The curious case of Mark V. Shaney

Comp 140
Fall 2008

Who is Mark V. Shaney?

- Mark was a member of a UseNet News group called net.singles, a users group chock full of dating tips, lonely heart chatter, frank discussions of sexual problems and high tech missionary gospel about the sins of premarital smut-typing.

Penn Jilette's description

Who is MVS?

- Mr Shaney was a little goofy but he was always there. He chimed in with poetic opinions on romantic empathy:
  - "As I've commented before, really relating to someone involves standing next to impossible."

MVS contd…

- And he had a great Groucho Marx sense of humor:
  - "One morning I shot an elephant in my arms and kissed him. So it was too small for a pill? Well, it was too small for a while.“

MVS

- And his idea of a good closing was:
  - "Oh, sorry. Nevermind. I am afraid of it becoming another island in a nice suit."

MVS on Bush’s speech

- Mr. Chairman, delegates, fellow citizens, I'm honored to aid the rise of democracy in Germany and Japan, Nicaragua and Central Europe and the freedom of knowing you can take them. Tonight, I remind every parent and every school must teach, so we do to improve health care and a more hopeful America. I am in their days of worry. We see that character in our future. We will build a safer world today. The progress we and our friends and allies seek in the life of our work. The terrorists are fighting freedom with all their cunning and cruelty because freedom is not America's gift to every man and woman in this place, that dream is renewed. Now we go forward, grateful for our older workers. With the huge baby boom generation approaching retirement, many of our work. About 40 nations stand beside us in the next four years.

Compiled musings of MVS

- http://www.harlanlandes.com/shaney/1984_09.html

Mark V. Shaney

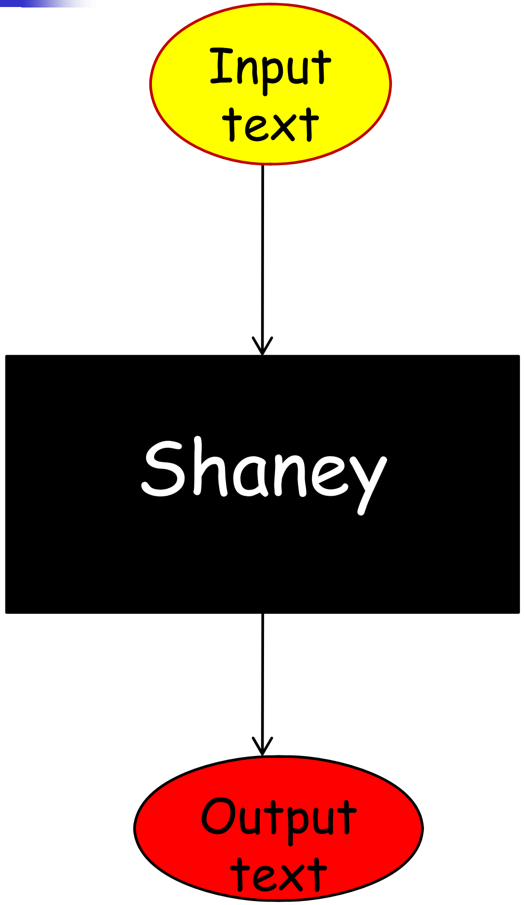
- A program created at AT&T Research Labs by
  - Bruce Ellis
  - Rob Pike
  - Don Mitchell
- Name is a play on “Markov chain“, which is the underlying technology.

Motivating the reconstruction of Shaney

- Wouldn't you like to write a program that could read a thousand words of something and spew out lovable nonsense in the same style? Your own little desktop Bret Easton Ellis, that sucks up the culture of your choice and spits it back at you? Don't let the Murray Hill address scare you, now that rob and brucee have done the hard work of thinking it up, even you and I can understand how Mark V. Shaney works and with a little work you and I can write our own (but let's hope to hell we all have something better to do with our lives - what is on the Weather Channel tonight?)

--- Penn Jillette

What does Shaney do?



We know Shaney riffs on texts that he reads. We can therefore guess his inputs and outputs.

We also know that
Shaney generates output text that is similar to the input text.
(of the same genre, on the same topic, with similar words)

Outline of lecture

- Reverse engineering Shaney
  - 10 minute group exercise
- Shaney's recipe
- Mathematical model
- Computational realization of model
- Fun with our model

Allen B. Downey

- The goal is to teach you to think like a computer scientist. This way of thinking combines some of the best features of mathematics, engineering, and natural science. Like mathematicians, computer scientists use formal languages to denote ideas (specifically computations). Like engineers, they design things, assembling components into systems and evaluating tradeoffs among alternatives. Like scientists, they observe the behavior of complex systems, form hypotheses, and test predictions. The single most important skill for a computer scientist is problem solving. Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately.

-- How to think like a computer scientist

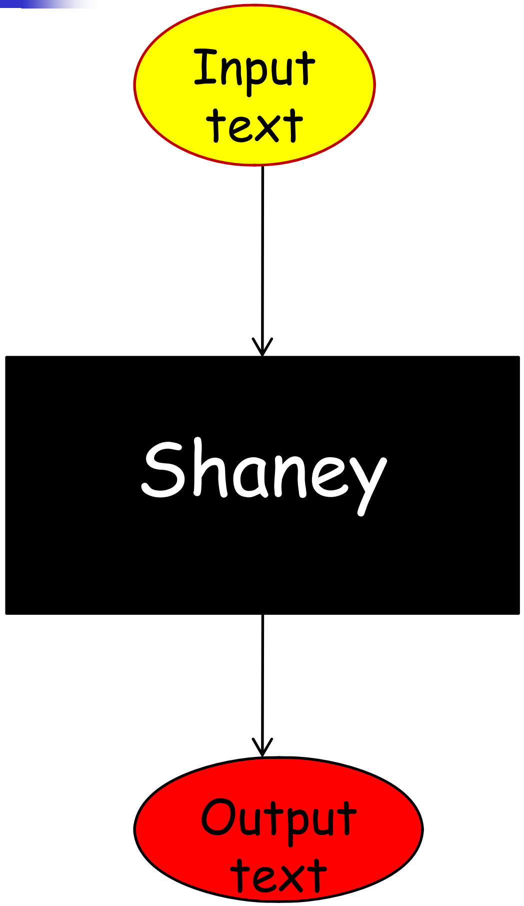
Questions

- Abstraction
  - Was the problem specified precisely? What are the inputs and outputs?
  - How did you represent the inputs and outputs?
- Automation
  - How did you express your recipe for a solution?
  - How can you demonstrate that your recipe solves the problem?
  - How expensive is it to run/use your recipe (where cost is defined in units related to the size of the input)?
  - Are there other recipes to solve the problem? Is your recipe the best there could ever be?

Abstraction

- Inputs: sequence of words
- Outputs: sequence of words “similar” to inputs
  - Use same or similar vocabulary (be about the same topic(s))
  - Use same or similar “phrases” (short sequences) (have similar linguistic style)

Automation



Reads posts on net.singles or some other source.

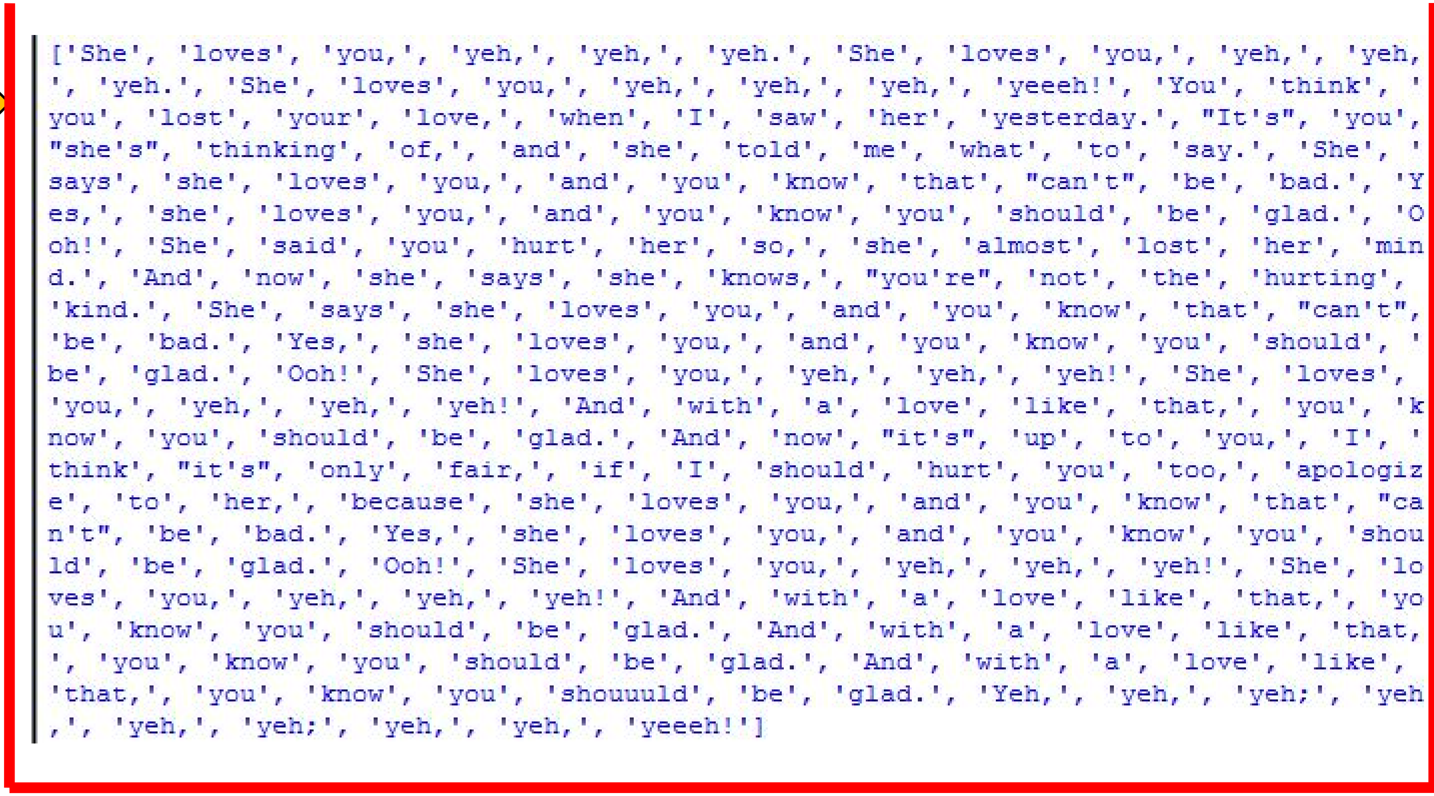
Creates a GENERATIVE mathematical model of these posts

Computationally constructs new posts based on this model

The lyrics of "She loves you"

She loves you, yeh, yeh, yeh. She loves you, yeh, yeh, yeh.
She loves you, yeh, yeh, yeh, yeeeh!
You think you lost your love, when I saw her yesterday.
It's you she's thinking of, and she told me what to say.
She says she loves you, and you know that can't be bad.
Yes, she loves you, and you know you should be glad. Ooh!
She said you hurt her so, she almost lost her mind.
And now she says she knows, you're not the hurting kind.
She says she loves you, and you know that can't be bad.
Yes, she loves you, and you know you should be glad. Ooh!
She loves you, yeh, yeh, yeh! She loves you, yeh, yeh, yeh!
And with a love like that, you know you should be glad.
And now it's up to you, I think it's only fair,
if I should hurt you too, apologize to her,
because she loves you, and you know that can't be bad.
Yes, she loves you, and you know you should be glad. Ooh!
She loves you, yeh, yeh, yeh! She loves you, yeh, yeh, yeh!
And with a love like that, you know you should be glad.
And with a love like that, you know you should be glad.
And with a love like that, you know you shouuuld be glad.
Yeh, yeh, yeh; yeh, yeh, yeh; yeh, yeh, yeeeh!

The simplest model



```
['She', 'loves', 'you,', 'yeh,', 'yeh,', 'yeh.', 'She', 'loves', 'you,', 'yeh,', 'yeh,
', 'yeh.', 'She', 'loves', 'you,', 'yeh,', 'yeh,', 'yeh,', 'yeeeh!', 'You', 'think', '
you', 'lost', 'your', 'love,', 'when', 'I', 'saw', 'her', 'yesterday.', "It's", 'you',
"she's", 'thinking', 'of,', 'and', 'she', 'told', 'me', 'what', 'to', 'say.', 'She', '
says', 'she', 'loves', 'you,', 'and', 'you', 'know', 'that', "can't", 'be', 'bad.', 'Y
es,', 'she', 'loves', 'you,', 'and', 'you', 'know', 'you', 'should', 'be', 'glad.', 'O
oh!', 'She', 'said', 'you', 'hurt', 'her', 'so,', 'she', 'almost', 'lost', 'her', 'min
d.', 'And', 'now', 'she', 'says', 'she', 'knows,', "you're", 'not', 'the', 'hurting',
'kind.', 'She', 'says', 'she', 'loves', 'you,', 'and', 'you', 'know', 'that', "can't",
'be', 'bad.', 'Yes,', 'she', 'loves', 'you,', 'and', 'you', 'know', 'you', 'should', '
be', 'glad.', 'Ooh!', 'She', 'loves', 'you,', 'yeh,', 'yeh,', 'yeh!', 'She', 'loves',
'you,', 'yeh,', 'yeh,', 'yeh!', 'And', 'with', 'a', 'love', 'like', 'that,', 'you', 'k
now', 'you', 'should', 'be', 'glad.', 'And', 'now', "it's", 'up', 'to', 'you,', 'I', '
think', "it's", 'only', 'fair,', 'if', 'I', 'should', 'hurt', 'you', 'too,', 'apologiz
e', 'to', 'her,', 'because', 'she', 'loves', 'you,', 'and', 'you', 'know', 'that', "ca
n't", 'be', 'bad.', 'Yes,', 'she', 'loves', 'you,', 'and', 'you', 'know', 'you', 'shou
ld', 'be', 'glad.', 'Ooh!', 'She', 'loves', 'you,', 'yeh,', 'yeh,', 'yeh!', 'She', 'lo
ves', 'you,', 'yeh,', 'yeh,', 'yeh!', 'And', 'with', 'a', 'love', 'like', 'that,', 'yo
u', 'know', 'you', 'should', 'be', 'glad.', 'And', 'with', 'a', 'love', 'like', 'that,
', 'you', 'know', 'you', 'should', 'be', 'glad.', 'And', 'with', 'a', 'love', 'like',
'that,', 'you', 'know', 'you', 'shouuuld', 'be', 'glad.', 'Yeh,', 'yeh,', 'yeh;', 'yeh
,', 'yeh,', 'yeh;', 'yeh,', 'yeh,', 'yeeeh!']
```

Get all the words from the lyrics and put them in a giant bowl/envelope

Generative model based on randomization

- Extract all words from the text and put them in a giant bowl/envelope.
- Repeat N times
  - Draw a word at random (with replacement) from bowl/envelope.
  - Print it out

Computational mapping

- How to represent the bowl of words?
  - Our old friend, the Python list

She	loves	you	yeh	• • • •	yeeh!

- Now throw a dart at this list with your eyes closed, and pick the word where your dart lands on.
- Repeat the dart throw as many times as the length of the text you want to generate.

How to extract words into a list

```
def read_file_into_word_list(filename):
    inputFile = open(filename, 'r')
    text = inputFile.read()
    words = text.split()
    return words
```

Open the file for reading

Read the entire file into a string called “text”

Split the text into a list of words, separating on space

return words as a list

Making the bowl

How to throw a computational dart

```
import random

def make_random_text_simple(words, num_words = 100):
random_text = ''
    for i in range(num_words):
        next = random.choice(words)
        random_text = random_text + ' ' + next
    return random_text
```

Putting it all together

```
words = read_file_into_word_list(‘shelovesyou.txt')
riff = make_random_text_simple(words)
print riff
```

More complex models

- The model we just developed (random drawing out of a list of words) is called a zeroth-order Markov model.
- Each word is generated independently of any other.
- However, English has sequential structure. We will now build better models to capture this structure.

The lyrics of “She loves you“

She loves you, yeh, yeh, yeh. **She loves** you, yeh, yeh, yeh.
She loves you, yeh, yeh, yeh, yeeeh!
You think you lost your love, when I saw her yesterday.
It's you she's thinking of, and she told me what to say.
She says she loves you, and you know that can't be bad.
Yes, **she loves** you, and you know you should be glad. Ooh!
She said you hurt her so, **she almost** lost her mind.
And now **she says she knows**, you're not the hurting kind.
She says she loves you, and you know that can't be bad.
Yes, **she loves** you, and you know you should be glad. Ooh!
She loves you, yeh, yeh, yeh! **She loves** you, yeh, yeh, yeh!
And with a love like that, you know you should be glad.
And now it's up to you, I think it's only fair,
if I should hurt you too, apologize to her,
because **she loves** you, and you know that can't be bad.
Yes**,** **she loves** you, and you know you should be glad. Ooh!
She loves you, yeh, yeh, yeh**!** **She loves** you, yeh, yeh, yeh!
And with a love like that, you know you should be glad.
And with a love like that, you know you should be glad.
And with a love like that, you know you shouuuld be glad.
Yeh, yeh, yeh; yeh, yeh, yeh; yeh, yeh, yeeeh!

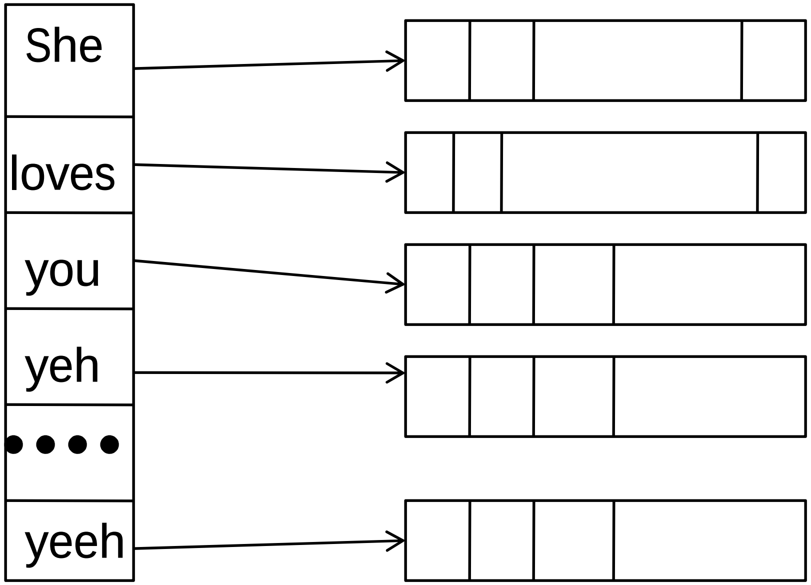
Look for patterns!

Example of structure

- In the lyrics of “She loves you” by the Beatles, what words follow the word “she”?
- She --> ['loves' , 'loves' , 'loves' , 'says' , ‘loves’, ‘said’, ‘almost’, ‘says’, ‘knows’, 'says' , 'loves' , 'loves' , 'loves' , 'loves’, ‘loves’, ‘loves’, ‘loves’]

Computational mapping

- How to represent this structure?
  - For every distinct word in the text, store a list of words that follow it immediately in the text



A prefix dictionary

Creating the prefix dictionary

- Example text: She loves you yeh yeh yeh She loves you yeh yeh yeh
- Prefix dictionary:
  - She → [loves, loves]
  - loves → [you, you]
  - you → [yeh, yeh]
  - yeh → [yeh, yeh, She, yeh, yeh]

Generation recipe

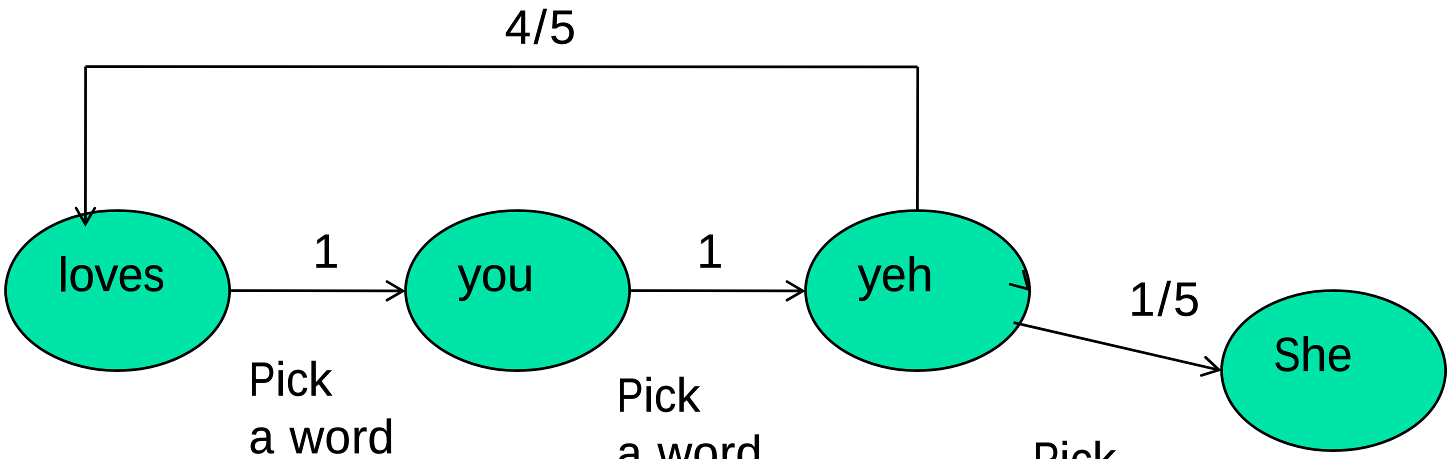
- Generate a random word w from text, and set riff = w.
- Repeat N times
  - Get list associated with word w from prefix dictionary
  - Make a random choice from that list, say w', then add w' to riff
  - Set w = w'
- Print riff

Generation example

- Random word picked at start = loves
- What word is likely to be picked after that?
  - you (probability = 1)
- What word is likely to be picked after that?
  - yeh (probability = 1)
- What word is likely to be picked after that?
  - With probability 4/5 it will be yeh, with probability 1/5 it will be She

- Prefix dictionary:
  - She → [loves, loves]
  - loves → [you, you]
  - you → [yeh, yeh]
  - yeh → [yeh, yeh, She, yeh, yeh]

The generation process



4/5

loves

1

you

1

yeh

1/5

She

Pick
a word
at random
from
prefix[“loves”]

Pick
a word
at random
from
prefix[“you”]

Pick
a word
at random
from
prefix[“yeh”]

Recipe for constructing prefix dictionary

- Example text: She loves you yeh yeh yeh She loves you yeh yeh yeh
- Prefix dictionary:
  - She → [loves]

Recipe for constructing prefix dictionary

- Example text: She loves you yeh yeh yeh She loves you yeh yeh yeh
- Prefix dictionary:
  - She → [loves]
  - Loves → [you]

Recipe for constructing prefix dictionary

- Example text: She loves you yeh yeh yeh She loves you yeh yeh yeh
- Prefix dictionary:
  - She → [loves]
  - Loves → [you]
  - you → [yeh]

How to make a prefix dictionary using Python

```
def make_prefix_dictionary(words):

    prefix = {}
    for i in range(len(words)-1):
        if words[i] not in prefix:
            prefix[words[i]] = []
        prefix[words[i]].append(words[i+1])
    return prefix
```

Generating text using the prefix dictionary in Python

```
def make_random_text(prefix, num_words=100):

    current_word = random.choice(prefix.keys())
    random_text = current_word
    for i in range(num_words-1):
        # last  word in document may not have a suffix
        if current_word not in prefix:
            break
        next = random.choice(prefix[current_word])
        random_text = random_text + ' ' + next
        current_word = next
    return random_text
```

Putting it all together

```
words = read_file_into_word_list(‘shelovesyou.txt')
prefix = make_prefix_dictionary(words)
riff = make_random_text (words)
print riff
```

The model we just built is called a first-order Markov model.

Making even more complex models

- Idea: why look at the current word alone to determine the next word? How about making a prefix dictionary indexed by two previous words, rather than a single word?

Such a model is a second-order Markov model.

Penn Jilette’s description

- Mr Shaney takes the input text and measures how many times each triple occurs - How many times does "you like to" occur in our sample - let's say twice. And how many times does "you like macrame" (for example) occur? Let's say once. All you got to do to generate output text is have Shaney print a pair of words and then choose, according to the probability of the input text, what the next word should be. So after it prints "you like " it will print the word "to" 2/3rds of the time and the word "macrame" 1/3rd of the time at random. Now, let's say, it prints "macrame". Now the current pair becomes "like macrame" (you see? this IS nonsense) - Shaney looks to see what word could follow that pair and he's off and running.

Creating a more complex prefix dictionary

- Example text: She loves you yeh yeh yeh She loves you yeh yeh yeh
- Prefix dictionary:
  - [She loves]→ [you, you]
  - [loves you] → [yeh, yeh]
  - [you yeh] → [yeh, yeh]
  - [yeh yeh] → [yeh, She, yeh]
  - [yeh She] → [loves]

Generation using the more complex prefix dictionary

- Random word pair picked at start = loves you
- What word is likely to be picked after that?
  - yeh (probability = 1)
- What word is likely to be picked after that?
  - yeh (probability = 1)
- What word is likely to be picked after that?
  - With probability 2/3 it will be yeh, with probability 1/3 it will be She

- Prefix dictionary:
  - [She loves]→ [you, you]
  - [loves you] → [yeh, yeh]
  - [you yeh] → [yeh, yeh]
  - [yeh yeh] → [yeh, She, yeh]
  - [yeh She] → [loves]