Outline

Today we'll be tackling a hodge-podge of more advanced topics in the NLP world. There will be several gear changes, but we'll be covering:

- Markov Chains
- Recurrent Neural Networks/LSTMs
- Seq2Seq

Markov Chains

- Stochastic processes that have "no memory" are often called Markov processes.
- The idea is that if I understand the probability of all changes from my current state, I can let dice rolls describe my next move. Let's take a look at the most common example: "The Drunkard's Walk"

BAR

Joe has no memory or coordination. He would like to end up in the bathroom, but he actually has no ability to "aim" for it. Every time he takes a step, it will be in a random direction – completely independent of his previous steps.

RESTROOM

BAR

After taking his first step, Joe has no sense of where he just was, nor any more sense of where the bathroom is. He just knows, "I'm here now, and my next step is equally likely to be 1 unit in any direction."

RESTROOM

BAR

RESTROOM

This process repeats indefinitely. This is sometimes called "Brownian Motion" and is characteristic of having all "next steps" be equally likely. Let's take a look.
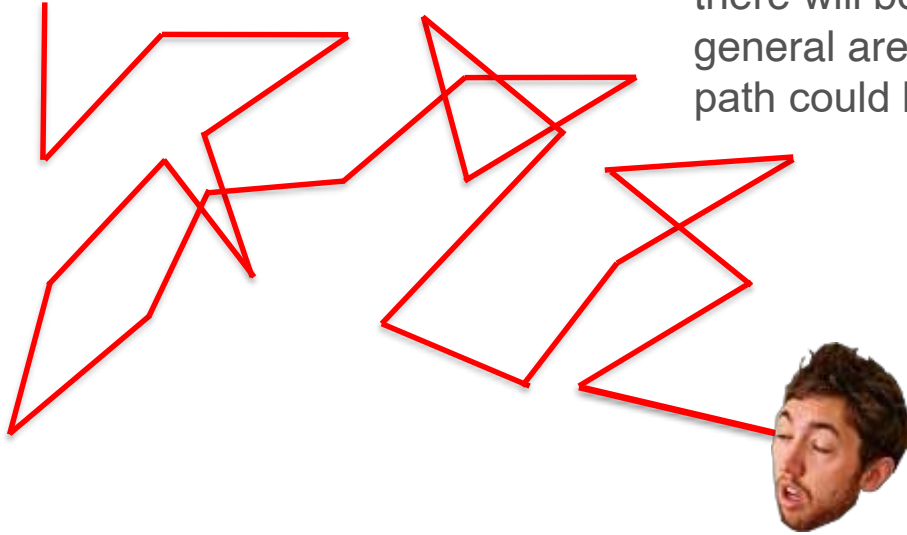
BAR

RESTROOM

BAR

RESTROOM

BAR

RESTROOM

BAR

In this Markov chain, it's possible for Joe to eventually make it to the restroom by having many fortuitous, but random, drunken steps. However, there will be a lot of stumbling around in the same general area for a while. One example fortuitous path could be…

RESTROOM

Markov Chains

- There's nothing in Markov chains that determines that all steps have to be equally likely though. Imagine I only ever eat fruit, vegetables, or steak. Let's look at a table that shows the probabilities for my next meal, based on my current meal.

	Next: Steak	Next: Fruit	Next: Vegetable
Current: Steak	10%	40%	40%
Current: Fruit	60%	5%	35%
Current: Vegetable	98%	1%	1%

	Next: Steak	Next: Fruit	Next: Vegetable
Current: Steak	10%	40%	40%
Current: Fruit	60%	5%	35%
Current: Vegetable	98%	1%	1%

Vegetable

Current Meal

	Next: Steak	Next: Fruit	Next: Vegetable
Current: Steak	10%	40%	40%
Current: Fruit	60%	5%	35%
Current: Vegetable	98%	1%	1%

Vegetable ✗

DICE →

Steak

Current Meal

NLP with Markov Chains

- Let's apply this concept to NLP by looking at an example sentence

    "Both the brown fox and the brown dog slept."

- How can we build a probabilistic understanding of this text?

"Both the brown fox and the brown dog slept."

Let's build a dictionary that tracks the "current" two words, and what word comes after them. We can treat the two words as the "current state" and the next word as a "next available" state.

"Both the brown fox and the brown dog slept."

{ (Both, the): [brown] }

"Both the brown fox and the brown dog slept."

{ (Both, the): [brown],
  (the, brown): [fox], }

"Both the brown fox and the brown dog slept."

{ (Both, the): [brown],
  (the, brown): [fox],
  (brown,fox): [and], }
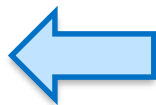
"Both the brown **fox and** **the** brown dog slept."

{ (Both, the): [brown],
  (the, brown): [fox],
  (brown,fox): [and],
  (fox, and): [the], }

"Both the brown fox **and the** **brown** dog slept."

{ (Both, the): [brown],
  (the, brown): [fox],
  (brown,fox): [and],
  (fox, and): [the],
  (and, the): [brown],}

{ (Both, the): [brown],
 (the, brown): [fox, dog],
 (brown,fox): [and],
 (fox, and): [the],
 (and, the): [brown],}

Now if we are in a current state of "the brown", both "fox" and "dog" are equally likely to occur!

So we have a state dependent chain of probabilities.

We can use Markov Chains here!

Text Generation with Markov Chains

{ (Both, the): [brown],
  (the, brown): [fox, dog],
  (brown,fox): [and],
  (fox, and): [the],
  (and, the): [brown],
  (brown, dog): [slept] }

Let's start with a sentence seed of "and the" and this dictionary of word relationships and roll some dice.

"and the "

Text Generation with Markov Chains

{ (Both, the): [brown],
  (the, brown): [fox, dog],
  (brown,fox): [and],
  (fox, and): [the],
  (and, the): [brown],
  (brown, dog): [slept] }

100%
brown

"and the brown "

Let's start with a sentence seed of "and the" and this dictionary of word relationships and roll some dice.

Text Generation with Markov Chains

{ (Both, the): [brown],
  (the, brown): [fox, dog],
  (brown,fox): [and],
  (fox, and): [the],
  (and, the): [brown],
  (brown, dog): [slept] }

Let's start with a sentence seed of "and the" and this dictionary of word relationships and roll some dice.

50% dog, 50% fox.
ROLL DICE.

"and the brown fox "

Text Generation with Markov Chains

{ (Both, the): [brown],
  (the, brown): [fox, dog],
  (brown,fox): [and],
  (fox, and): [the],
  (and, the): [brown],
  (brown, dog): [slept] }

Let's start with a sentence seed of "and the" and this dictionary of word relationships and roll some dice.

Because of our limited selection of words, we are now stuck completing the sentence.

"and the brown fox and the brown dog slept"

Example output with a bigger corpus

Input: A few H.P. Lovecraft Books
Markov System: Two word current state.

```
the moon, and the newcomers and there are things about the lock
on the deck of a monstrous odour...senses transfigured...
boarding at that moment it struck me, known all along the banks
were bent and dirty, he was shown the thing vanished with him.
the dusk of the blasphemously stupendous bulk of our ascent of
the real article, when marsh suddenly swerved from abstractions
to the town and on april 2nd was driven by some remoter advancing
bulk--and then came the final throaty rattle came. dr houghton of
aylesbury, who had reigned as a beginning of a laugh. the horror
```

NLP with Markov Chains

- Markov chains work "well" if we have a BIG corpus to draw from originally. Then they won't get stuck repeating the same sentences over and over.
- Even working "well" they tend to be a bit gibberish-y. That's the price to pay for allowing topic jumps whenever words are shared. However, Markov chains can generate some good stuff too.
- One of the most simplistic, but still decently powerful, methods of text generation.
- What would happen if we allowed more than 2 words to define our current state? We'll see in the exercise.

A quick review of neural nets without NLP

Hidden Layer

Output Layer
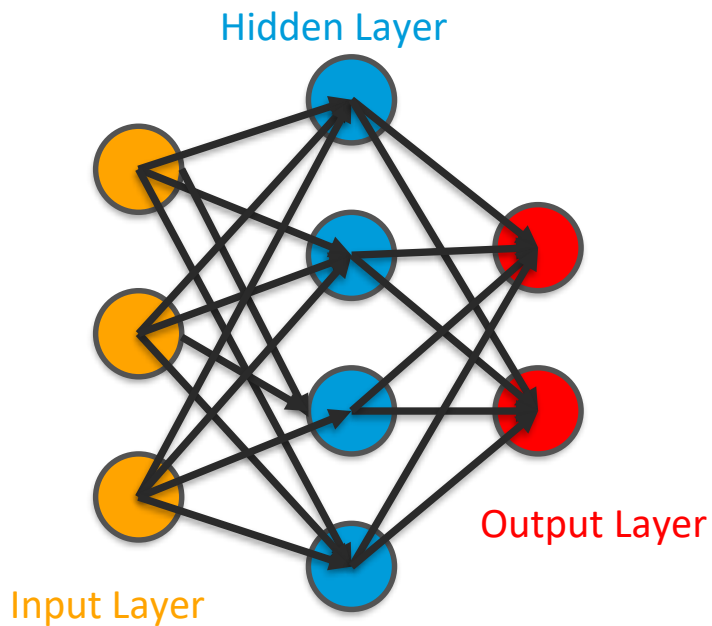
Input Layer

- Input layer is a set of features, each arrow represents a weight (float number) that tells us how much each input contributes to each following step.
- Each node in the hidden layer is some combination of all the inputs. The hidden layer acts as the 'input' for the output layer.
- Backpropagation allows us to adjust the weights to improve accuracy and find the 'correct' way to combine the inputs and hidden layers to get the best possible results.

Neural Nets with NLP (Traditional)

- We can pre-process our data and assign every word some ID.

<div align="center">

0   1   2   3

The cat ran fast.

</div>

- Then we can use these IDs as inputs to our training by converting them to one-hot-encoded vectors.

- If we want to do classification, we are can then feed it through our neural net.

Recurrent Neural Nets (RNNs)

- RNN's allow us to remember what happened in our last decision making process. So if we're asking it to learn to spell "HELLO" it needs to know what the previous letters are to make a good decision. You can't just start at 'L' and decide if you're spelling the word right, and neither can it. So we feed the previous decision making back into the network at the next step. It can then learn: "I've got as input "HEL" and my last prediction was that last "L," so I should probably add another "L."
- This adds an understanding of "timing" to our networks.

Keras – Simplifying LSTMs in Python

Keras is a Python package that makes building and training TensorFlow neural networks really simple. We'll be working with the "Sequential" model which lets you add layers one at a time. As an example, let's see how to build a 1-layer LSTM model with 10 hidden nodes.

```python
from keras.models import Sequential
from keras.layers import Dense, Activation, LSTM

model = Sequential()
model.add(LSTM(10, input_shape=(TIMESTEPS, FEATURE_LENGTH)))
model.add(Dense(NUMBER_OF_OUTPUT_NODES))
model.add(Activation('softmax'))
```

Keras – Simplifying LSTMs in Python

Why does Keras expect a 2D matrix as input for the LSTM?
It needs a list of lists, ordered by time!

If we were trying to teach it the alphabet, we'd need to send in:

`[['A'], ['B'], ['C']]` as input to get `['D']` back out. Keras knows that in this format 'A' comes before 'B'. Or more realistically, we'd need to send in one-hot encoded versions like so:

`[[1,0,0,0], [0,1,0,0], [0,0,1,0]]` → `[0,0,0,1]`

LSTMs for Text Generation

- Using this setup, we can chop our corpus up into character level sequences. "Hello" could be come [['H'],['e'],['I'],['I']] with a corresponding label of ['o']. If we do that for the whole corpus, the LSTM can learn how to spell and when spaces and line breaks tend to happen.
- How well could this possible work?

LSTMs for Text Generation

Pretty well, actually. This method has been used to:

- Learn C++ syntax (while writing mostly gibberish code)
- Write fake math proofs in LaTeX
- Write fake Wikipedia articles in Markdown
- Write fake Shakespeare plays (including character names and appropriate line breaks)
- Many many more…

- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTMs for Text Generation

For the exercise, we will be going through the process of building an LSTM model with Keras, one-hot encoding all of our characters, building out an ID-to-Character conversion, and training/generating with our model. It will be guided, such that each question is a step in the process.

One pre-warning: training these models can take a long time. So don't be surprised if your initial runs are a bit gibberishy. If you start to see it spelling words correctly, you're definitely headed in the right direction. It just may take some more training. This is one of the reasons we love GPUs. They take what can be days of training and make it hours.

LSTMs for Text Generation - Data Prep

Input:

```
# Step 1: generate input data
# Inputs: one hot encoded characters: X

print(X[0]) # Sequence of three one hot encoded vectors, of length 59
```

Output:

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

LSTMs for Text Generation - Data Prep

Input:

```
# Step 2: generate output data
# Inputs: one hot encoded characters: y

print(y[0:3]) # First three "output" vectors we are trying to predict
```

Output:

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

LSTMs for Text Generation - Define Model

```python
# Step 3: define model architecture

from keras.models import Sequential
from keras.layers import LSTM, Dense, Activation

sequence_length = 40
model = Sequential()
model.add(LSTM(128, input_shape=(MAX, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
```

LSTMs for Text Generation - Compile and Train

```python
# Step 4: compile and train model

from keras.optimizers import RMSprop

optimizer = RMSprop(lr=0.01)

model.compile(loss='categorical_crossentropy', optimizer=optimizer)

for iteration in range(num_epochs):

    model.fit(X, y, batch_size=128, nb_epoch=1)
```
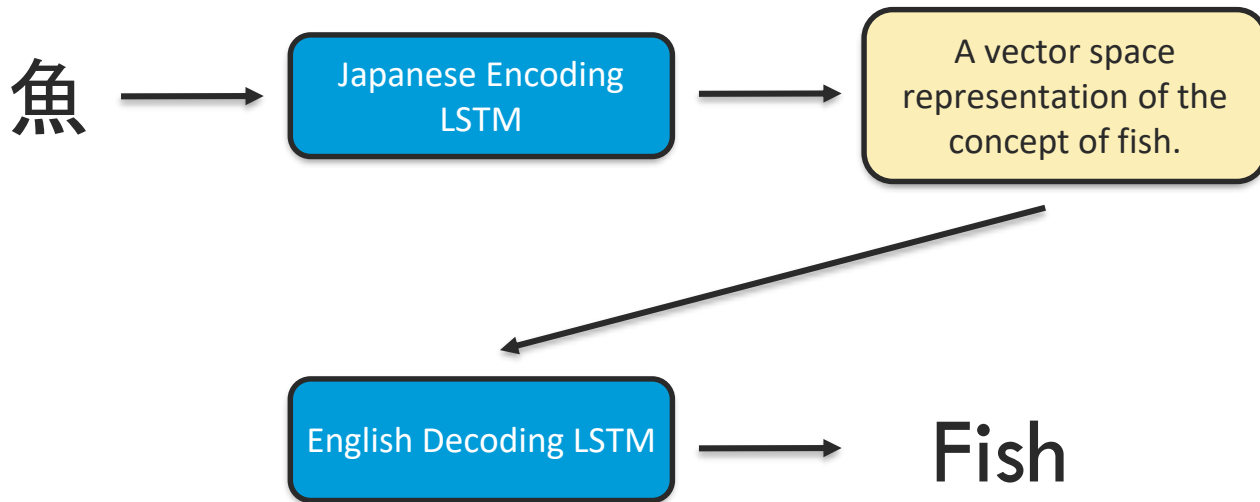
Seq2Seq and other Advanced NLP

Advanced NLP Applications

There are several really interesting uses of NLP techniques happening at the moment that warrant some discussion. We'll overview a few that are linked to Seq2Seq models:

- Automated Language Translation
- Questions answering / Task taking (Siri/Alexa/Ok Google/Chatbots)
- Speech Recognition / Text-to-Speech

Language Translation: Seq2Seq

- Oversimplifying, a Seq2Seq model is a set of two deep LSTM networks: one for encoding and one for decoding.

魚 → Japanese Encoding LSTM → A vector space representation of the concept of fish.

English Decoding LSTM → Fish

Language Translation: Seq2Seq

- Both Tensorflow and Google have released Seq2Seq models you can play with (https://www.tensorflow.org/tutorials/seq2seq and https://google.github.io/seq2seq/)

- It doesn't only apply to translation. Anything that can be encoded and then decoded can use Seq2Seq. Examples: Image captioning, summarizing documents, conversation modeling, etc.

Question Answering

- Who was the 11th president of the United States?

- The 11th president of the USA was…?

- Who was George Dallas' running mate?

- Who was the 11th president?

- Who was the president before Zachary Taylor?

That's a lot of different ways to ask about James K. Polk. How can we teach a machine to know all of these are the same question?

Question Answering

- Semantic Indexing is step one. This can be similar to Seq2Seq – can we encode this question into some concepts that we can use to ask the appropriate question that the machine can understand.

Let's encode this into a 3D tuple: Who was the 11th president of the USA?

(president, 11, USA) $\rightarrow$ Search knowledge base for answer $\rightarrow$ James Polk

Question Answering

- With a large enough database that's encoded in a consistent method, all we need to train is the semantic comprehension. This can be done with seq2seq models.

- Downside: limited by DB and encoding/decoding accuracy.

- Another common QA method is document comprehension: can it read a document and extract the answer? If so, how can I determine what documents to feed it? Also deep learning questions.

- Chatbots are an extreme form of this where the answers need to be found and then sentences need to be generated to make the answer seem like natural conversation.
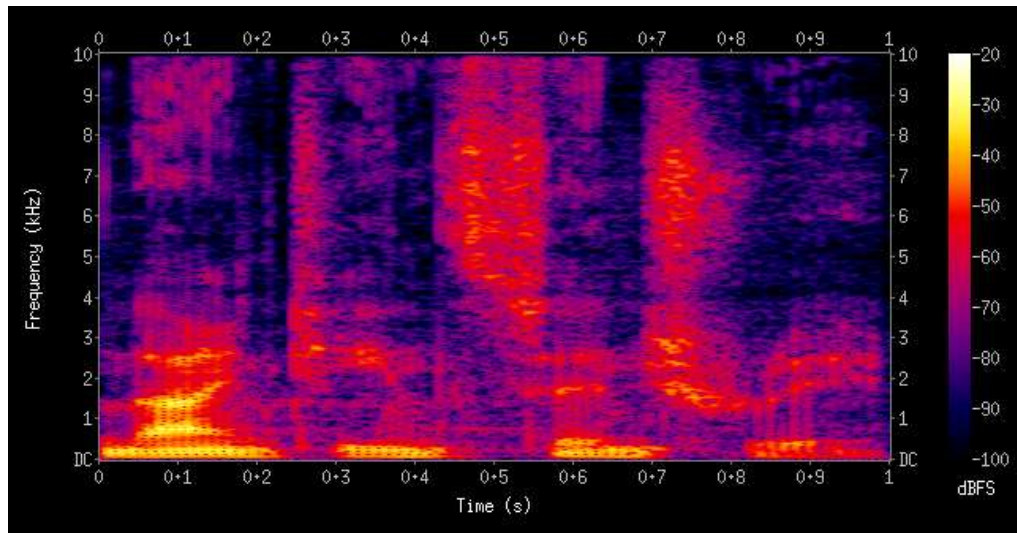
Speech Recognition

One　　　Two　　　Three　　　Four　　　Five　　　Six

Sound signals are just amplitude vs time, so it's hard to gather too much information from them. However, we can extract more useful things by looking at spectrograms.

Speech Recognition

Spectrograms encode the frequency and amplitude overtime. We can analyze these as a 2D matrix to try to extract speech, using convolutional neural network architectures and Seq2Seq style encoding and decoding.