Recurrent Neural Networks/Long-Short Term Memory for Natural Language Processing

And applications in clinical note analytics

Yanshan Wang, PhD
Mayo Clinic
wang.yanshan@mayo.edu

AGENDA

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

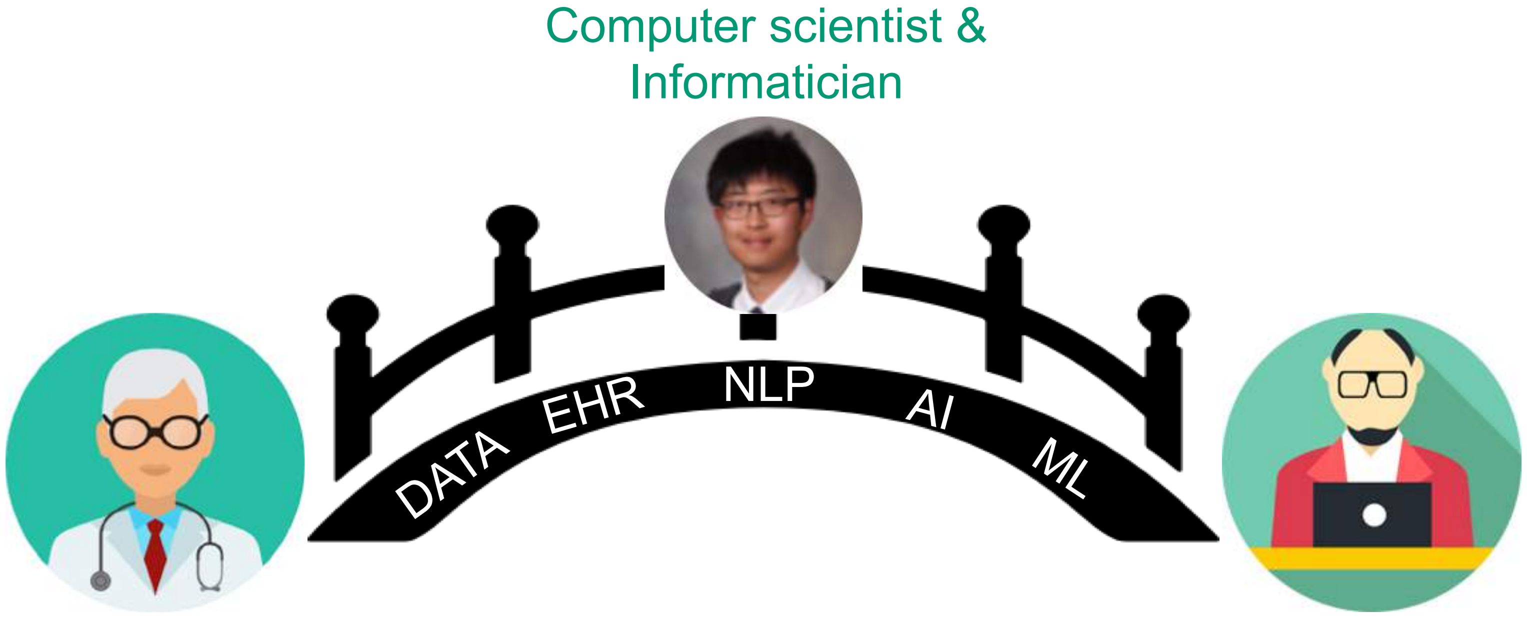
AGENDA

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

MAYO CLINIC

INTRODUCTION

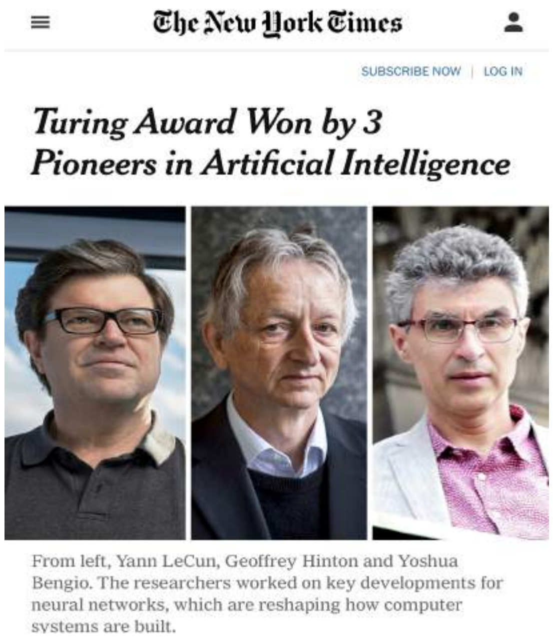
- Who am I?



- How can you contact me?
  - Email: wang.yanshan@mayo.edu
  - LinkedIn, Twitter (yanshan_wang)

WHY TAKE THIS TUTORIAL

- Artificial Intelligence (AI) is one of the most interesting fields of research today.
- The growth of and interest in AI is due to the recent advances in deep learning.
- Language is the most compelling manifestation of intelligence.


The New York Times

SUBSCRIBE NOW | LOG IN

Turing Award Won by 3 Pioneers in Artificial Intelligence

From left, Yann LeCun, Geoffrey Hinton and Yoshua Bengio. The researchers worked on key developments for neural networks, which are reshaping how computer systems are built.

Goal of this tutorial

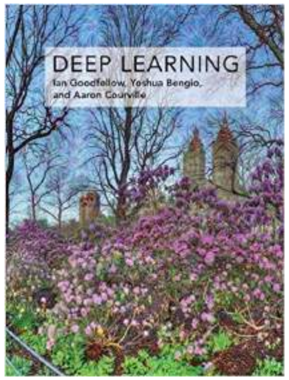
- To get an understanding of some core concepts in deep learning (RNN/LSTM) and natural language processing.
- To connect theoretical with practical knowledge.
- To get an introduction into programming of RNN/LSTM in Python.

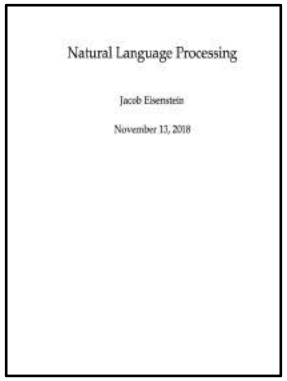
PREREQUISITES

- Maths
  - Linear Algebra
  - Probability
- Machine learning
  - Evaluating ML models (train/validation/test, cross validation, etc.)
  - Overfitting, generalization
  - Optimization (objective function, stochastic gradient descent)
- Programming
  - Python (e.g., Keras, NumPy, TensorFlow, etc.)

SUGGESTED READING

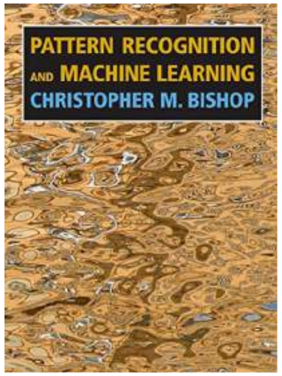
- Deep learning for NLP Courses
  - Oxford Deep Learning for NLP class: https://github.com/oxford-cs-deepnlp-2017/lectures
  - Stanford CS 224D: Deep Learning for NLP class: http://cs224d.stanford.edu/syllabus.html
- Books




Natural Language Processing

Jacob Eisenstein

November 13, 2018



- Resources
  - https://github.com/andrewt3000/DL4NLP
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Suggested reading

- Papers (general NLP)
  - A neural probabilistic language model. Bengio et al. 2003
  - Efficient estimation of word representations in vector space. Mikolov et al. 2013.
  - Long short-term memory. Hochreiter and Schmidhuber. 1997.
  - Recurrent neural network based language model. Mikolov et al. 2010.
  - Generating sequences with recurrent neural networks. Graves. 2013.
  - On the state of the art of evaluation in neural language models. Melis et al. 2018
- Paper (clinical NLP)
  - Clinical information extraction applications: A literature review. Wang et al. 2018
  - Using clinical Natural Language Processing for health outcomes research: Overview and actionable suggestions for future advances. Velupillai et al. 2018.

MAYO CLINIC

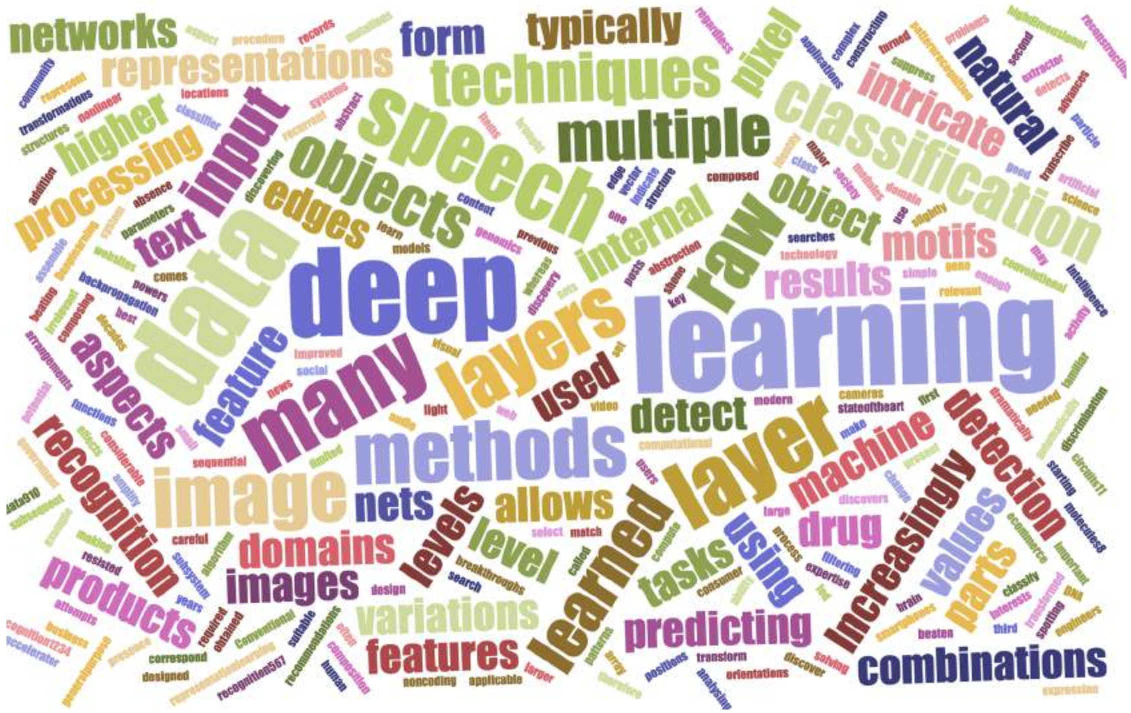
Agenda

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

MAYO CLINIC

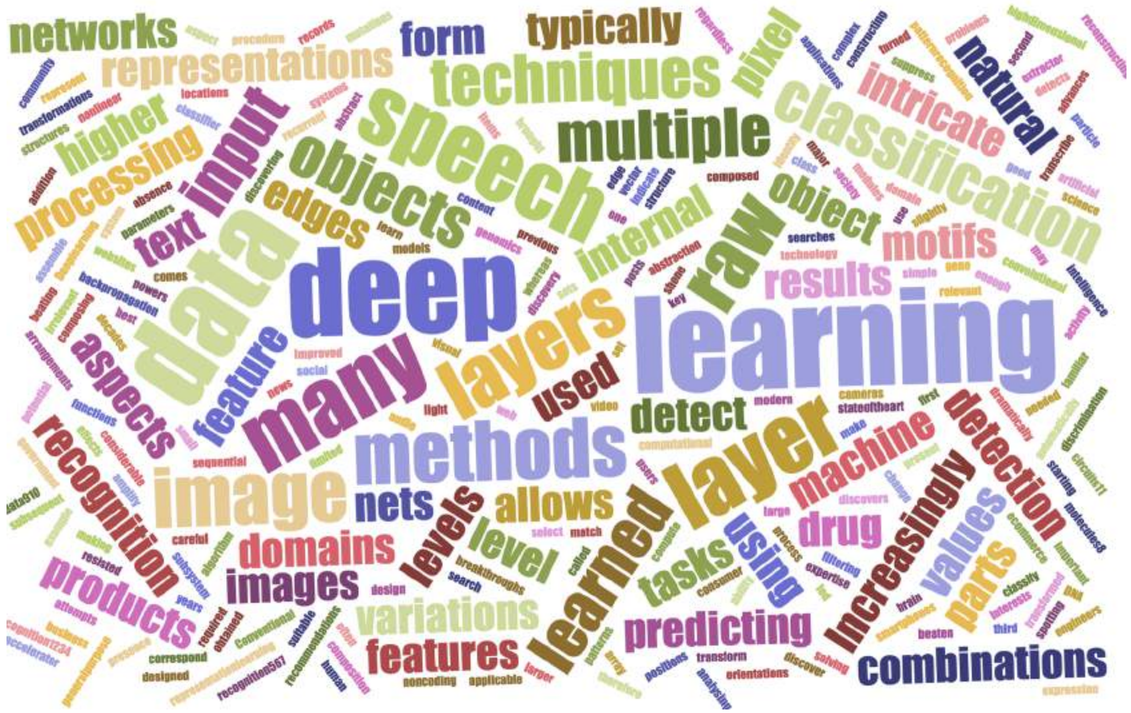
Natural Language Processing

- What is NLP?
  - "Natural language processing (NLP) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data." (Wikipedia)



Natural Language Processing

- What is NLP?
  - "Natural language processing (NLP) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data." (Wikipedia)



Question Answering

IBM Watson

Voice Assistant

Source: https://www.youtube.com/watch?v=BkpAro4zIwU

INFORMATION EXTRACTION

A presenilin 1 mutation (*Ser169Pro*) associated with *early-onset AD* and *myoclonic seizures*.

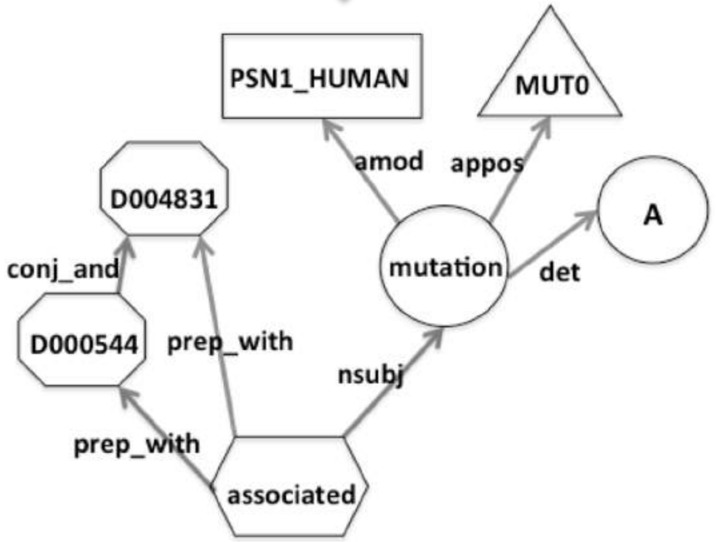
Information Extraction

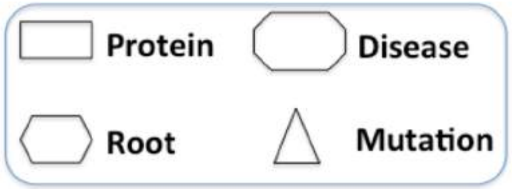
A **presenilin 1** mutation (*Ser169Pro*) associated with ***early-onset AD*** and ***myoclonic seizures***.

Entity normalization

A **PSN1_HUMAN** mutation (*MUT0*) associated with ***D000544*** and ***D004831***.

Dependency parser

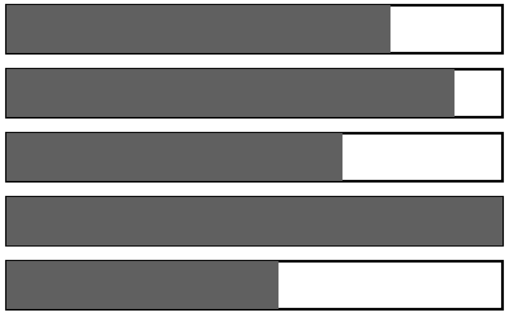




Information Extraction & Sentiment Analysis

Attributes:

zoom
affordability
size and weight
flash
ease of use

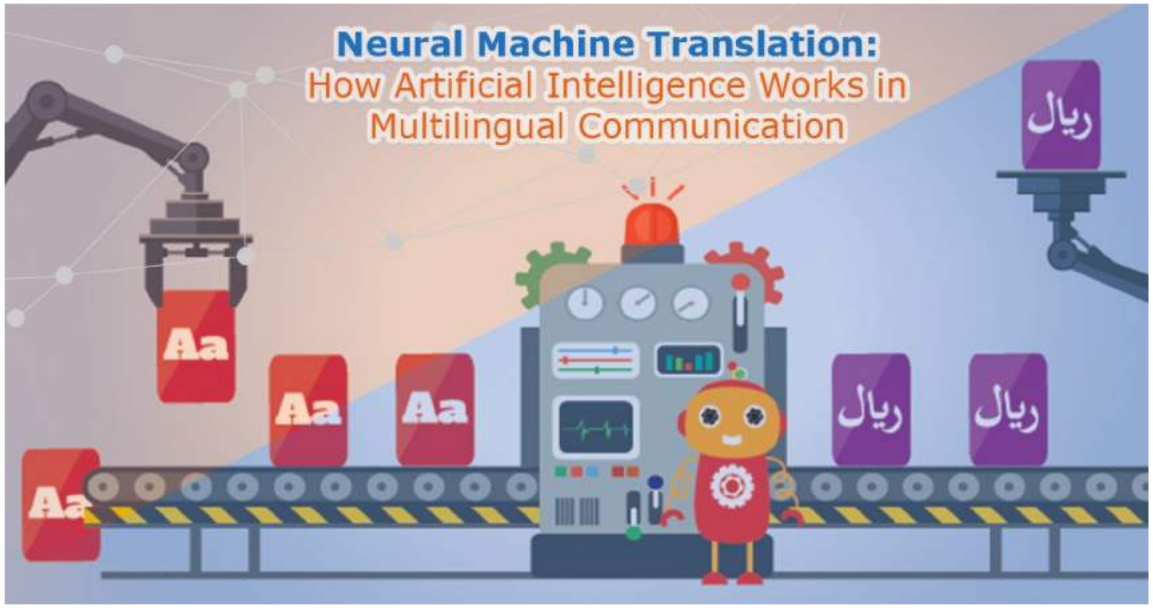


Reviews:

- ✓ nice and compact to carry!
- ✓ since the camera is small and light, I won't need to carry around those heavy, bulky professional cameras either!
- ✗ the camera feels flimsy, is plastic and very light in weight you have to be very delicate in the handling of this camera

Source: https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html

Machine Translation


Neural Machine Translation:
How Artificial Intelligence Works in Multilingual Communication
Aa
ريال

Source: https://chatbotslife.com/machine-translation-by-artificial-intelligence-9d7a732a4ee1

Source: http://www.tagxp.com/m/html/2372.html

AGENDA

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

MAYO CLINIC

How to represent Natural Language

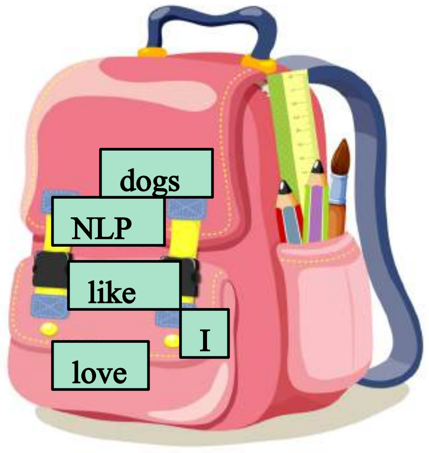
- Natural language text = sequences of discrete symbols (e.g. words).

I love NLP and I like dogs

- Vector representations of words: Vector Space Model
  – Bag-of-words

Vocabulary list

	I	love	NLP	and	like	dogs
I	1	0	0	0	0	0
love	0	1	0	0	0	0
NLP	0	0	1	0	0	0
like	0	0	0	0	1	0



MAYO CLINIC

How to represent Natural Language

Sparse representation

I	=[	1	0	0	0	0	0	]
love	=[	0	1	0	0	0	0	]
NLP	=[	0	0	1	0	0	0	]
like	=[	0	0	0	0	1	0	]

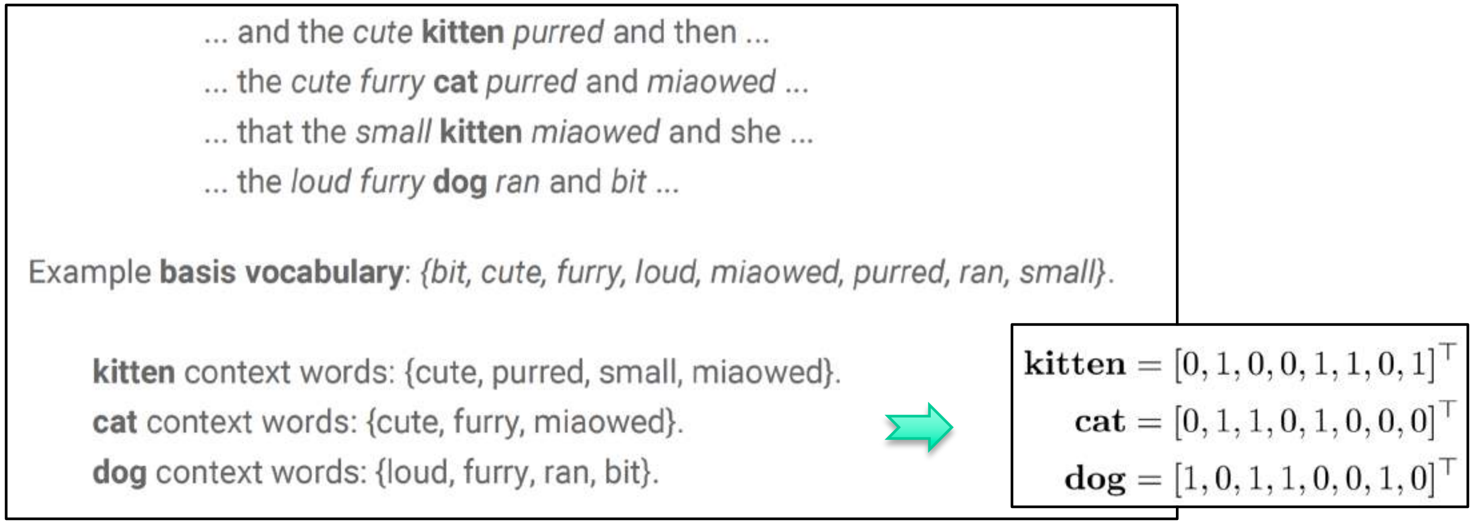
1-of-N (one-hot) encoding

- Drawbacks
  - love=[0,1,0,0,0,0] AND like=[0,0,0,0,1,0] = 0!
  - Using such an encoding, there's no meaningful (semantic) comparison we can make between word vectors other than equality testing.

MAYO CLINIC

How to represent Natural Language

- Semantic representations
  - Count-based methods
    - Define a basis vocabulary C of context words.
    - Define a word window size w.
    - Count the basis vocabulary words occurring w words to the left or right of each instance of a target word in the corpus.
    - Form a vector representation of the target word based on these counts.



... and the *cute* **kitten** *purred* and then ...

... the *cute furry* **cat** *purred* and *miaowed* ...

... that the *small* **kitten** *miaowed* and she ...

... the *loud furry* **dog** *ran* and *bit* ...

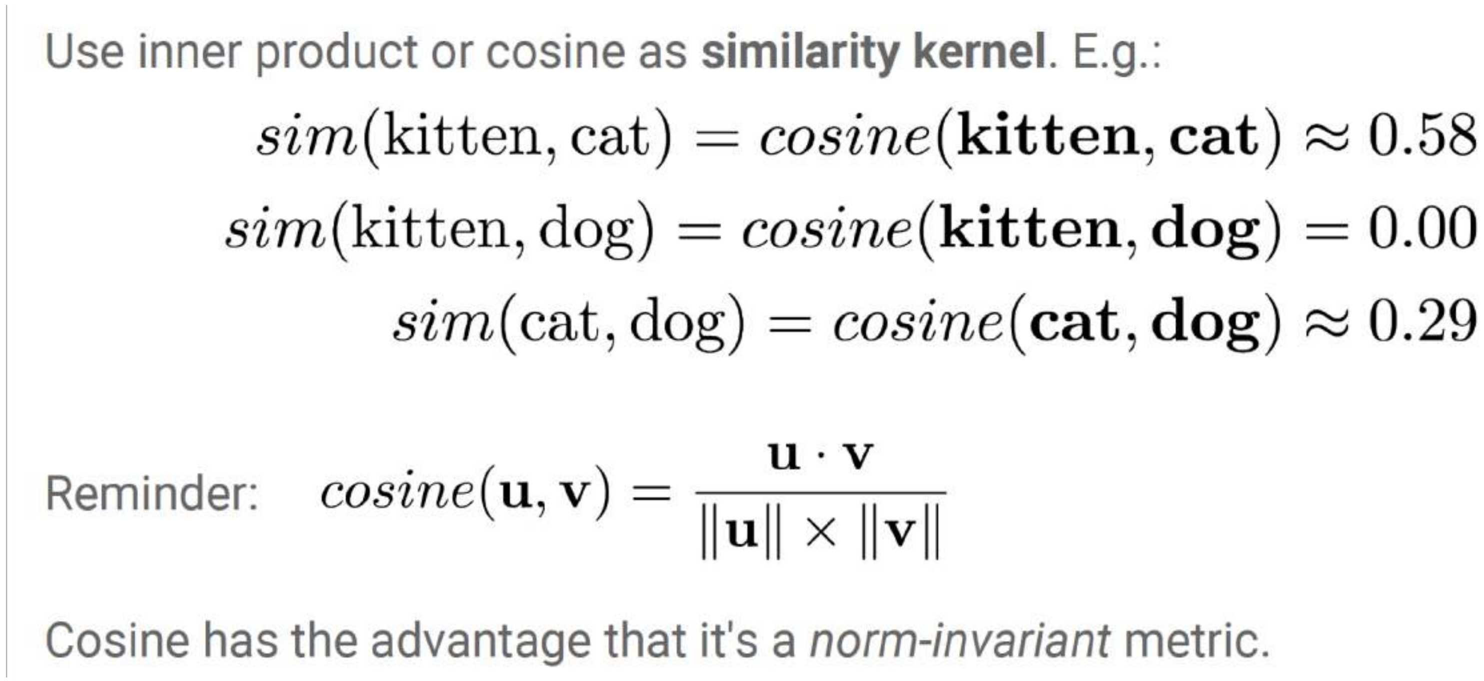
Example **basis vocabulary**: {*bit, cute, furry, loud, miaowed, purred, ran, small*}.

kitten context words: {cute, purred, small, miaowed}.
cat context words: {cute, furry, miaowed}.
dog context words: {loud, furry, ran, bit}.

$$\begin{aligned}\textbf{kitten} &= [0,1,0,0,1,1,0,1]^\top\\ \textbf{cat} &= [0,1,1,0,1,0,0,0]^\top\\ \textbf{dog} &= [1,0,1,1,0,0,1,0]^\top\end{aligned}$$

Source: Oxford Deep Learning for NLP class

How to represent Natural Language



Use inner product or cosine as **similarity kernel**. E.g.:

$$sim(\text{kitten}, \text{cat}) = cosine(\mathbf{kitten}, \mathbf{cat}) \approx 0.58$$

$$sim(\text{kitten}, \text{dog}) = cosine(\mathbf{kitten}, \mathbf{dog}) = 0.00$$

$$sim(\text{cat}, \text{dog}) = cosine(\mathbf{cat}, \mathbf{dog}) \approx 0.29$$

Reminder: $cosine(\mathbf{u}, \mathbf{v}) = \dfrac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \times \|\mathbf{v}\|}$

Cosine has the advantage that it's a *norm-invariant* metric.

MAYO CLINIC

Language models

- The simple objective of modelling the next word given the observed history of natural language

$$p(\cdot\,|\mathit{There\ she\ built\ a})$$

- With more context we are able to use our knowledge of both language and the world to heavily constrain the distribution over the next word:

$$p(\cdot\,|\mathit{Alice\ went\ to\ the\ beach.There\ she\ built\ a})$$

- Language modelling is a time series prediction problem in which we must be careful to train on the past and test on the future.

Language models

- Language models learn “meaning” of a word using dense representation

Dense representation

I	=[	0.99	0.05	0.1	0.87	0.1	0.1	]
love	=[	0.1	0.85	0.99	0.1	0.83	0.09	]
NLP	=[	0.67	0.23	0.01	0.02	0.01	0.81	]
like	=[	0.1	0.73	0.99	0.05	1.79	0.09	]

  – Simply examining a large corpus it’s possible to learn word vectors that are able to capture the relationships between words in a surprisingly expressive way.

- Popular language models
  – Topic models: LSA, pLSA, LDA, etc.
  – Neural language models

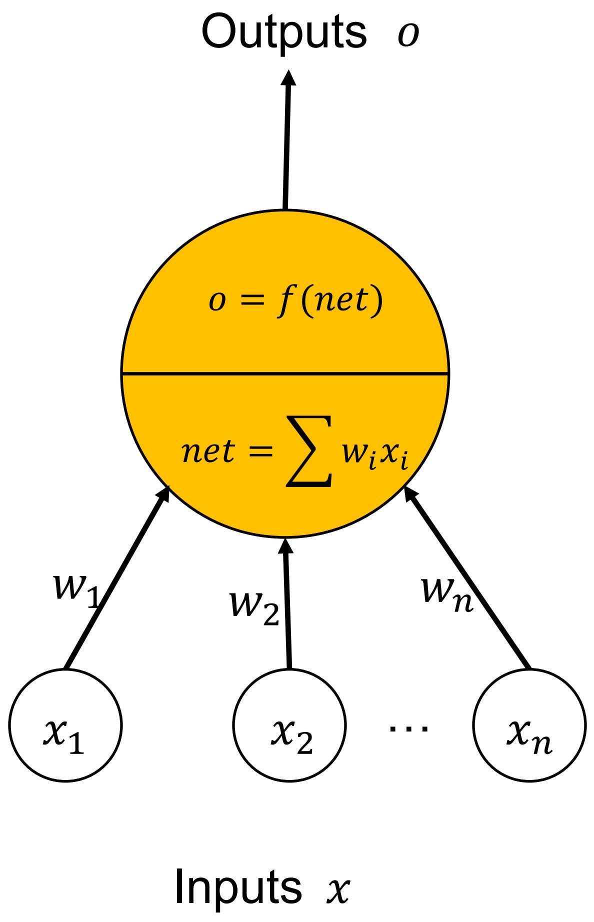
AGENDA

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

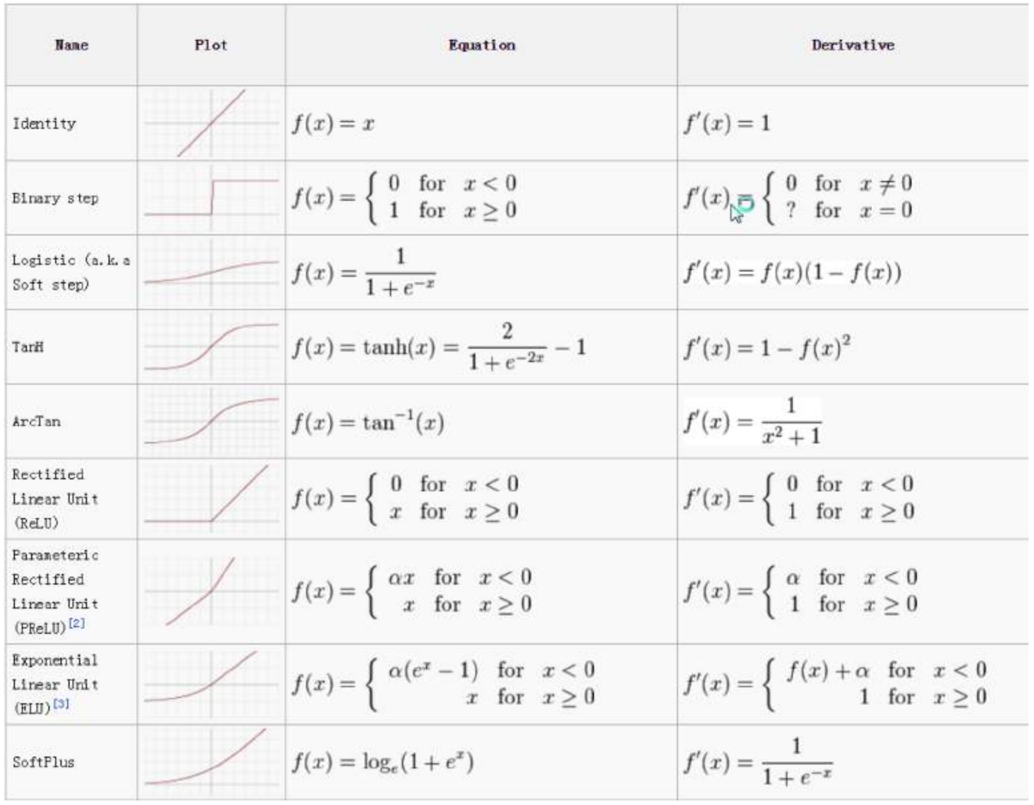
MAYO CLINIC

DEEP LEARNING BASICS

- Artificial neuron is a nonlinear processing unit



Activation functions f

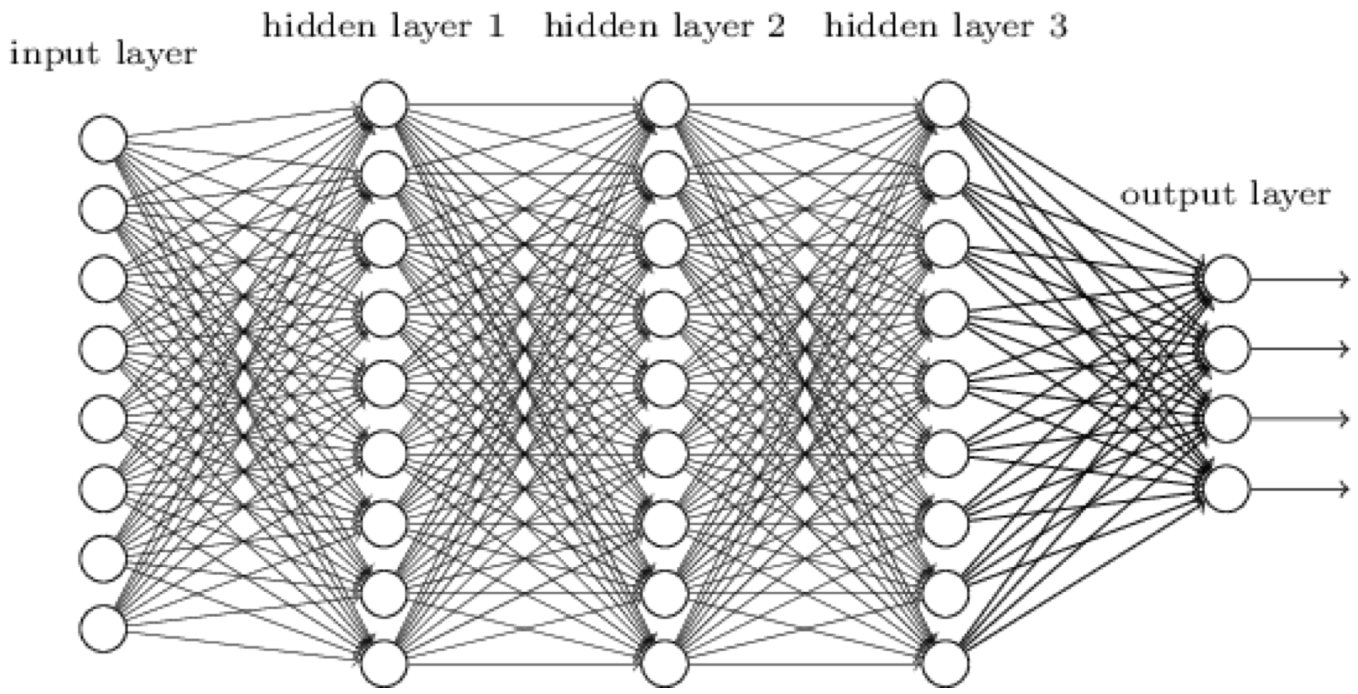


Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Source: https://en.wikipedia.org/wiki/Activation_function

Deep learning

- Deep learning is a class of machine learning algorithms that use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation.



AGENDA

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

MAYO CLINIC

Agenda

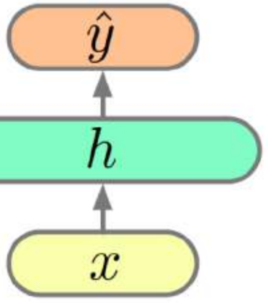
- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

Neural language models

- Feed forward network

$$h = g(Vx + c)$$

$$\hat{y} = Wh + b$$

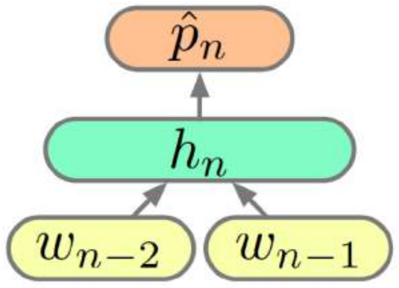


- Trigram neural language model

$$h_n = g(V[w_{n-1}; w_{n-2}] + c)$$

$$\hat{p}_n = softmax(Wh_n + b)$$

  - w_i are one hot vectors, $\hat{p}_n$ is distribution
  - $|w_i|$ is the size of vocabulary which is usually very large.

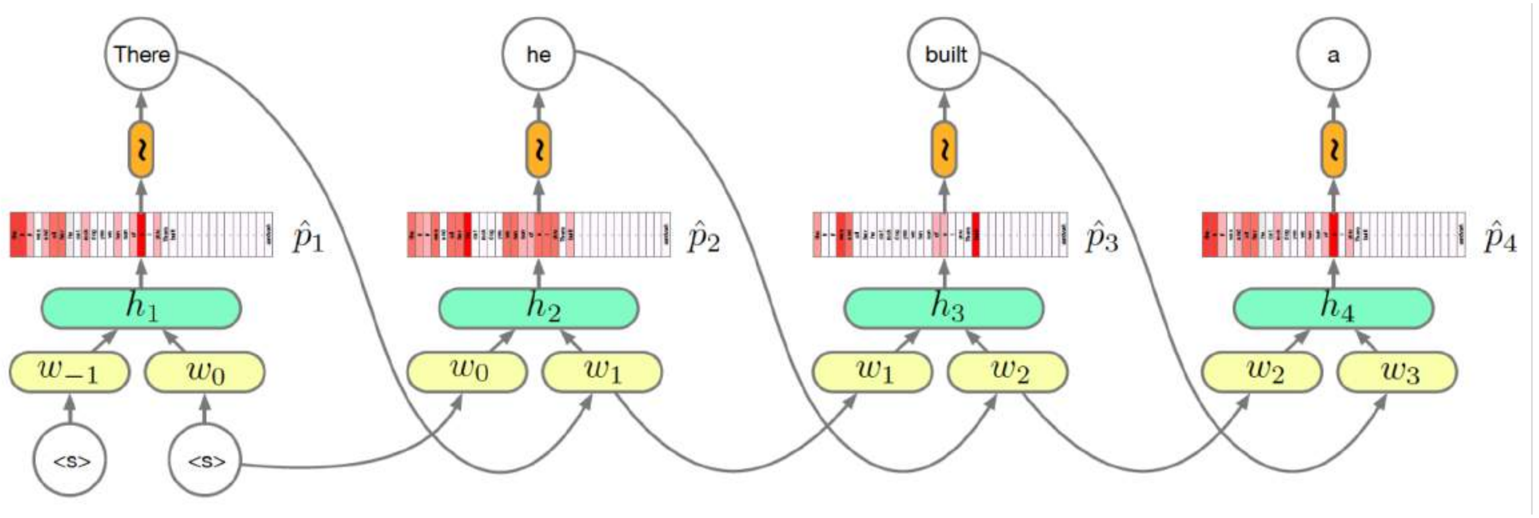


Source: Oxford Deep Learning for NLP class

NEURAL LANGUAGE MODELS

- When learning, the output is a probability distribution instead of one word. When generating text we choose only one of the words ourselves given the probabilities and feed that back into the network. This is called sampling

$$w_n | w_{n-1}, w_{n-2} \sim \hat{p}_n$$

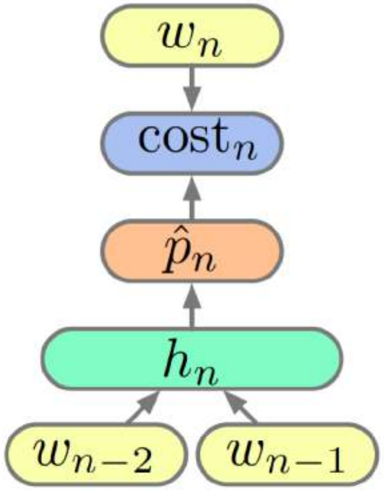


Source: Oxford Deep Learning for NLP class

Neural language models

- Objective function
  - The usual training objective is the cross entropy of the data given the model (MLE):

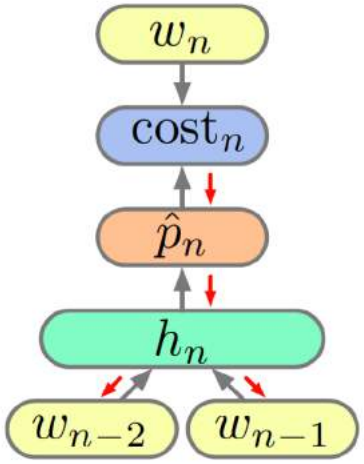
$$L = -\frac{1}{N} cost_n(w_n, \hat{p}_n)$$



  - Calculating the gradients is straightforward with back propagation:

$$\frac{\partial L}{\partial W} = -\frac{1}{N}\sum_n \frac{\partial cost_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial W}$$

$$\frac{\partial L}{\partial V} = -\frac{1}{N}\sum_n \frac{\partial cost_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$

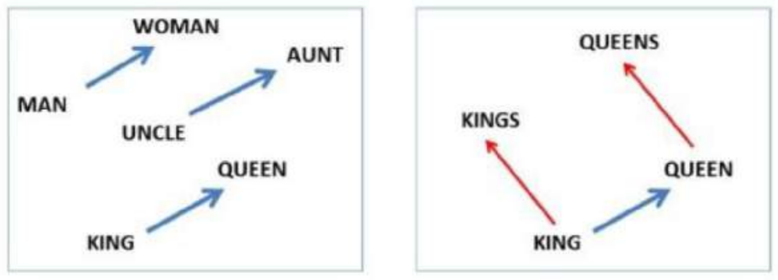


Source: Oxford Deep Learning for NLP class

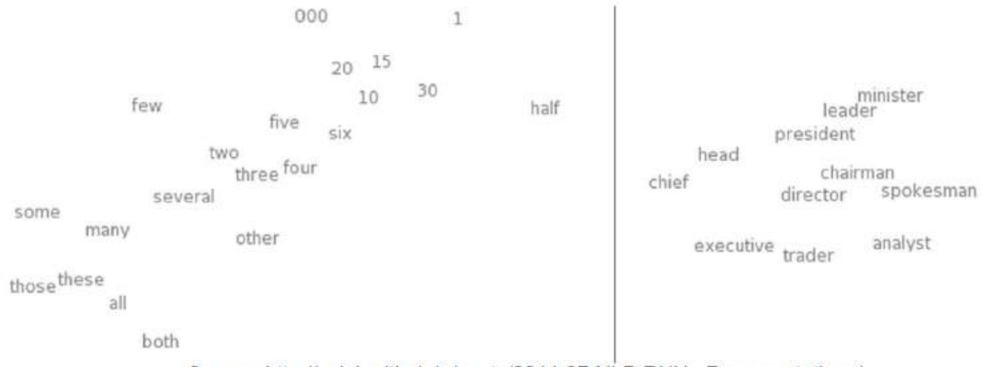
Neural language models

- The insight behind neural language models is to treat word prediction as a discriminative learning task.
- The goal is to compute the probability p(word|context). Rather than directly estimating the word probabilities from (smoothed) relative frequencies, we can treat language modeling as a machine learning problem, and estimate parameters that maximize the log conditional probability of a corpus.
- W and V are model parameters, $\hat{p}_n$ can be computed in various ways, depending on the neural model.
- Vector representations of words are sometimes called word embeddings.

WORD EMBEDDINGS



Source: http://nlp.yvespeirsman.be/blog/visualizing-word-embeddings-with-tsne/



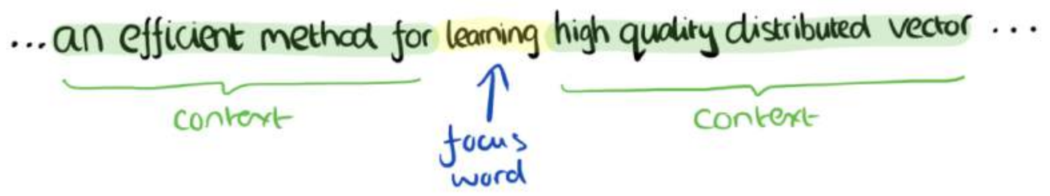
Source: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

1. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).
2. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

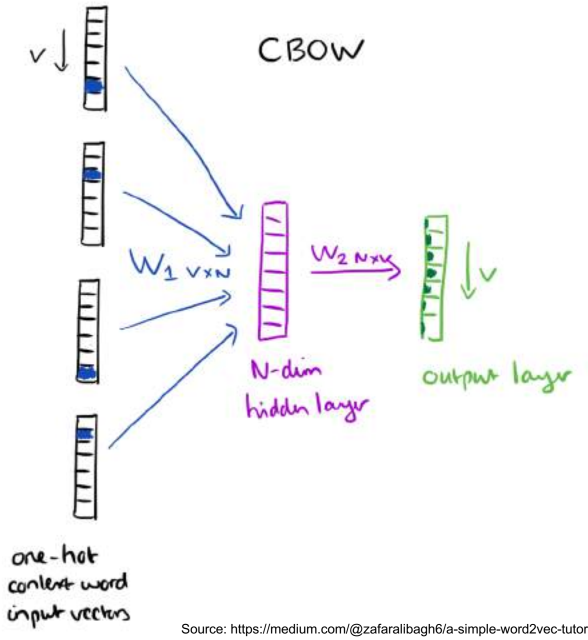
- Mikolov et al. (from Google) weren't the first to use continuous vector representations of words, but they did show how to reduce the computational complexity of learning such representations – making it practical to learn high dimensional word vectors on a large amount of data.
- Two new architectures are proposed in their study: a Continuous Bag-of-Words (CBOW) model, and a Continuous Skip-gram model.

CBOW

- Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words (a.k.a. word window =4) and precede it, and the four words that follow it:


...an efficient method for learning high quality distributed vector ...
context
focus word
context

- The context words form the input layer. Each word is encoded in one-hot form, so if the vocabulary size is V these will be V-dimensional vectors with just one of the elements set to one, and the rest all zeros. There is a single hidden layer and an output layer.



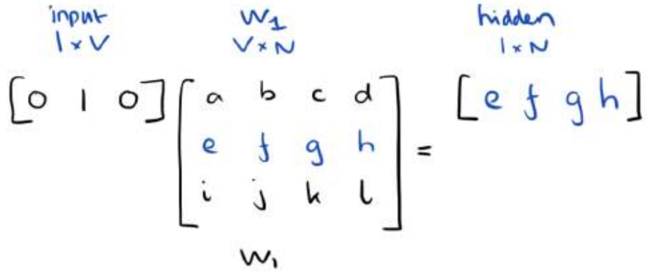
Source: https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

CBOW

- The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights.

max Prob(“learning”|“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”)

- Since our input vectors are one-hot, multiplying an input vector by the weight matrix W_1 amounts to simply selecting a row from W_1. Given C input word vectors, the activation function for the hidden layer h amounts to simply summing the corresponding ‘hot’ rows in W_1, and dividing by C to take their average.



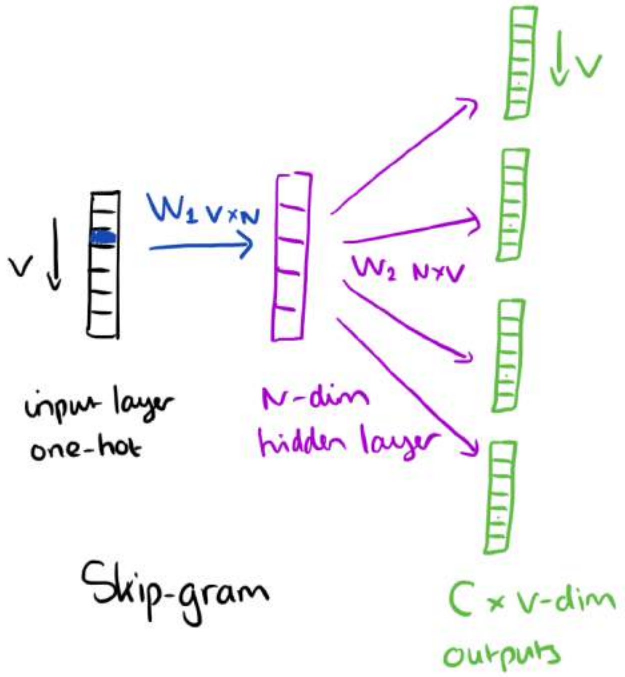
- From the hidden layer to the output layer, the second weight matrix W_2 can be used to compute a score for each word in the vocabulary, and softmax can be used to obtain the posterior distribution of words.

Source: https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

Skip-gram

- The skip-gram model is the opposite of the CBOW model. It is constructed with the focus word as the single input vector, and the target context words are now at the output layer:
- At the output layer, we now output C multinomial distributions instead of just one. The training objective is to minimize the summed prediction error across all context words in the output layer.

- Optimization
- Having to update every output word vector for every word in a training instance is very expensive.
- Mikolov et al. used Negative Sampling. It is simply the idea that we only update a sample of output words per iteration. The target output word should be kept in the sample and gets updated, and we add to this a few (non-target) words as negative samples.

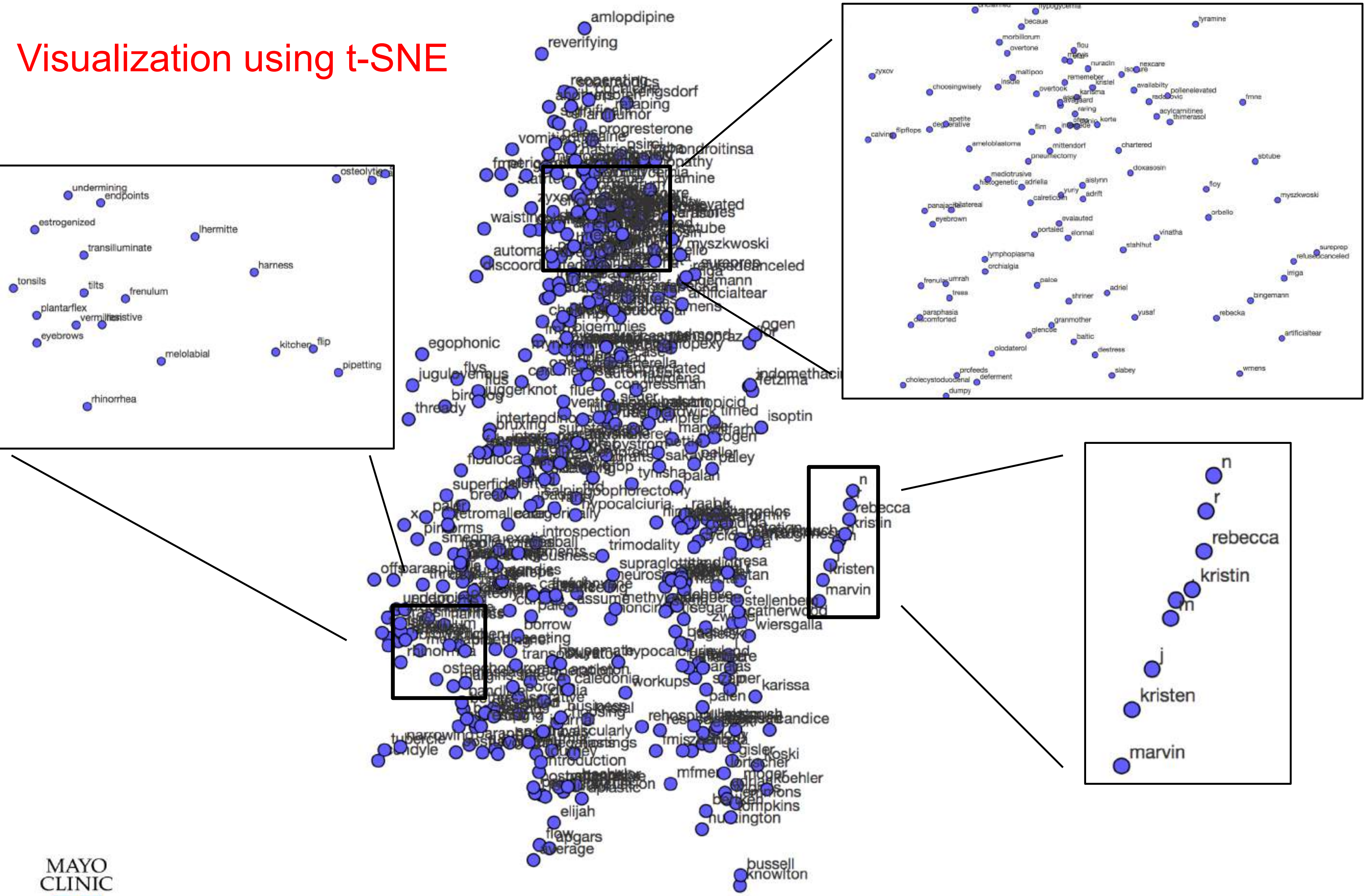


Source: https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

Word embeddings trained on Clinical notes

- Corpus
  - The collection of clinical documents consists of clinical notes for a cohort of patients receiving their primary care at Mayo Clinic, spanning a period of 15 years (1998–2013), and covering both inpatient and outpatient settings.
  - 115k patients, 3m notes, 27.8 notes/patient.
  - Vocabulary size: 103k
- Model
  - Skip-gram
  - Dimension: 20, 60
  - Minimal Word Frequency: 7
  - Word window: 5
  - Iteration: 1

Visualization using t-SNE



AGENDA

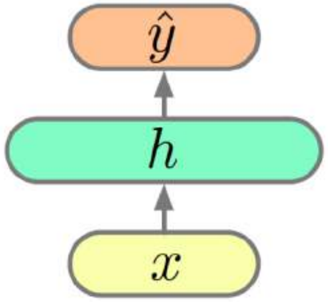
MAYO CLINIC

Recurrent neural network

- Feed forward network

$$h = g(Vx + c)$$

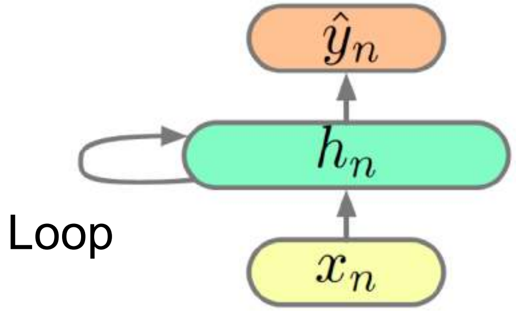
$$\hat{y} = Wh + b$$



- Recurrent neural network

$$h_n = g(V[x_n; h_{n-1}] + c)$$

$$\hat{y}_n = Wh_n + b$$



  – Humans don’t start their thinking from scratch. For example, we want to classify what kind of event is happening at every point in a movie. We want to use reasoning about previous events in the film to inform later ones.
  – RNN adds loops, allowing information to persist.

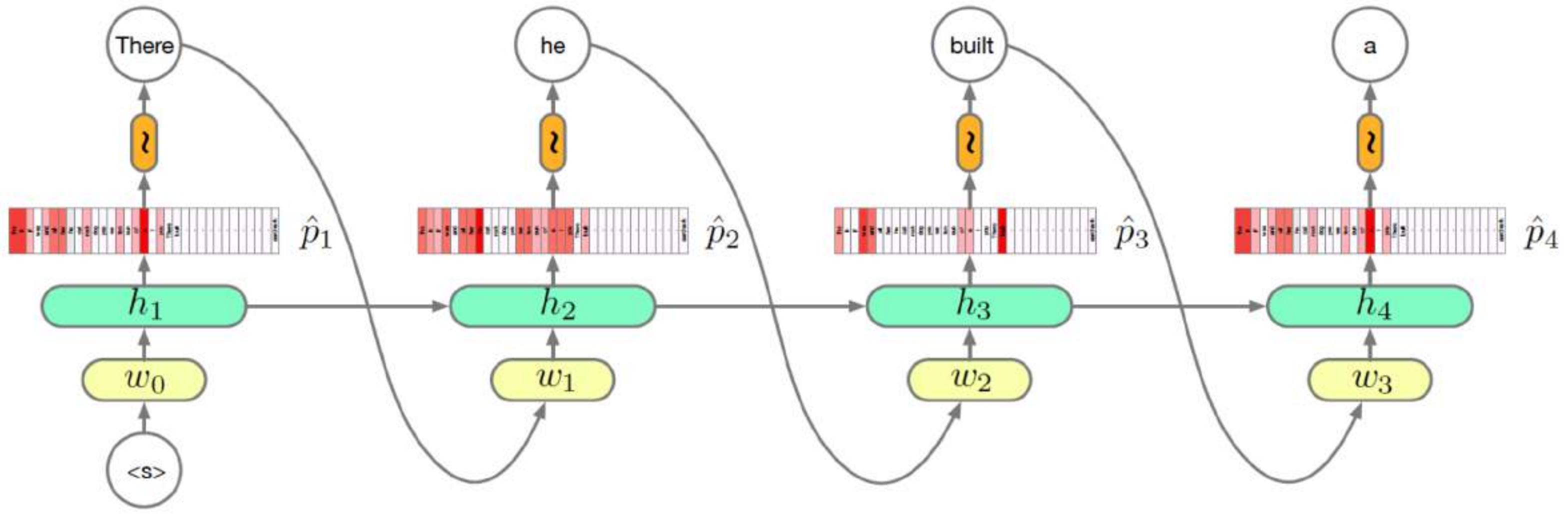
MAYO CLINIC

Source: Oxford Deep Learning for NLP class

Recurrent neural network

- A RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

$$h_n = g(V[x_n; h_{n-1}] + c)$$

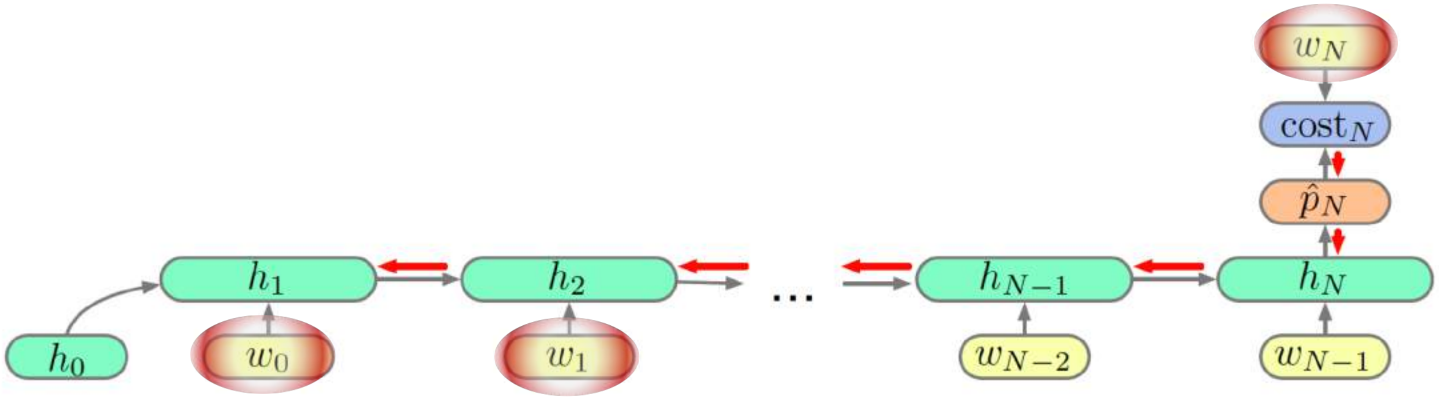


Source: Oxford Deep Learning for NLP class

RECURRENT NEURAL NETWORK

- Problem
  - RNNs can learn to use the past information. However, not all the past information is equally important to predict the next word.

$$p(sandcastle|Alice\ went\ to\ the\ beach.There\ she\ built\ a)$$



  - Unfortunately, as that gap grows, RNNs become unable to learn to connect the information

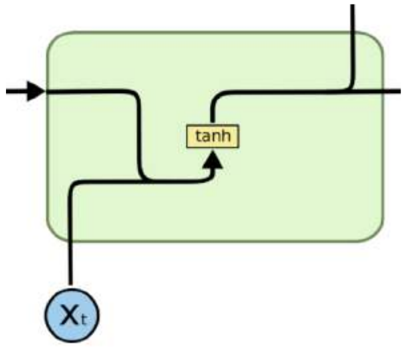
AGENDA

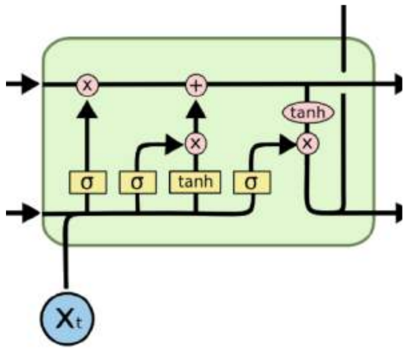
- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

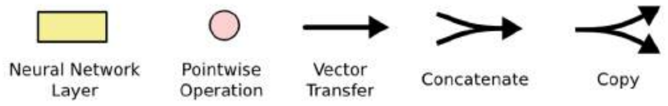
MAYO CLINIC

LSTM

- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.
- In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.
- LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



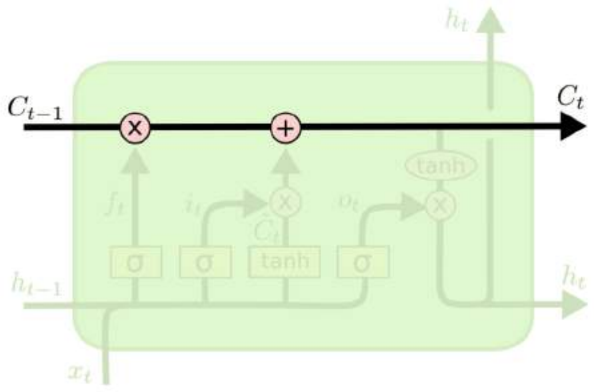




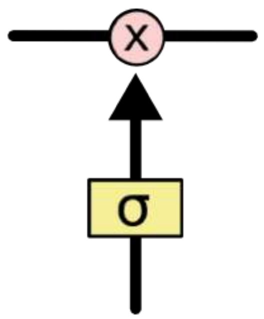
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

- Cell state
  - The key to LSTMs is the cell state.
  - Information flow along the horizontal line unchanged.



- Gate
  - In order to have the ability to remove or add information to the cell state, LSTM uses gates to optionally let information through.
  - Gates are composed out of a sigmoid neural net layer and a pointwise multiplication operation.
  - An LSTM has three of these gates, to protect and control the cell state.



MAYO CLINIC

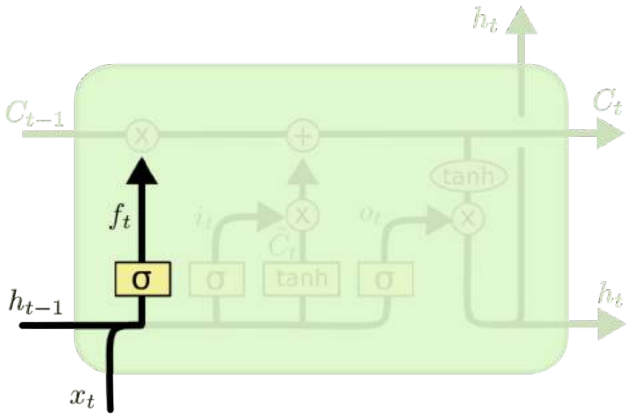
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM – STEP 1

- Forget gate layer
  - To decide what information we're going to throw away from the cell state.

$$f_t = \sigma(W_f[x_t; h_{t-1}] + c_f)$$

  - It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

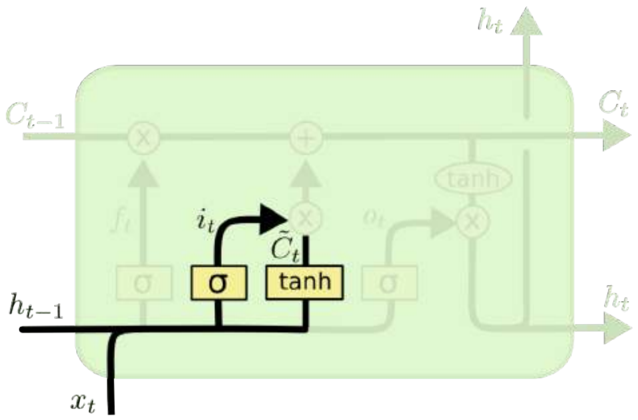
LSTM – Step 2

- Input gate layer
  - To decide what new information we're going to store in the cell state.

$$i_t = \sigma(W_i[x_t; h_{t-1}] + b_i)$$

$$\tilde{C}_t = \tanh(W_C[x_t; h_{t-1}] + b_C)$$

  - First, a sigmoid layer decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.



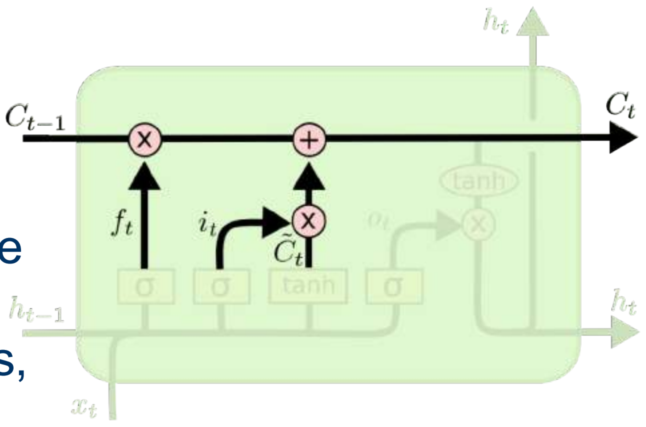
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM – STEP 3

- Update
  - To update the old cell state, C_{t-1}, into the new cell state C_t.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

  - We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ 49

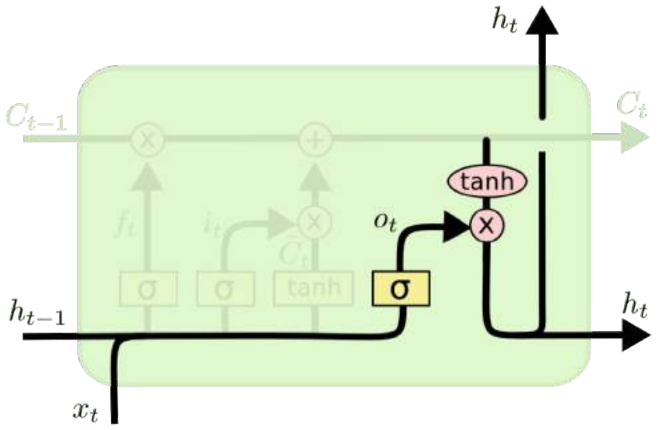
LSTM – Step 4

- Output
  - To decide what we're going to output.

$$o_t = \sigma(W_o[x_t; h_{t-1}] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

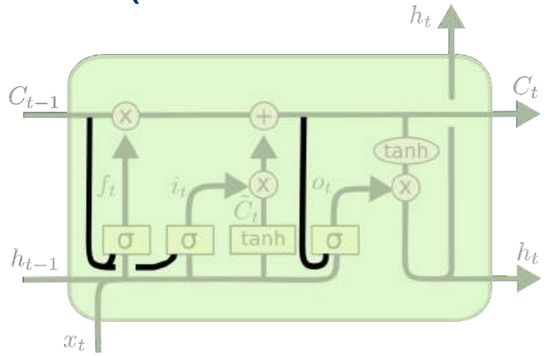
  - This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

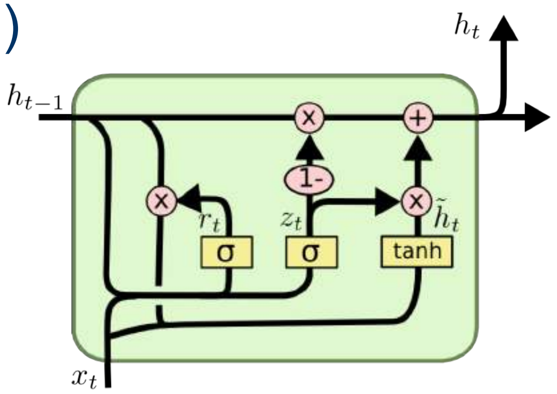
LSTM VARIANTS

- Adding ”peephole connections” to let the gate layers look at the cell state. (Gers and Schmidhuber, 2000.)



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

- Gated Recurrent Unit, or GRU, combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. (Cho, et al. 2014)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

MAYO CLINIC

APPLICATIONS: TEXT CLASSIFICATION

- Binary classification (true, false)
- Multi-class classification (politics, sports, gossip)
- Multi-label classification (#party #FRIDAY #fail)

Supervised Learning

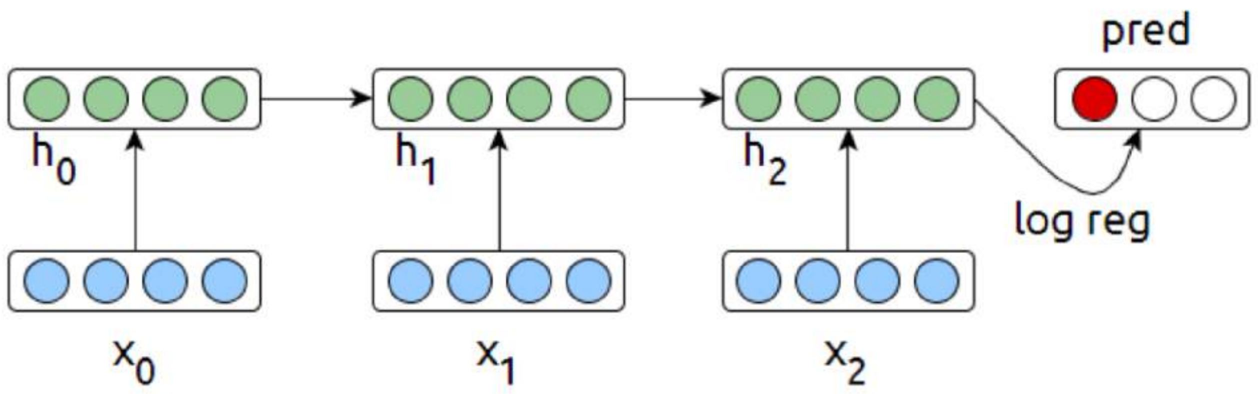
- Clustering (labels unknown)

Unsupervised Learning

MAYO CLINIC

TEXT CLASSIFICATION WITH RNN/LSTM

- So in order to classify text we can simply take a trained language model (Embeddings/RNN/LSTM), extract text representations from the final hidden state C_n, and apply softmax function.

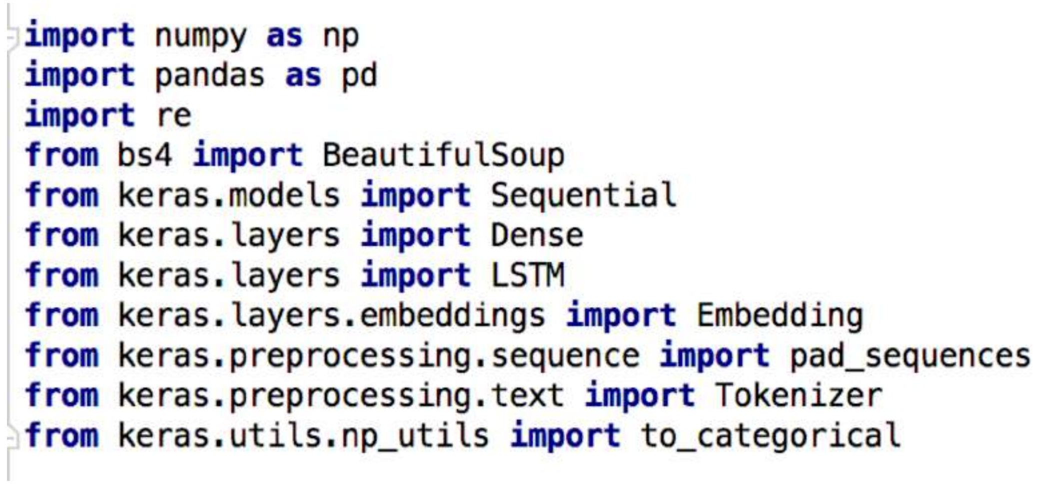


AGENDA

- Introduction
- Background
  - Natural Language Processing
  - Language Models
  - Deep Learning
- Deep Learning for NLP
  - Neural Language Models
  - Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
- RNN/LSTM in Python

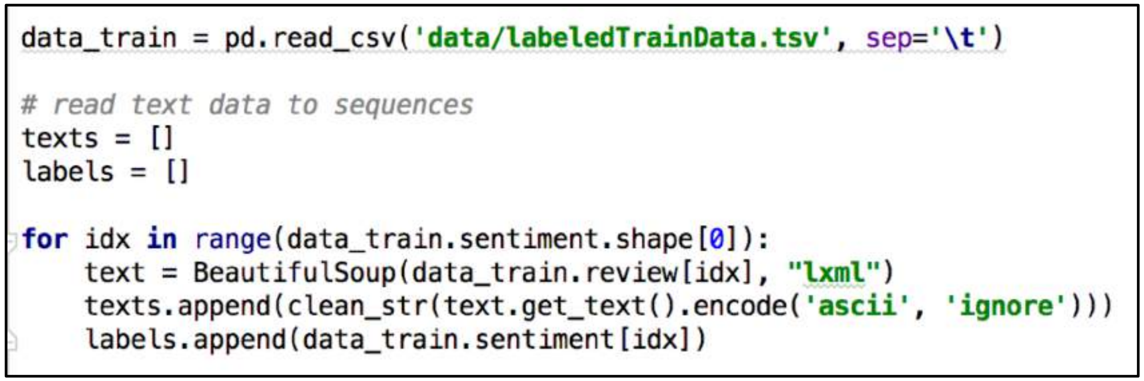
RNN/LSTM IN PRACTICE

- Environment
  - Python=2.7
  - Anaconda 2018.12 OR
  - Keras=2.2.4, TensorFlow=1.13.1

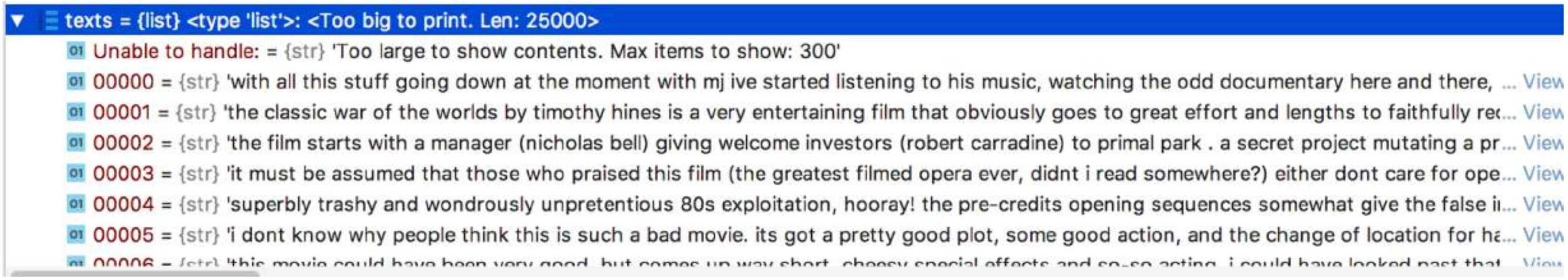


```
import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.utils.np_utils import to_categorical
```

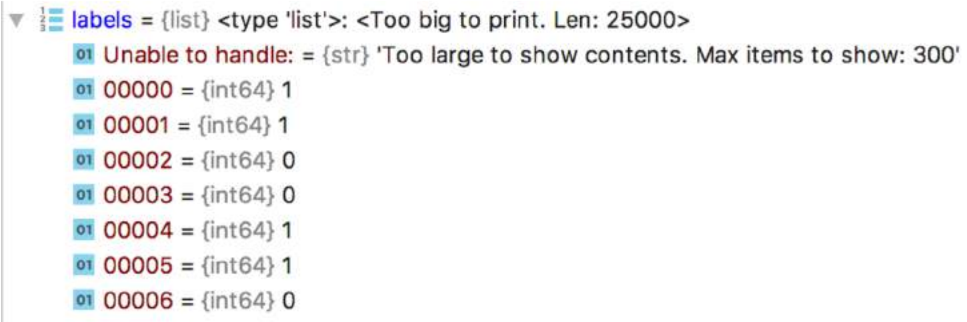
RNN/LSTM IN PYTHON



```
data_train = pd.read_csv('data/labeledTrainData.tsv', sep='\t')

# read text data to sequences
texts = []
labels = []

for idx in range(data_train.sentiment.shape[0]):
    text = BeautifulSoup(data_train.review[idx], "lxml")
    texts.append(clean_str(text.get_text().encode('ascii', 'ignore')))
    labels.append(data_train.sentiment[idx])
```

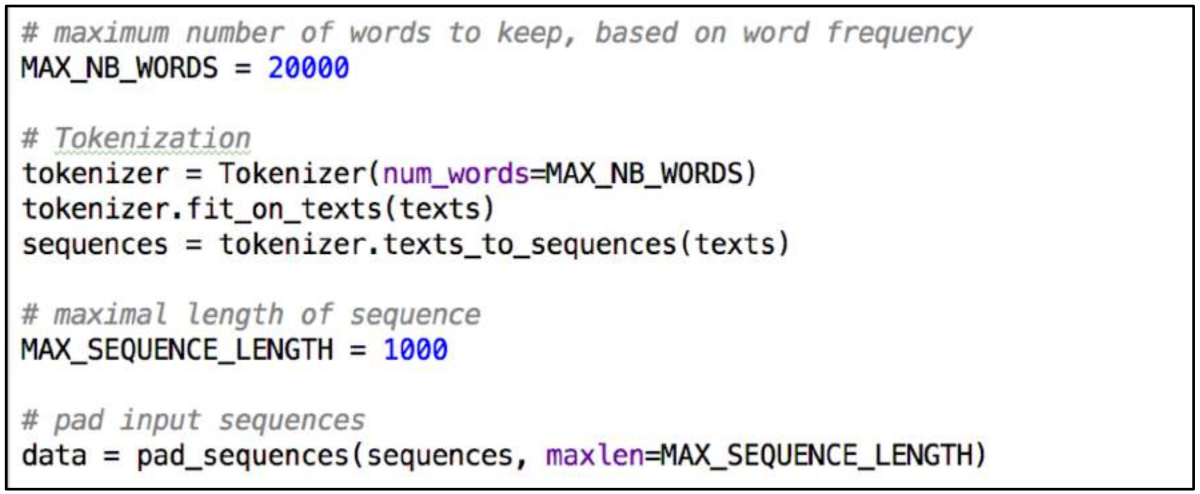


```
▼ texts = {list} <type 'list'>: <Too big to print. Len: 25000>
    Unable to handle: = {str} 'Too large to show contents. Max items to show: 300'
    00000 = {str} 'with all this stuff going down at the moment with mj ive started listening to his music, watching the odd documentary here and there, ... View
    00001 = {str} 'the classic war of the worlds by timothy hines is a very entertaining film that obviously goes to great effort and lengths to faithfully re... View
    00002 = {str} 'the film starts with a manager (nicholas bell) giving welcome investors (robert carradine) to primal park . a secret project mutating a pr... View
    00003 = {str} 'it must be assumed that those who praised this film (the greatest filmed opera ever, didnt i read somewhere?) either dont care for ope... View
    00004 = {str} 'superbly trashy and wondrously unpretentious 80s exploitation, hooray! the pre-credits opening sequences somewhat give the false i... View
    00005 = {str} 'i dont know why people think this is such a bad movie. its got a pretty good plot, some good action, and the change of location for ha... View
    00006 = {str} 'this movie could have been very good, but comes up way short. cheesy special effects and so-so acting. i could have looked past that ... View
```

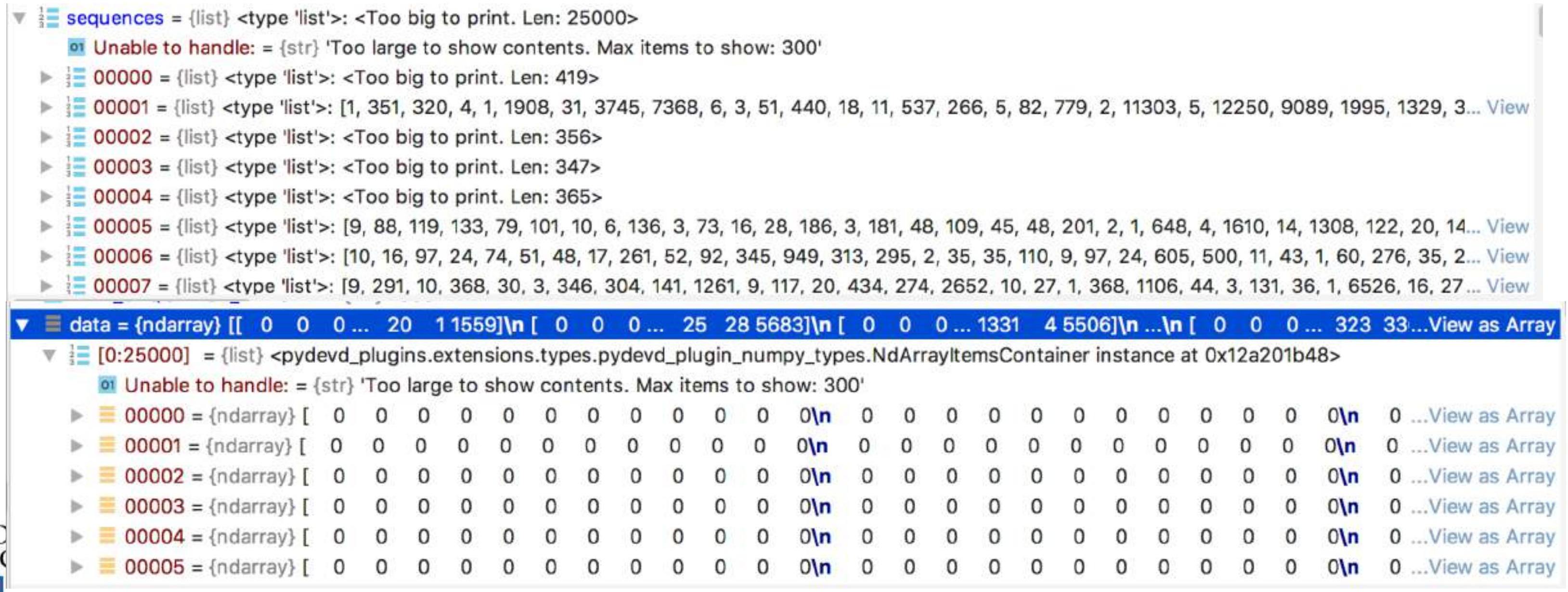


```
▼ labels = {list} <type 'list'>: <Too big to print. Len: 25000>
    Unable to handle: = {str} 'Too large to show contents. Max items to show: 300'
    00000 = {int64} 1
    00001 = {int64} 1
    00002 = {int64} 0
    00003 = {int64} 0
    00004 = {int64} 1
    00005 = {int64} 1
    00006 = {int64} 0
```

RNN/LSTM in Python



```
# maximum number of words to keep, based on word frequency
MAX_NB_WORDS = 20000

# Tokenization
tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# maximal length of sequence
MAX_SEQUENCE_LENGTH = 1000

# pad input sequences
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
```



```
▼ sequences = {list} <type 'list'>: <Too big to print. Len: 25000>
    Unable to handle: = {str} 'Too large to show contents. Max items to show: 300'
  ▶ 00000 = {list} <type 'list'>: <Too big to print. Len: 419>
  ▶ 00001 = {list} <type 'list'>: [1, 351, 320, 4, 1, 1908, 31, 3745, 7368, 6, 3, 51, 440, 18, 11, 537, 266, 5, 82, 779, 2, 11303, 5, 12250, 9089, 1995, 1329, 3... View
  ▶ 00002 = {list} <type 'list'>: <Too big to print. Len: 356>
  ▶ 00003 = {list} <type 'list'>: <Too big to print. Len: 347>
  ▶ 00004 = {list} <type 'list'>: <Too big to print. Len: 365>
  ▶ 00005 = {list} <type 'list'>: [9, 88, 119, 133, 79, 101, 10, 6, 136, 3, 73, 16, 28, 186, 3, 181, 48, 109, 45, 48, 201, 2, 1, 648, 4, 1610, 14, 1308, 122, 20, 14... View
  ▶ 00006 = {list} <type 'list'>: [10, 16, 97, 24, 74, 51, 48, 17, 261, 52, 92, 345, 949, 313, 295, 2, 35, 35, 110, 9, 97, 24, 605, 500, 11, 43, 1, 60, 276, 35, 2... View
  ▶ 00007 = {list} <type 'list'>: [9, 291, 10, 368, 30, 3, 346, 304, 141, 1261, 9, 117, 20, 434, 274, 2652, 10, 27, 1, 368, 1106, 44, 3, 131, 36, 1, 6526, 16, 27... View
▼ data = {ndarray} [[   0    0    0 ...   20    1 1559]\n [   0    0    0 ...   25   28 5683]\n [   0    0    0 ... 1331    4 5506]\n ...\n [   0    0    0 ...  323  33...View as Array
  ▼ [0:25000]  = {list} <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer instance at 0x12a201b48>
      Unable to handle: = {str} 'Too large to show contents. Max items to show: 300'
    ▶ 00000 = {ndarray} [  0  0  0  0  0  0  0  0  0  0  0  0\n  0  0  0  0  0  0  0  0  0  0  0  0\n  0 ...View as Array
    ▶ 00001 = {ndarray} [  0  0  0  0  0  0  0  0  0  0  0  0\n  0  0  0  0  0  0  0  0  0  0  0  0\n  0 ...View as Array
    ▶ 00002 = {ndarray} [  0  0  0  0  0  0  0  0  0  0  0  0\n  0  0  0  0  0  0  0  0  0  0  0  0\n  0 ...View as Array
    ▶ 00003 = {ndarray} [  0  0  0  0  0  0  0  0  0  0  0  0\n  0  0  0  0  0  0  0  0  0  0  0  0\n  0 ...View as Array
    ▶ 00004 = {ndarray} [  0  0  0  0  0  0  0  0  0  0  0  0\n  0  0  0  0  0  0  0  0  0  0  0  0\n  0 ...View as Array
    ▶ 00005 = {ndarray} [  0  0  0  0  0  0  0  0  0  0  0  0\n  0  0  0  0  0  0  0  0  0  0  0  0\n  0 ...View as Array
```

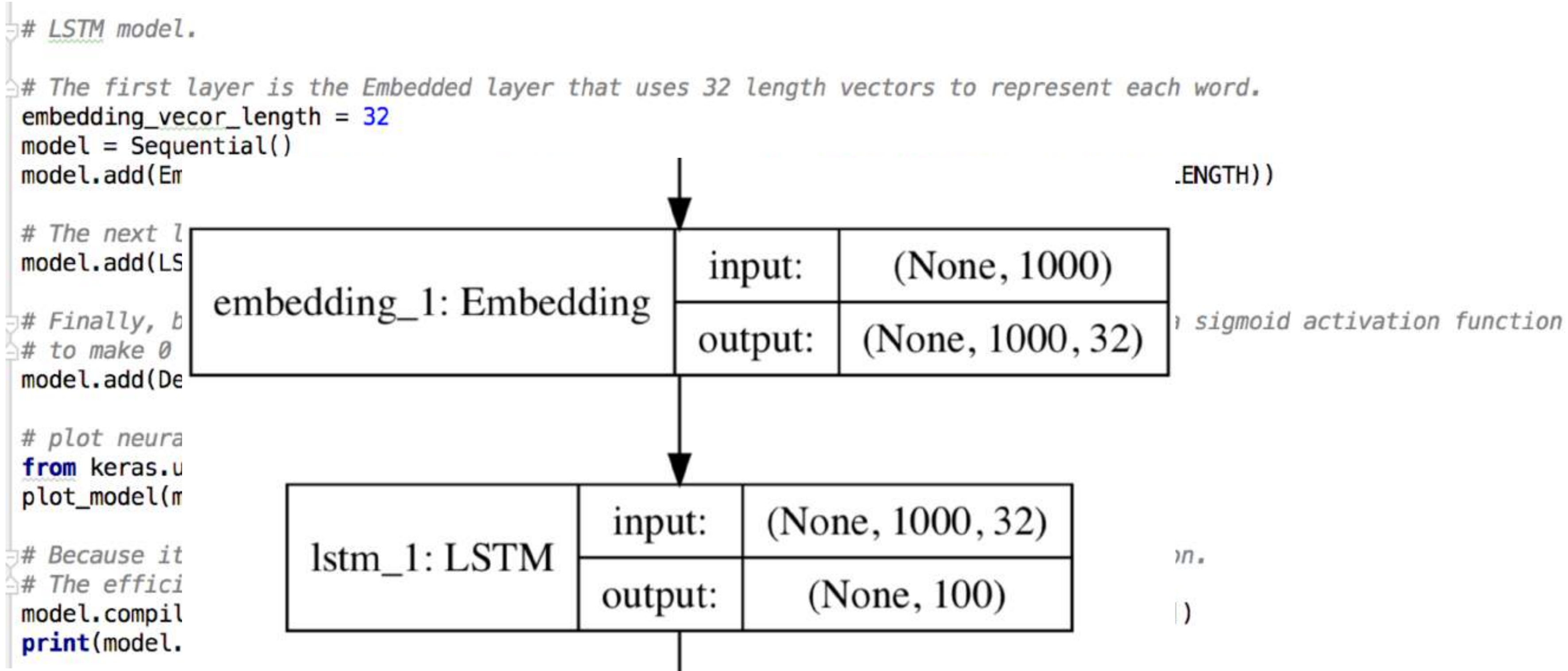
RNN/LSTM IN PYTHON

```python
# converts a class vector (integers) to binary class matrix.
labels = to_categorical(np.asarray(labels))

# shuffle data
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
```

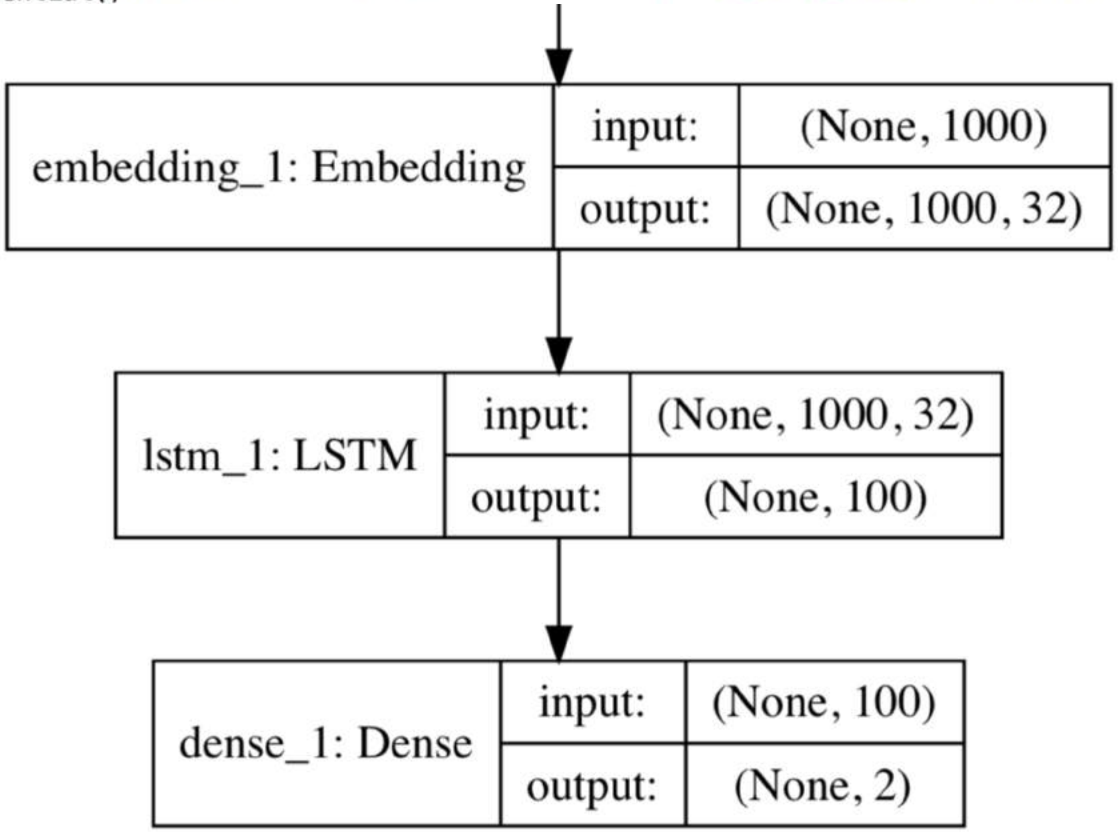
```python
# training/testing data split
VALIDATION_SPLIT = 0.1
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])
x_train = data[:data.shape[0]-nb_validation_samples]
y_train = labels[:data.shape[0]-nb_validation_samples]
x_test = data[data.shape[0]-nb_validation_samples:]
y_test = labels[data.shape[0]-nb_validation_samples:]

print('Number of positive and negative reviews in traing and validation set ')
print y_train.sum(axis=0)
print y_test.sum(axis=0)
```

MAYO CLINIC

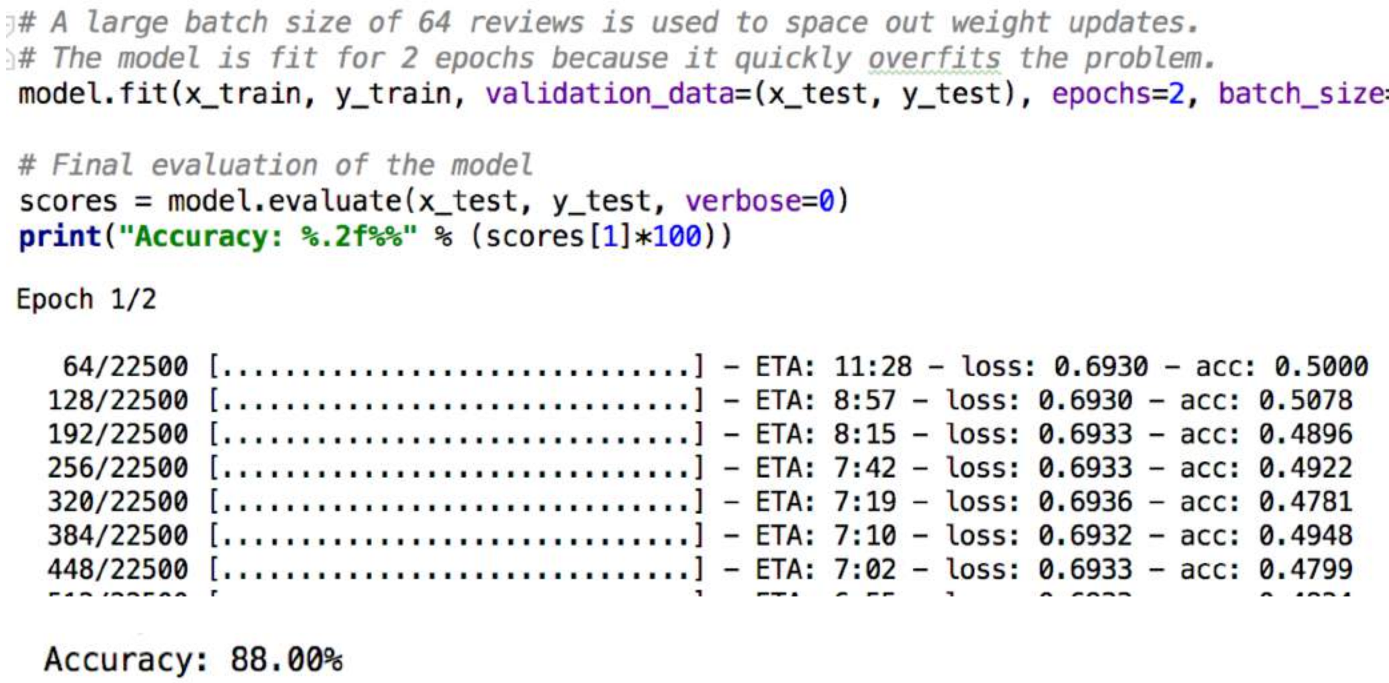
RNN/LSTM IN PYTHON



```
# LSTM model.

# The first layer is the Embedded layer that uses 32 length vectors to represent each word.
embedding_vecor_length = 32
model = Sequential()
model.add(Em                                                                        ENGTH))

# The next l
model.add(LS

# Finally, b                                                                      sigmoid activation function
# to make 0
model.add(De

# plot neura
from keras.u
plot_model(m

# Because it                                                                      n.
# The effici
model.compil                                                                      )
print(model.
```



RNN/LSTM in Python



```
# A large batch size of 64 reviews is used to space out weight updates.
# The model is fit for 2 epochs because it quickly overfits the problem.
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=2, batch_size=64)

# Final evaluation of the model
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/2

   64/22500 [..............................] - ETA: 11:28 - loss: 0.6930 - acc: 0.5000
  128/22500 [..............................] - ETA: 8:57 - loss: 0.6930 - acc: 0.5078
  192/22500 [..............................] - ETA: 8:15 - loss: 0.6933 - acc: 0.4896
  256/22500 [..............................] - ETA: 7:42 - loss: 0.6933 - acc: 0.4922
  320/22500 [..............................] - ETA: 7:19 - loss: 0.6936 - acc: 0.4781
  384/22500 [..............................] - ETA: 7:10 - loss: 0.6932 - acc: 0.4948
  448/22500 [..............................] - ETA: 7:02 - loss: 0.6933 - acc: 0.4799

  Accuracy: 88.00%
```

RNN/LSTM in Python

- Demo
- Hyper-parameters fine-tuning
  - Number of layers
  - Number of hidden units per layer
  - Activation function
  - Optimizer
  - Learning rate
  - Batch size
  - Number of epochs

- Thank you!

- Contact
  - Email: wang.yanshan@mayo.edu
  - LinkedIn, Twitter (yanshan_wang)

MAYO CLINIC