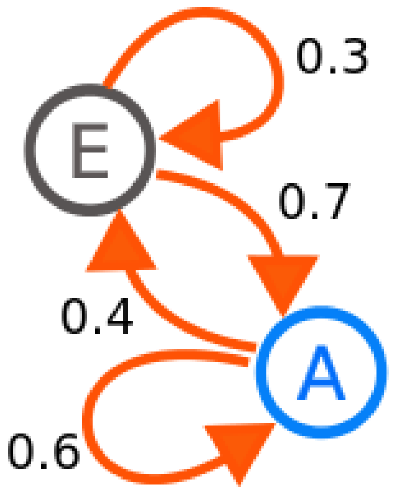
## Contents

## 1. Introduction

Markov chains are used in many different applications in life, from creating simple weather maps, to the autosuggestion system in smartphones. Markov chains often see use in chat bots as a way of generating sentences to show to the user. In this report we explore how a Markov chain can be used for sentence generation and also what properties make a good textual source for Markov chain generation.

## 2. What is a Markov Chain?

A Markov chain is a mathematical state system that hops from one state to another. It also tells you the probability of hopping from one state to another (Victor Powell, 2014).

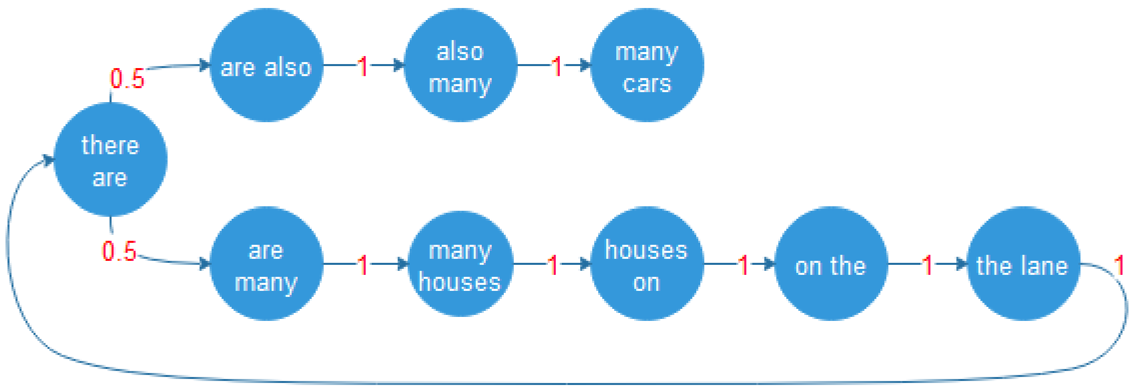


*1 https://en.wikipedia.org/wiki/Markov_chain#/media/File:Markovkate_01.svg*

The above image shows how a Markov chain can be modelled. A Markov chain can be used to model the behaviour of many different entities. If we take the above image for example, if we were modelling the behaviour of a Dog, state E might represent when the dog barks and state A might represent when the dog whimpers. So, there is a 70% chance that the dogs bark will be followed by a whimper.

## 3. How can it be used for sentence generation?

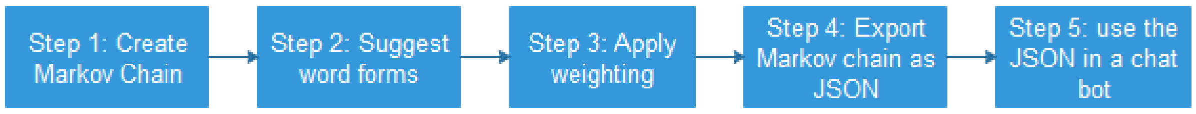
Markov chains can also be used to represent sentence structure. If we take the sentence "There are many houses on the lane, there are also many cars" we can convert this sentence into a Markov chain.



This means that we can implement an algorithm that can convert any text into a Markov chain, and then use that chain to generate sentences.

## 4. Program flow

The flow of the program can be broken down into 5 major steps, the diagram below shows these steps.

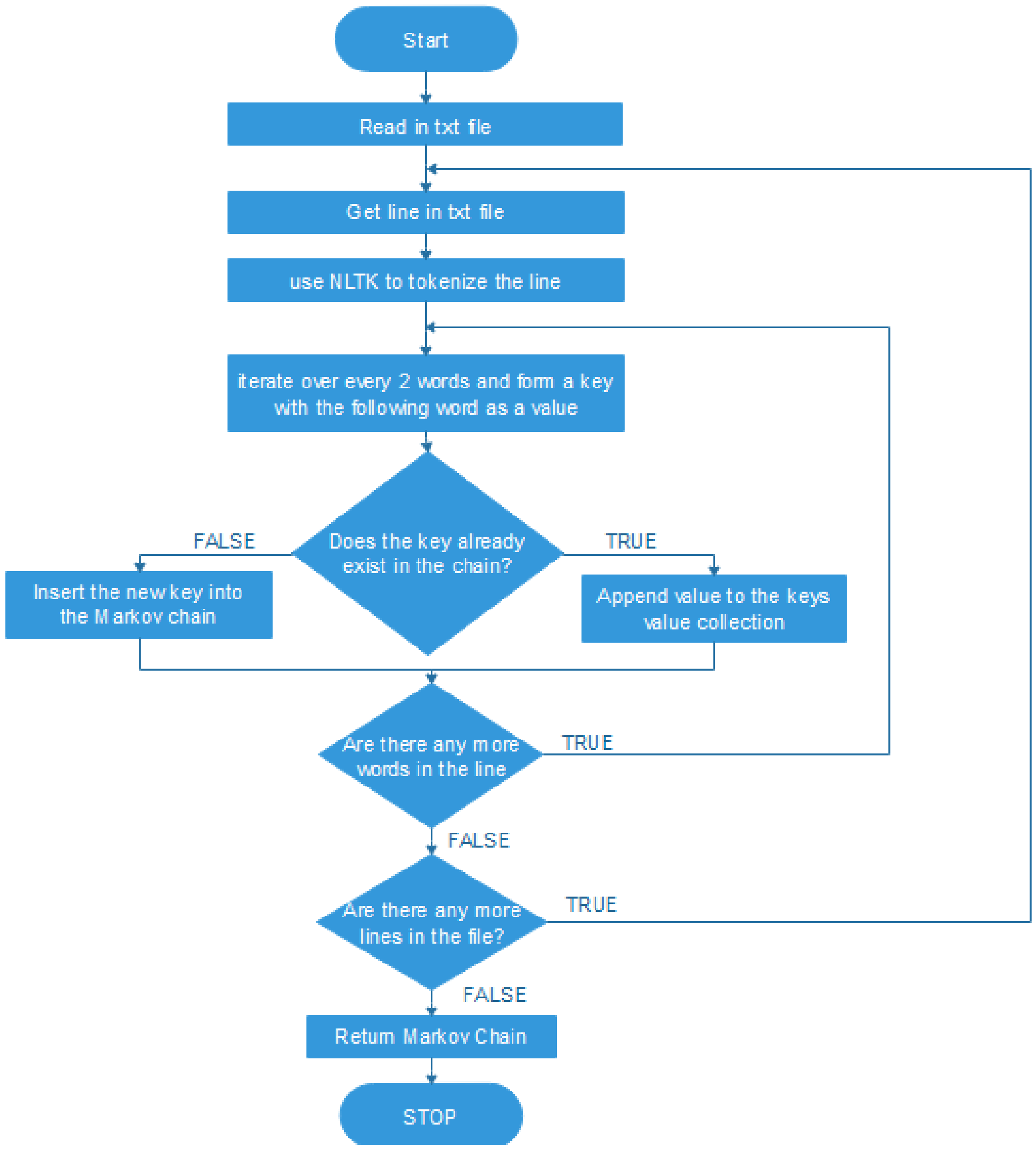


Step 1: Create Markov Chain
Step 2: Suggest word forms
Step 3: Apply weighting
Step 4: Export Markov chain as JSON
Step 5: use the JSON in a chat bot

We will go over the flow of each step and explain what process happens across each step and what the end result is.

## 4.1 Step 1: Create Markov Chain

The first thing that needs to happen is the creation of the Markov chain, and to start creating a Markov Chain we need to read in a .txt file which contains some text.


Start
Read in txt file
Get line in txt file
use NLTK to tokenize the line
iterate over every 2 words and form a key with the following word as a value
Does the key already exist in the chain?
FALSE
Insert the new key into the Markov chain
TRUE
Append value to the keys value collection
Are there any more words in the line
TRUE
FALSE
Are there any more lines in the file?
TRUE
FALSE
Return Markov Chain
STOP

After we read in the .txt file we want to iterate over every line in the file. As we iterate over every line, we tokenize the line and tag each word with their word class so for example, the word 'man' would be tagged as a NOUN we do this to help us in step 2 later. Once we have a line we want to iterate over every 2 words in the line and create a key with those words, the word that is immediately after the second word will be the keys value. Once we have iterated over every word in the line we iterate to the next line, and repeat this process. Once we have reached the end of the file, we return the Markov Chain. If the read in text file had the sentence "Hey there man, just writing to say happy birthday! "the result would look like this:

```
“Hey there : NOUN DET” : {“man : NOUN”},
“there man : DET NOUN” : {“just : ADV”},
“man just : NOUN ADV” : {“writing : VERB”},
“just writing : ADV VERB” : {“to : PRT”},
“writing to : VERB PRT” : {“say : VERB”},
“to say : PRT VERB” : {“happy” : ADJ}
…
```

## 4.2 Step 2: Suggest word forms

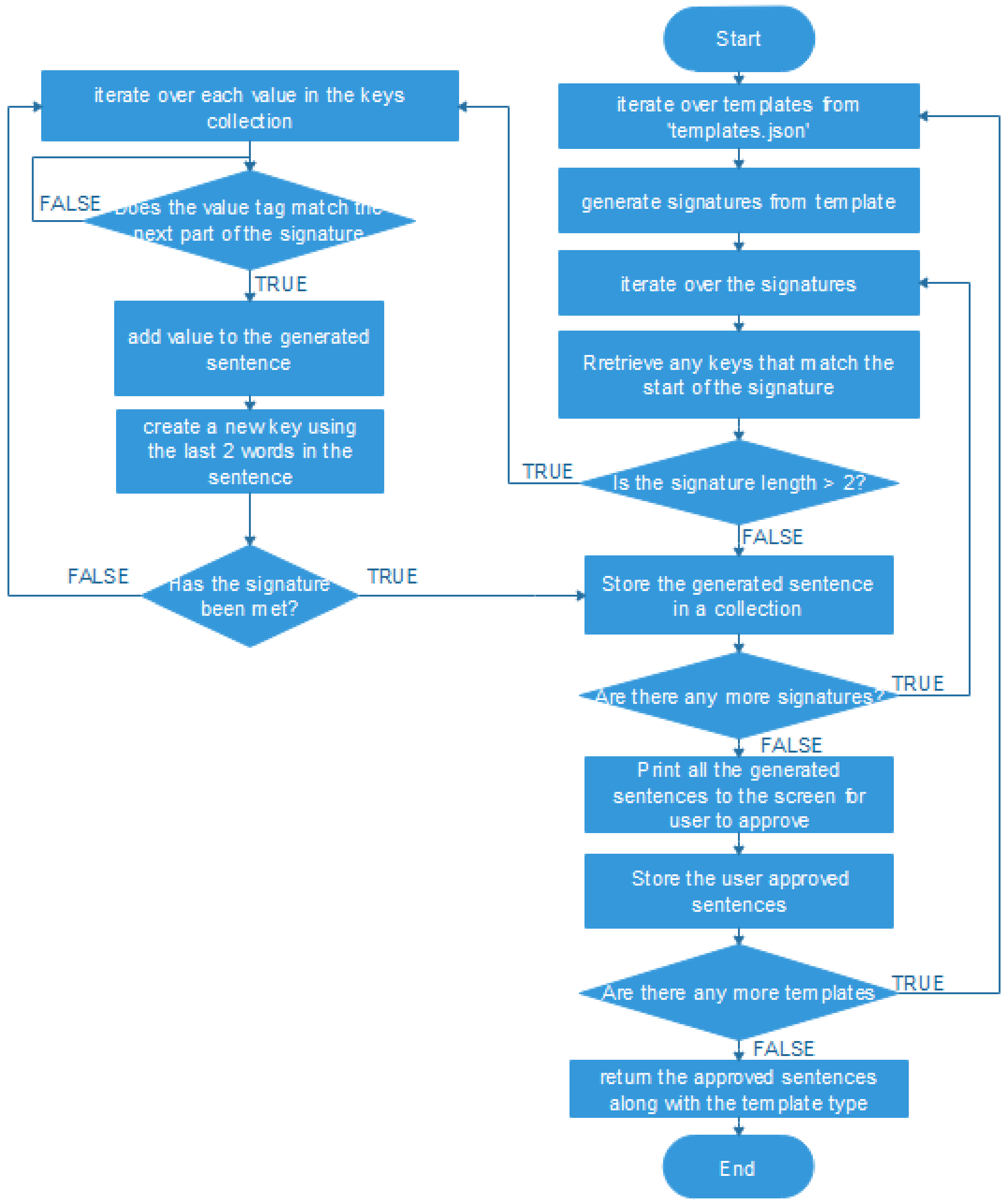
Now that we have the Markov chain generated, we can begin generating sentences. However any sentences we generate are very likely to generate nonsense, this is because we have yet to apply probability into the Markov chain itself.

In order to generate sentences that actually make sense we need a way of telling the program what sentences make sense. To tell the program what sentences to create we need to use the NLTK library to generate a signature for the Markov chain to use when generating sentences (Steve Bird et al, 2015). For example let’s say we want the bot to be able to generate an appropriate greeting to the user, we will want the Markov chain to generate a greeting that might be along the lines of “hello there”. We can’t suggest the words ‘hello’ or ‘there’ because it is possible that the Markov chain does not have either word, instead what we can do is take the word classes of both words which would be “NOUN DET” and suggest that the program instead build a sentence to match that signature. So instead of “hello there” the program might build a string called “hey there” which matches the “NOUN DET” signature and is a greeting.

We prepared for this in step 1 of the program by tagging each word as we were reading in the txt file. We are going to store the templates for suggested sentences along with the type of template it is, in a JSON file called ‘template.json’. Doing this allows us to easily extend the templates without having to edit the source code. A template in the JSON might look like this

```
{
    "Type":"Age",
    "Responses":["22 years old", "I am 20", "20 years"]
},
```

What the above object does is propose potential responses to the Markov chain when responding to an input that is interrogating it for its age. The flow chart for this logic can be seen below.

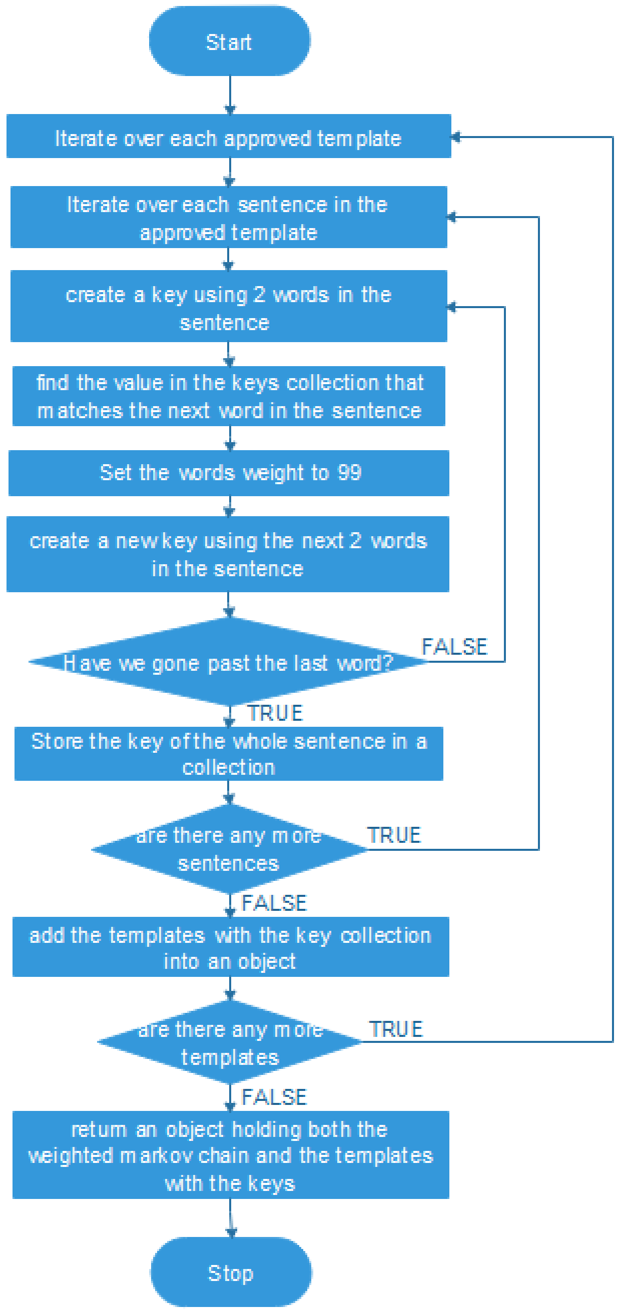

Start
iterate over templates from 'templates.json'
generate signatures from template
iterate over the signatures
Rretrieve any keys that match the start of the signature
Is the signature length > 2?
TRUE
FALSE
Store the generated sentence in a collection
Are there any more signatures?
TRUE
FALSE
Print all the generated sentences to the screen for user to approve
Store the user approved sentences
Are there any more templates
TRUE
FALSE
return the approved sentences along with the template type
End
iterate over each value in the keys collection
Does the value tag match the next part of the signature
FALSE
TRUE
add value to the generated sentence
create a new key using the last 2 words in the sentence
Has the signature been met?
FALSE
TRUE

We need to iterate over each template that is stored in the 'templates.json' file, we then generate a signature from each response in the template. We need to find any keys that match up with the first 2 parts of the signature so if we have a signature that is "NOUN DET NOUN" then we want a key that has the tags "NOUN DET". If the signature is not greater than 2 words, then we don't need to generate any more of the sentence and can just use the key as a generated sentence. If the signature is greater than 2 words, then we need to search for a value that matches the next part of the signature, so we parse the key and signature into a recursive function that iterates through the Markov chain and appends any matching values to the sentence, once the signature has been met this process ends and we store the sentence. Once we have been through every signature and generated every sentence the Markov can provide, we print them to the screen along with their index in the list. The user then types the index of any sentence that is appropriate for the type of response it is trying to match. Once every template has been iterated over and all the sentences have been approved we return all the approved sentences including the templates they were approved for.

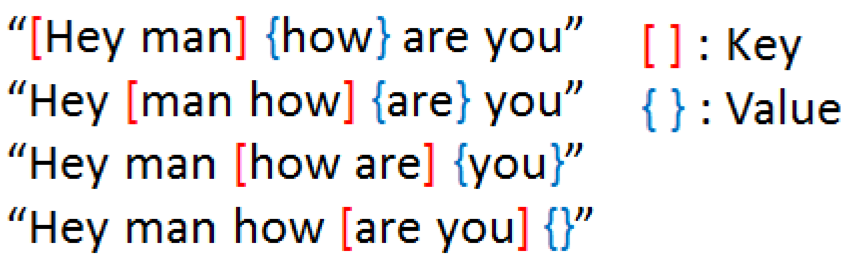
## 4.3 Step 3: Apply weight

After we have our approved sentences we need to iterate over each approved sentence and apply a weight to each word in the chain that was approved, these weights act as the probability that a particular value will be picked over another.

The below diagram shows the flow of this particular part of the program:


Start
Iterate over each approved template
Iterate over each sentence in the approved template
create a key using 2 words in the sentence
find the value in the keys collection that matches the next word in the sentence
Set the words weight to 99
create a new key using the next 2 words in the sentence
Have we gone past the last word?
FALSE
TRUE
Store the key of the whole sentence in a collection
are there any more sentences
TRUE
FALSE
add the templates with the key collection into an object
are there any more templates
TRUE
FALSE
return an object holding both the weighted markov chain and the templates with the keys
Stop

We go over every sentence in each approved template, for each sentence we incrementally go over every 2 words in the sentence, which is used to represent a key in the Markov chain. We then search the keys values collection for the word immediately after the key in the sentence.


“[Hey man] {how} are you”
“Hey [man how] {are} you”
“Hey man [how are] {you}”
“Hey man how [are you] {}”

[ ] : Key
{ } : Value

The image shows how the iteration works; we get 2 words in the sentence, which is the key we need. The key should have a value in its collection that matches the value in the sentence. We want to find the value in the keys collection and set its weight to 99. Once the value exceeds the length of the sentence, we will have applied the necessary weight to every word in the sentence. This process repeats for every sentence that was approved by the user in step 2, the result is a Markov chain that has been properly weighted, and a collection of templates and the keys that will begin sentence construction in the chat bot later.

## 4.4 Step 4: Export Markov chain as JSON

Step 4 is very straight forward, at this point in the program we will have all the information that we need to start creating responses in our chat bot, so all we need to do now is represent that data in a format that can be dumped into a JSON file. Python requires data to be in a very specific data representation in order for it to dump it as JSON, all this step does is make sure that the data is in that specific representation. Once it has formatted the data properly it writes it to a file called ‘brain.json’ the data that gets written to the ‘brain.json’ is the Markov chain and the templates the chat bot will need to use.

## 4.5 Step 5: use the JSON in a chat bot

The chat bot reads in ‘brain.json’ and is then able to access both the weighted Markov chain and the templates it will use, along with the keys its needs to use when creating sentences.

The chat bot waits for the user to input some text, once the user has it will check the text for some keywords. For example, the chat bot might look for the word ‘hello’ to determine if it is being greeted by the user. If it determines that the user has input a greeting it will return one of the keys stored in the ‘Greeting’ template and parse it into the Markov chain.

The Markov chain will return its collection of values, it then selects a value randomly, but this choice is influenced by the weight of the word, so the chat bot is much more likely to pick the word with a weight of 99 than one that has a weight of 1. Once it has generated a sentence with all the values it can find, it outputs it to the user and then awaits user input again. This process does not end until the user inputs an empty string.

## 5. Results

A range of text sources were used in the program to see which texts provided results for a set of 7 templates:

- Hello there (greeting)
- Are you real? (reality)
- What is your name? (Name)
- How old are you (Age)
- How are you? (State)
- What are your hobbies (Hobby)
- What do you look like? (Appearance)

Most of these were taken from a blog regarding the most common chat bot questions on the MSDN website (Benjamin Perkins, 2016). The goal of this testing was to see which forms of textual media best answered these questions if at all, and what were the properties within these texts that best helped in answering these templates. In the interest of keeping the results as fair as possible, the same expected responses and templates were used for each text source.

| File name | Source type | Size | Greeting | Reality | Name | Age | State | Hobby | Appearance |
|---|---|---|---|---|---|---|---|---|---|
| **Bruce Almighty** | Movie Script | 6340 lines | 1 | 0 | 12 | 12 | 1 | 1 | 0 |
| **Beyond Good and Evil** | EBook | 6507 lines | 0 | 1 | 6 | 3 | 1 | 0 | 0 |
| **Ace Ventura Pet Detective** | Movie Script | 6374 lines | 3 | 0 | 10 | 4 | 2 | 0 | 0 |
| **A Sailor in Spite of Himself** | EBook | 9690 lines | 1 | 0 | 52 | 18 | 0 | 0 | 0 |
| **Aliens** | Movie Script | 6796 lines | 0 | 0 | 4 | 5 | 0 | 2 | 0 |
| **Pride and Prejudice** | EBook | 13427 lines | 0 | 0 | 26 | 22 | 0 | 0 | 0 |

The values in each column show how many believable responses there were for each template. The number of tested sources is not big enough to give concrete or accurate data on what textual media provides better results. But within the context of this assignment, there was greater success with the movie scripts in providing believable and accurate responses. This may be due to the first-person point of view that the movie scripts had, as well as their focus on dialogue. So, the properties that best helped with finding appropriate responses were:

- First-person point of view
- Large line count

- Heavy use of dialog

Bearing this in mind, large interpersonal chatlogs might act as a good source for a Markov Chain chat bot.

## 6. Problems with Markov Chains in chat bots

An aspect of chat bots that give the impression of a smart AI is its ability to answer open ended questions such as “What is the time?” or “What is the capital of England”. A Markov chain can only answer a question, if the answer itself is within its vocabulary. If the words don’t exist in a Markov chain then it can’t use them, so if a Markov chain does not have the word “London” in it, then it can’t tell you what the capital of England is. As for time, because time changes constantly, there is no way a Markov chain can accurately tell you the current time.

Another problem is that getting a Markov chain to even respond with a sentence that makes relative sense, takes a lot of work. There are simpler implementation’s that achieve the same and some times better results than a Markov Chain can. Having a simple templating system with pre-defined responses and a case based reasoning system is a lot simpler to implement and is more guaranteed to produce believable results.

Because of the way Markov chains store information, they can take up quite a bit of memory space. Reading in the script of Bruce Almighty, converting it into a Markov chain and then exporting it in JSON format creates a file that is 1.10MB in size, which is impressive considering the original Bruce Almighty script, is only 133KB in size.

## 7. Potential Improvements

Currently the system is only capable of reading in 1 source and creating a Markov chain using that one source. It would be pretty interesting if the program was able to read in multiple sources into the ‘brain.json’ and add to the existing Markov chain. This would expand its vocabulary and allow it to answer more complex questions.

The current templating system in place for detecting the meaning of each input is very basic and could easily be expanded on. Currently all the program does is check the user’s inputs for keywords relating to the sentiment, and then chooses the appropriate response. It might be better to do something with NLP here and check the input against a corpus of words rather than a few keywords, this might provide a more dynamic and interesting way of detecting meaning.

Ideally the keyword identifying part of the application should be moved to the JSON file, this way keywords can be specified without having to edit the source code.

## 6. Literature Review

### Markov Chains Explained Visually

url: http://setosa.io/ev/markov-chains/
archived url: https://perma.cc/ERK9-WZH7

This source is great for understanding what Markov chains do and how they work.  It explains the idea behind Markov chains in a visual way that makes it very easy to grasp and understand. It allows the user to manipulate Markov chains directly to get a better understanding on how adding states and changing probability's affect the Markov Chains themselves. If you are looking to explain Markov chains in a lecture or tutorial then this is a great resource to use.

### Natural Language Processing with Python

url: http://www.nltk.org/book/
archived chapter 5 url: https://perma.cc/KXF2-8CZF
archived book url: https://perma.cc/8HX8-4BAZ

This is an open source book which details how to use the NLTK toolkit in python 3. This was a huge help in understanding how to use NLTK to tag sentences and words for use in this program. The section of the book which was especially useful was chapter 5 of the book which focuses on categorising and tagging words. The specific section of this chapter that was a great help was section 2.3 of chapter 5, which shows a universal part-of-speech tag set which is used in this project. I encourage anyone looking to learn more about NLP and NLTK to read through this and try out some of the various topics in the book, it might even be worth including this as part of the reading list on the module.

## 7. Conclusion

Markov chains are very useful in many different aspects of the industry, and as seen in this assignment, it can be used to generate sentences and can be weighted to provide specific responses. However, I think that a chat bot would benefit from alternative implementations if one is seeking to create a smarter AI that can answer more open ended and intriguing questions. Another alternative is to have a separate system work alongside the Markov Chain implementation to make up for its short comings. NLP provides a lot of potential improvements to the system and may be worth investigating further.

## 8. References

Victor Powell. (2014). *Markov Chains Explained Visually.* Available: http://setosa.io/ev/markov-chains/. Last accessed 18/05/2017.

see literature review.

Steven Bird, Ewan Klein, Edward Loper. (2015). *Natural Language Processing with Python.* Available: http://www.nltk.org/book/. Last accessed 18/05/2017.

see literature review.

Micheal Hart. (n.d.). *Project Gutenberg.* Available: https://www.gutenberg.org/wiki/Main_Page. Last accessed 18/05/2017.

resource for the open source eBooks in this assignment.

IMSDb. (). *The Webs Largest Movie Script Resource.* Available: http://www.imsdb.com/. Last accessed 18/05/2017.

Resource for the movie scripts used in this assignment.

Benjamin Perkins. (2016). *Most common chatbot questions and how to answer them.* Available: https://blogs.msdn.microsoft.com/benjaminperkins/2016/11/21/most-common-chatbot-questions-and-how-to-answer-them/. Last accessed 18/05/2017.