Natural Language Processing

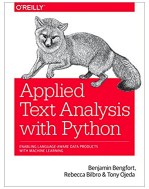
Bruno Gonçalves

www.data4sci.com/newsletter

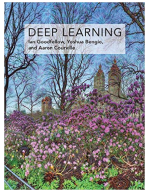
https://github.com/DataForScience/NLP_LL

References

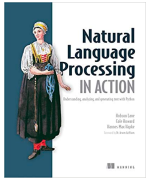
https://github.com/DataForScience/NLP_LL



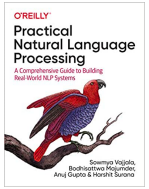
https://amzn.to/3iMqanY



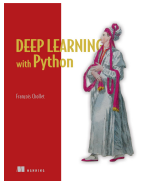
https://amzn.to/2BGr0RL



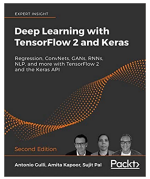
https://amzn.to/3sXAZbm



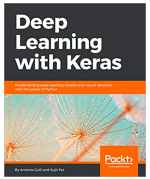
https://amzn.to/3a2fhui



https://amzn.to/30fTJqB



https://amzn.to/30fTMCN



https://amzn.to/3qR3rKh

https://amzn.to/2AavBuT

Lesson 1: Text Representation

1.1 - One-hot encoding
1.2 - Bag of Words
1.3 - Stop words
1.4 - TF/IDF
1.5 - N-grams
1.6 - Word Embeddings

Lesson 1: Text Representation

1.1 - One-hot encoding

1.2 - Bag of Words

1.3 - Stop words

1.4 - TF/IDF

1.5 - N-grams

1.6 - Word Embeddings

Tokenization

- The first step in handling text in an automated way is to convert it to individual tokens, to tokenize the text.
- Individual tokens can correspond to individual words, numbers, or punctuation that we want our code to process in our specific application.
- The way in which we define what counts as a token will have a tremendous impact on what our application will be able to do:
  - Will it include numbers or just words?
  - Is punctuation kept as individual tokens? as part of the preceding word? removed completely?
  - Are urls, #tags, @usernames, etc included or excluded?

Tokenization

- The total number of tokens we'll need to keep tract of, the size of our vocabulary, will depend on our choice and have a direct impact on the computational cost of our analysis.
- In the following, whenever we say 'word' we actually mean 'token'

Tokenization

- The sentence:

  "The Earth is estimated to be 4.54 billion years old, plus or minus about 50 million years."

- Can be tokenized in various ways:
  - ["The", "Earth", "is", "estimated", "to", "be", "4.54", "billion", "years", "old", ",", "plus", "or", "minus", "about", "50", "million", "years", "."]
  - ["The", "Earth", "is", "estimated", "to", "be", "4.54", "billion", "years", "old", "plus", "or", "minus", "about", "50", "million", "years"]
  - ["The", "Earth", "is", "estimated", "to", "be", "4.54", "billion", "years", "old,", "plus", "or", "minus", "about", "50", "million", "years."]
  - ["The", "Earth", "is", "estimated", "to", "be", "billion", "years", "old,", "plus", "or", "minus", "about", "million", "years."]
- Or any combination thereof.

Tokenization

- nltk has extensive support for tokenization through the nltk.tokenize submodule
- Two convenience methods are particularly interesting:
  - word_tokenize() - tokenize the text splitting at words and punctuation:
  - sent_tokenize() - tokenize the text into individual sentences.
- In the background, these functions are just using one of nltk Tokenizer objects, which provide a wealth of options specialized for various cases
  - 'LineTokenizer', 'NLTKWordTokenizer', 'PunktSentenceTokenizer', 'RegexpTokenizer', 'SpaceTokenizer', 'SyllableTokenizer','TweetTokenizer', 'WhitespaceTokenizer', etc

Words and Numbers

- How can computers represent, analyze and understand a piece of text?
- Computers are really good at crunching numbers but not so much when it comes to words.
- Perhaps we can substitute words with numbers?
  - Unfortunately, computers assume that numbers are sequential.
- Vectors work much better.

1	a
2	about
3	above
4	after
5	again
6	against
7	all
8	am
9	an
10	and
11	any
12	are
13	aren't
14	as
…	…

One-Hot Encoding

- The most basic approach to representing text in a way that can be easily manipulated numerically is known as One-Hot Encoding:
  - Each word corresponds to a different dimension in a high-dimensional space
  - All elements of the vector are zero, except the one corresponding to the word

$$v_{fleece} = (0, 0, 0, 0, 1, 0, 0, \cdots)^T$$
$$v_{everywhere} = (0, 0, 0, 1, 0, 0, 0, \cdots)^T$$

- One-hot encoded vectors are extremely sparse and contain no semantic information

One-Hot Encoding

- So the text for “Mary had a little lamb”:

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

- Could be represented using this one-hot encoded matrix (we omit the 0 values for clarity).

	a	and	as	everywhere	fleece	go	had	lamb	little	mary	snow	sure	that	the	to	was	went	white	whose
0										1									
1							1												
2	1																		
3									1										
4								1											
5									1										
6								1											
7									1										
8								1											
9										1									
10							1												
11	1																		
12									1										
13								1											
14																			1
15						1													
16																	1		
17																		1	
18			1																
19											1								
20		1																	
21					1														
22														1					
23										1									
24																		1	
25										1									
26																		1	
27										1									
28																		1	
29					1														
30														1					
31										1									
32																		1	
33															1				
34								1											
35																	1		
36												1							
37																1			
38						1													

One-Hot Encoding

- What about full texts instead of single words?
- The vector representation of a text is simply the vector sum of all the words it contains:

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

0	had	10	lamb
1	went	11	as
2	and	12	that
3	a	13	sure
4	was	14	whose
5	to	15	go
6	snow	16	the
7	everywhere	17	little
8	mary	18	white
9	fleece		

- Or, in other words, this text would be represented by the vector:

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, 6, 1, 5, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

One-Hot Encoding

- What about full texts instead of single words?
- The vector representation of a text is simply the vector sum of all the words it contains:

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

0	had	10	lamb
1	went	11	as
2	and	12	that
3	a	13	sure
4	was	14	whose
5	to	15	go
6	snow	16	the
7	everywhere	17	little
8	mary	18	white
9	fleece		

- Or, in other words, this text would be represented by the vector:

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, 6, 1, 5, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

- Which is just the column sum of. the one-hot encoded matrix.

One-Hot Encoding

- pandas provides a pd.get_dummies() method to generate one hot encoded representations of Series
- sklearn has a OneHotEncoder transformer as part of the sklearn.preprocessing module
- Both version expect an array (or Series) of integer or string elements representing the possible values of a Categorical variable and return an array (or DataFrame) with one of the categorical values per column and a 0 or 1 signaling which value corresponds to which row.
- nltk represents one-hot encoded vectors as dictionary where the keys (tokens) map to True values as a way of taking full advantage of sparseness

Lesson 1: Text Representation

1.1 - One-hot encoding

1.2 - Bag of Words

1.3 - Stop words

1.4 - TF/IDF

1.5 - N-grams

1.6 - Word Embeddings

Bag-of-Words

- In practice, it's much more convenient to use a dictionary as a compact and easily extendable version of a vector
- This is known as a bag-of-words, and word order is completely discarded.

Bag-of-Words

- In practice, it's much more convenient to use a dictionary as a compact and easily extendable version of a vector
- This is known as a bag-of-words, and word order is completely discarded.
- For our little nursery rhyme, this could simply be:
- Similar representations could be generated for different documents, allowing us to compare or cluster them easily.

	Count
a	2
and	1
as	1
everywhere	2
fleece	1
go	1
had	2
lamb	5
little	4
mary	6
snow	1
sure	1
that	2
the	1
to	1
was	2
went	4
white	1
whose	1

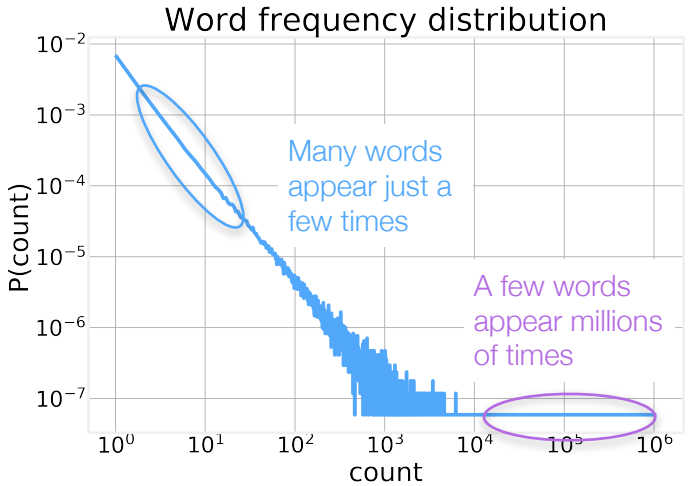
Lesson 1: Text Representation

1.1 - One-hot encoding
1.2 - Bag of Words
1.3 - Stop words
1.4 - TF/IDF
1.5 - N-grams
1.6 - Word Embeddings

Stopwords

- Some words are much more common than others.
- While most words are very rare.
- The most common words in a corpus of 17M words:
- These are known as "stopwords", words that carry little meaning and can be discarded.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644



Stopwords

- After removing the most common words we go from 17M words to just 9M, without significantly losing any information!
- In practice, stopwords aren't simply the most common words but rather curated lists of common and non-informative words.
- Computational linguists have published lists of stop words that can easily be found online, and that were curated for different languages and purposes.
- Stopwords in 40 languages: https://www.ranks.nl/stopwords

Stopwords

- Different applications might use different sets of stop words of none at all.
- The goal of removing them from your text is to significantly reduce the number of words you must process while losing as little information as possible.
- Naturally, these are language dependent.

Original	Filtered
Mary	Mary
had	
a	
little	little
lamb	lamb
little	little
lamb	lamb
little	little
lamb	lamb
Mary	Mary
had	
a	
little	little
lamb	lamb
whose	whose
fleece	fleece

Stopwords

- NLTK supports 23 languages out of the box. These are typically stored as plain text files under '~/nltk_data/corpora/stopwords/'
- You can add more by simply adding a text file in the proper directory with one word per line.
- Stopwords can be loaded to NTLK by using the file name

```
'dutch'
'german'
'slovene'
'hungarian'
'romanian'
'kazakh'
'turkish'
'russian'
'italian'
'english'
'greek'
'tajik'
'norwegian'
'portuguese'
'finnish'
'danish'
'french'
'swedish'
'azerbaijani'
'spanish'
'indonesian'
'arabic'
'nepali'
```

Lesson 1: Text Representation

1.1 - One-hot encoding
1.2 - Bag of Words
1.3 - Stop words
1.4 - TF/IDF
1.5 - N-grams
1.6 - Word Embeddings

Term Frequency - Inverse Document Frequency

- We already saw that some words are much more common than others
- The number of times that a word appears in a document is known as the "Term Frequency" (TF)
- After the removal of stopwords, the term frequency is a good indicator of what words are most important. A book on Python programming will likely have words like "code", "script", "print", "error", etc much more frequently than a book on football.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644

Term Frequency - Inverse Document Frequency

- TF gives us an idea of how popular a specific term is within a document, but how can we compare across documents within a corpus?
- The Inverse Document Frequency (IDF) tells us how unusual it is for a document to include that word. The idea is that words that appear in more documents are less meaningful

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644

TF-IDF

- Mathematically there are several possible definitions for both TF and IDF

	TF	IDF
	$N_{t,d}$	$\log\left(\frac{N}{N_t}\right)$
	$\frac{N_{t,d}}{\sum_{t'} Nt', d}$	$\log\left(1+\frac{N}{N_t}\right)$
	$1+\log\left(N_{t,d}\right)$	

Number of times term t occurs in document d

Number of documents

Number of documents in which term t appears

- TF-IDF is the product of these two quantities and is useful for finding terms that are important for the specific document (high TF) and uncommon in the corpus as a whole (large IDF/small DF).

TF-IDF

- In particular, a term that occurs in every document is meaningless when it comes to distinguishing between documents.
- Stopwords, are naturally weighed down due to appearing in all documents and can even be automatically given zero weight with the right definition of TF

Lesson 1: Text Representation

1.1 - One-hot encoding
1.2 - Bag of Words
1.3 - Stop words
1.4 - TF/IDF
1.5 - N-grams
1.6 - Word Embeddings

N-grams

- N-grams are co-occurring sequences of N items from a sequence of words of characters
- nltk provides the nltk.util.ngrams utility function to easily generate N-Grams of specific lengths
- N-grams are important to account for modifiers, Named Entity Recognition, etc.
- But how can we know if a N-Gram is significant?

Collocations

- A closely related concept is that of Collocation - N-Grams that occur more commonly than expected by chance.
- The nltk.collocations submodule provides objects to identify and compute the most significant Bigram, Trigram and Quadgrams:
  - Bigram/Trigram/QuadgramCollocationFinder - support for different ways of finding 2, 3, and 4-grams.
  - Bigram/Trigram/QuadgramAssocMeasures - selection of metrics to quantify the relative importance of each 2, 3, and 4-grams. In particular:
    - chi_sq/jaccard/likelihood_ratio/mi_like/pmi/poisson_stirling/raw_freq/student_t
  - Significant collocations can prove useful for entity extraction, topic detection, etc.

Lesson 1: Text Representation

1.1 - One-hot encoding
1.2 - Bag of Words
1.3 - Stop words
1.4 - TF/IDF
1.5 - N-grams
1.6 - Word Embeddings

Word Embeddings

- Word-Embeddings are simply vector representations of words.
- Typical vector representations are;
  - one-hot encoding
  - bag of words
  - TF/IDF
  - etc
- None of these representations include semantic (meaning) information encoded in the vector.

Word Embeddings

- Two common variants:
  - word2vec
    - Developed by Google in 2013, it relies on the distributional hypothesis in linguistics: ‘words that appear in similar contexts have similar meanings’
    - 100+ languages trained using wikipedia: https://sites.google.com/site/rmyeid/projects/polyglot
  - GloVE
    - Developed by Stanford in 2014, it explicitly uses a co-occurrence matrix to determine the embedding vector
    - Pre-trained vectors: https://github.com/stanfordnlp/GloVe

Word Embeddings

- In either case, vector similarity correlates strongly with meaning similarity and is a powerful way to disambiguate the meaning of sentences and documents

Code - Text Representation

https://github.com/DataForScience/NLP_LL

Lesson 2: Text Cleaning

2.1 - Stemming

2.2 - Lemmatization

2.3 - Regular Expressions

Lesson 2: Text Cleaning

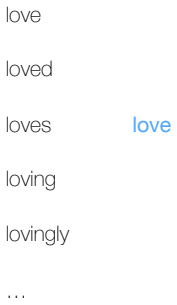
2.1 - Stemming

2.2 - Lemmatization

2.3 - Regular Expressions

Word Variants

- So far we have seen multiple techniques to represent words and to remove extraneous words from our vocabulary.
- But what about word variants? Verb conjugations, plurals, nouns and adverbs?



- In some cases they should just be represented by the same stem or root.

Stemming and Lemmatization

- In practical applications, Vocabularies can become extremely large (English is estimated to have over 1 million unique words).
- Several techniques have been developed to help reduce the vocabulary size with minimal loss of information. In particular:
  - Stemming - Use heuristics to identify the root (or stem) of the word.
    The stem doesn't need to be a "real" word as long as the mapping is consistent.
  - Lemmatization - Identify the "dictionary form" (lemma) of the word. This approach requires identifying the Part-of-Speech being used and using hand curated tables to find the correct lemma.
  - Stopwords - Remove the most common words that don't contain any semantic information (the, and, a, etc)

Stemming

- Stemming is a series of techniques to automatically identify the stem or root of a word
- Naturally, the rules that must be applied depend on the grammar of the specific language being used
- For English, the most common algorithm is called Porter stemmer

Stemming

- nltk contains several different stemmer algorithms, with varying support for different languages
  - Cistem - German
  - ISRIStemmer - Arabic
  - LancasterStemmer - English
  - PorterStemmer - English (the original one)
  - RSLPStemmer - Portuguese
  - RegexpStemmer - English (using Regular Expressions)
  - SnowballStemmer - Arabic, Danish, Dutch, English, Finnish, French,

	LancasterStemmer	PorterStemmer	RegexpStemmer	SnowballStemmer
playing	play	play	play	play
loved	lov	love	loved	love
ran	ran	ran	ran	ran
river	riv	river	river	river
friendships	friend	friendship	friendship	friendship
misunderstanding	misunderstand	misunderstand	misunderstand	misunderstand
trouble	troubl	troubl	troubl	troubl
troubling	troubl	troubl	troubl	troubl

Lesson 2: Text Cleaning

2.1 - Stemming

2.2 - Lemmatization

2.3 - Regular Expressions

Lemmatization

- nltk implements the WordNetLematizer algorithm that uses the WordNet database of concepts.
- WordNetLematizer algorithm is guaranteed to return a “real” word but the results depend on correct Part-Of-Speech identification. The result for a Noun will be different than the result for a Verb, Adverb, etc.

Lemmatization

- Lemmatization tends to be computationally more expensive than Stemming
- Depending on your specific application, you might prefer Stemming or Lemmatization

	LancasterStemmer	PorterStemmer	RegexpStemmer	SnowballStemmer	WordNetLemmatizer Noun	WordNetLemmatizer Verb
playing	play	play	play	play	playing	play
loved	lov	love	loved	love	loved	love
ran	ran	ran	ran	ran	ran	run
river	riv	river	river	river	river	river
friendships	friend	friendship	friendship	friendship	friendship	friendships
misunderstanding	misunderstand	misunderstand	misunderstand	misunderstand	misunderstanding	misunderstand
trouble	troubl	troubl	troubl	troubl	trouble	trouble
troubling	troubl	troubl	troubl	troubl	troubling	trouble

Lesson 2: Text Cleaning

2.1 - Stemming

2.2 - Lemmatization

2.3 - Regular Expressions

Regular Expressions

- Regular Expressions (called REs, or regexes, or regex patterns) are a tiny, highly specialized programming language designed to specify patterns and matches in text.
- Python provides this functionality in the re module
- regexes are implemented in C, so they're extremely fast, but can quickly get complex.
- regexes are specified using special strings that specify the rules that determine what matches look like.
- Most characters simply represent themselves, but there are several meta characters carry special meaning within the regex specification.

Metacharacters

Metcharacter	Example	Explanation
[]	[abc]	Square brackets specify character classes. This example matches any of the a, b or c characters
-	[a-z]	Within a character class, the dash represents a range. This example will match any lower case character from a to z.
^	[^z]	At the beginning of a character class, it inverts the selection. Anywhere else within a character class it stands for itself. This example will match any character other than z.
^	^5	Matches the beginning of the string only. This example checks if the string starts with the number 5
$	$5	Matches the end of the string only. This example checks if the string ends with the number 5
\	\[	The backslash is the escape character than you use to literally refer to other metacharacters. This example specifies an actual "[" by itself and not as the start of a character class.
*	a*	Matches zero of more of the preceding character of character class. This example will match no 'a's, a, aa, aaa, etc…
?	b?	Matches an optional (zero or one) character or character class.
+	a+	Represents one or more matches for the preceding character or character class. This example will match a, aa, aaa, etc…
.	.	Represents every character *except* a new line

Character classes

- Some useful character classes are predefined with a nice short hand notation:

Class	Explanation
\d	Matches any decimal digit; this is equivalent to the class [0-9].
\D	Matches any non-digit character; this is equivalent to the class [^0-9].
\s	Matches any whitespace character; this is equivalent to the class [\t\n\r\f\v].
\S	Matches any non-whitespace character; this is equivalent to the class [^ \t\n\r\f\v].
\w	Matches any alphanumeric character; this is equivalent to the class [a-zA-Z0-9_].
\W	Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].
\b	Matches word boundaries, the start/end of words or strings.

- This makes complex rules simpler to write.

Basic Usage

- regex support in Python lives in the re module:

```
import re
```

- regex are typically specified using raw strings and have to be ‘compiled’ before being usable.
- re.compile(regex_str) - return a compiled re.Pattern object representing the regex described in the regex_str raw string

Basic Usage

- **Pattern objects have several methods that make it easy to use regexes:**
  - .match() - Searches for a match at the beginning of the target string
  - .search() - Searches for the first match anywhere within the target string
  - .findall() - Return all matching substrings as a list
  - .finditer() - Return all match objects as an iterator

Basic Usage

- Successful matches are returned as re.Match objects
- Matches contain all the information you might. need to further process it:
- .group() - Returns the matched string
- .span() - Returns a tuple containing the start and end positions in the target string
- .start()/.end() - Returns the starting/ending position of the match

Grouping

- You can specify groups within your regex using parenthesis (…)
- The contents of the parenthesis will behave normally, but groups make it easier to perform more complex matches.
- Groups are numbered, starting at 0. Group 0 is always the full matching string, group 1 will be the contents of the first parenthesis, etc.
- Groups can also be nested
- Within your regex you can refer to the match of a group (after it's been found) by its number \1 is the contents of group 1, \2 the contents of group 2, etc.
- For example, the regex r'\b(\w+)\s+\1\b' will match repeated full words as the \1 will match only if exactly the same match is found in that position.

Modifying strings

- Sometimes we need to not only to check if a pattern is found within a string but rather to modify the string whenever the pattern is found.
- re.Pattern objects have several methods to allow us to do this:
  - .split() - Split the string whenever the pattern is matched
  - .sub() - Replace matched patterns with a specific string
  - .subn() - Similar to sub() but returns the number of replacements.
- The first argument of these functions is the replacement string and the second the string to process.

Code - Text Cleaning

https://github.com/DataForScience/NLP_LL

Lesson 3: Named Entity Recognition

3.1 - Part Of Speech Tagging

3.2 - Chunking

3.3 - Chinking

3.4 - Named Entity Recognition

Lesson 3: Named Entity Recognition

3.1 - Part Of Speech Tagging

3.2 - Chunking

3.3 - Chinking

3.4 - Named Entity Recognition

Parts Of Speech

- As you might recall from elementary school, sentences have well defined grammatical structures. For example, a simple sentence in English might be:

  "The dog kicked the ball"

- which is made up of a Subject, Verb and Object with each component playing a different role in the sentence.
- These are known as 'Parts Of Speech (POS)' and we are often interested in automatically tagging words with their correct POS that they represent within a specific sentence.
- POS tagging is a common pre-processing step in many algorithms as it helps us disambiguate the meaning of words, clarify the overall subject of a paragraph or document, determine sentiment, etc.
- Unfortunately, POS tagging is a hard problem so most methods are statistical in nature

Part-Of-Speech Tagging

- nltk provides a simple utility function to (probabilistically) determine the correct POS tags for a snippet of text through the pos_tag() function.
- By default, pos_tag() uses the Penn Treebank tagset for English (and the Russian National Corpus tagset for Russian).

Part-Of-Speech Tagging

- The Penn Treebank tagset is able to detect several dozen parts of speech such:

Tag	Meaning	Examples
DT	Determiner	all an another any both del each either every half la many much nary neither no some such that the them these this those
VB	Verb	ask assemble assess assign assume atone attention avoid bake balkanize bank begin behold believe bend benefit bevel beware bless boil bomb boost brace break bring broil brush build
NN	Noun	common-carrier cabbage knuckle-duster Casino afghan shed thermostat investment slide humour falloff slick wind hyena override subhumanity machinist
RB	Adverb	occasionally unabatingly maddeningly adventurously professedly stirringly prominently technologically magisterially predominately swiftly fiscally pitilessly
JJ	Adjective	third ill-mannered pre-war regrettable oiled calamitous first separable ectoplasmic battery-powered participatory fourth still-to-be-named multilingual multi-disciplinary

Part-Of-Speech Tagging

- You can find the complete documentation for all the tags by calling: nltk.help.upenn_tagset()
- pos_tag() expects previously tokenized inputs, so you'll often see this construct: pos_tag(word_tokenize(text))
- For example, calling pos_tag() for sentence:

  "They refuse to permit us to obtain the refuse permit"

- returns:

('They', 'PRP'),	pronoun, personal
('refuse', 'VBP'),	verb, present tense, not 3rd person singular
('to', 'TO'),	"to" as preposition or infinitive marker
('permit', 'VB'),	verb, base form
('us', 'PRP'),	pronoun, personal
('to', 'TO'),	"to" as preposition or infinitive marker
('obtain', 'VB'),	verb, base form
('the', 'DT'),	determiner
('refuse', 'NN'),	noun, common, singular or mass
('permit', 'NN'),	noun, common, singular or mass

Part-Of-Speech Tagging

('They', 'PRP'),	pronoun, personal
('refuse', 'VBP'),	verb, present tense, not 3rd person singular
('to', 'TO'),	"to" as preposition or infinitive marker
('permit', 'VB'),	verb, base form
('us', 'PRP'),	pronoun, personal
('to', 'TO'),	"to" as preposition or infinitive marker
('obtain', 'VB'),	verb, base form
('the', 'DT'),	determiner
('refuse', 'NN'),	noun, common, singular or mass
('permit', 'NN'),	noun, common, singular or mass

- Correctly identifying that two homonyms of 'refuse' and 'permit' are being used (a verb and a noun)

Part-Of-Speech Tagging

- nltk supports several other pos taggers, such as the, 'BrillTagger', 'ContextTagger', 'HiddenMarkovModelTagger', 'PerceptronTagger', etc
- 'HiddenMarkovModelTagger' and 'BrillTagger' can be trained using the corresponding Trainer object ('HiddenMarkovModelTrainer', 'BrillTaggerTrainer', respectively).
- To train a tagger you need a pre-tagger corpus. Fortunately, nltk also provides tagged versions of several corpora that we can easily use.
- Some other taggers, such 'RegExTagger' rely instead on a simple set of Regular Expressions to match patterns to tags

Lesson 3: Named Entity Recognition

3.1 - Part Of Speech Tagging

3.2 - Chunking

3.3 - Chinking

3.4 - Named Entity Recognition

Chunking

- Chunking is the process through which we identify ‘phrases’ in our text.
- phrases are groups of one or more words that are used as a single unit. For example:
  - a little lamb
  - Planet Earth
  - etc
- Chunking relies heavily on POS tagging and can only be performed on properly tagged datasets

Chunking

- Unfortunately, there's no objective way of defining Chunks. They must be defined 'by hand' using a "Tag Patterns", a variation on Regular Expressions where part-of-speech tags are delimited using angle brackets, e.g. <DT>?<JJ>*<NN>
- A dictionary mapping new tag names to tag patterns is used to initialize a Chunk Parser that will apply the rules we defined.

Chunking

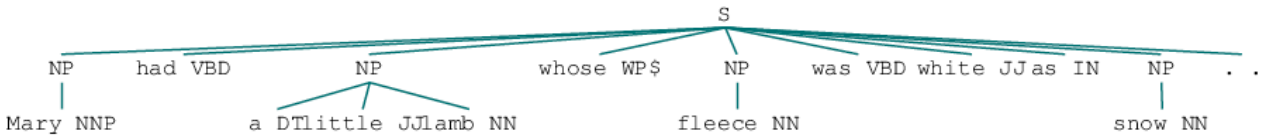
- Let us consider a simple example, of a grammar defining a noun phrase, NP:

grammar = "NP: {<DT>?<JJ.*>*<NN.*>+}"

When we apply this grammar to the sentence:

Mary had a little lamb whose fleece was white as snow.

- We obtain:



Chunking

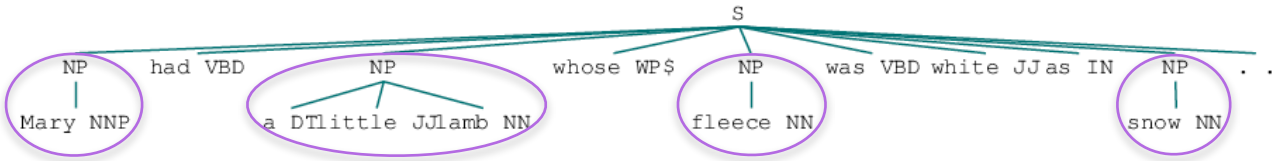
- Let us consider a simple example, of a grammar defining a noun phrase, NP:

grammar = "NP: {<DT>?<JJ.*>*<NN.*>+}"

When we apply this grammar to the sentence:

Mary had a little lamb whose fleece was white as snow.

- We obtain:



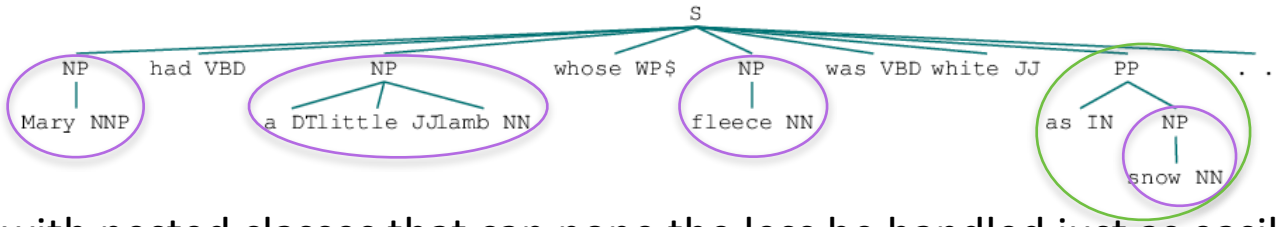
- Where 4 Noun-Phrases are correctly identified

Chunking

- grammars can be arbitrarily complex with varying numbers of new classes. For example:

```
grammar = r"""
    NP: {<DT>?<JJ.*>*<NN.*>+}
    PP: {<IN><NP>}
    VP: {<VB.*><NP|PP|CLAUSE>+$}
    CLAUSE: {<NP><VP>}"""
```

- Produces a significantly more complex tree structure:



- with nested classes that can none the less be handled just as easily

Lesson 3: Named Entity Recognition

3.1 - Part Of Speech Tagging

3.2 - Chunking

3.3 - Chinking

3.4 - Named Entity Recognition

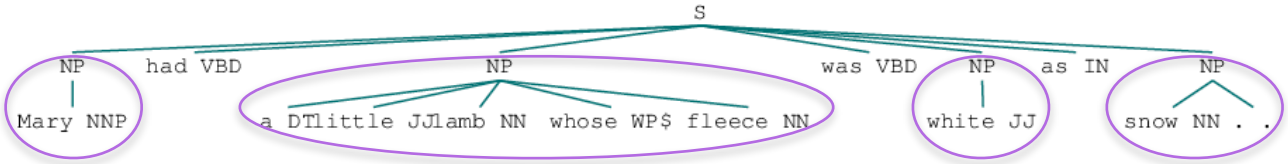
Chinking

- Chinking is a way to specify what to exclude from a chunk.
- You can specify a Chink as part of your grammar by inverting the curly braces (}…{ instead of {…})

```
grammar = r"""
                NP:
                    {<.*>+}             # Chunk everything
                    }<VBD|IN>+{      # Chink sequences of VBD and IN
"""
```

Chinking

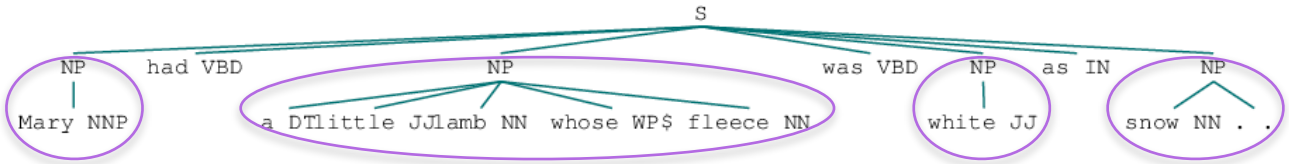
- With this new grammar our sentence becomes:



- where everything is mapped to an NP chunk except the VDB and IN pieces.

Chinking

- This example also helps illustrate how Chunks work:
  - Chunks and continuous, non-overlapping sequences of POS tagged tokens
  - When a portion is found that matches a Chink, then the Chunk is broken up in to 2 or more pieces



Lesson 3: Named Entity Recognition

3.1 - Part Of Speech Tagging

3.2 - Chunking

3.3 - Chinking

3.4 - Named Entity Recognition

Named Entities

- Named Entities are Noun Phrases that refer to individuals, organizations, dates, etc.
- One simple approach is to look up Noun Phrases in a gazetteer or entity directory, but this is limited and error prone
- nltk uses a classifier approach that is more robust and is able to recognizes several kinds of NE with the help of POS tags

Named Entities

NE Type	Examples
ORGANIZATION	Georgia-Pacific Corp., WHO
PERSON	Eddy Bonte, President Obama
LOCATION	Murray River, Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 million Canadian Dollars, GBP 10.40
PERCENT	twenty pct, 18.75 %
FACILITY	Washington Monument, Stonehenge
GPE	South East Asia, Midlothian

Named Entity Recognition

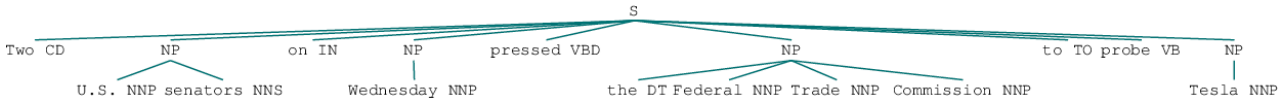
- Named Entity Recognition (NER) is a way to automatically identify mentions of named entities and is frequently a pre-processing step in Information Extraction applications.
- Two step process:
  - Delimit mentions of a NE (POS tagging and Chunking)
  - Identify the type of NE (lookup)
- nltk provides us with the utility ne_chunk() method to automatically performed Named Entity Recognition
- ne_chunk() takes POS tagged (and possibly chunked) tokens and it adds markers for Named Entities and their type

Named Entity Recognition

- Let us consider a recent news headline:

  “Two U.S. senators on Wednesday pressed the Federal Trade Commission to probe Tesla”

- After POS Tagging and Chunking we get:

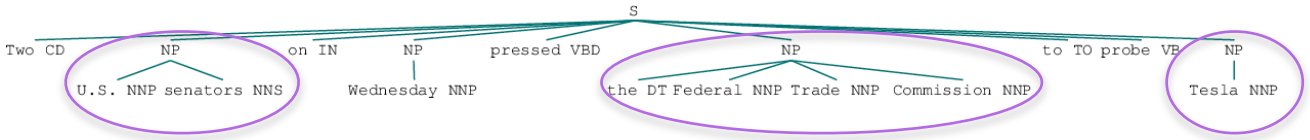


Named Entity Recognition

- Let us consider a recent news headline:

  “Two U.S. senators on Wednesday pressed the Federal Trade Commission to probe Tesla”

- After POS Tagging and Chunking we get:



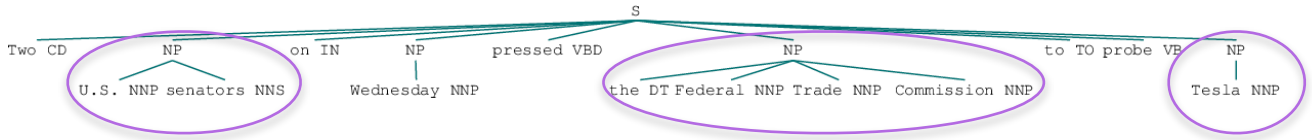
- where we correctly identified 3 noun phrases

Named Entity Recognition

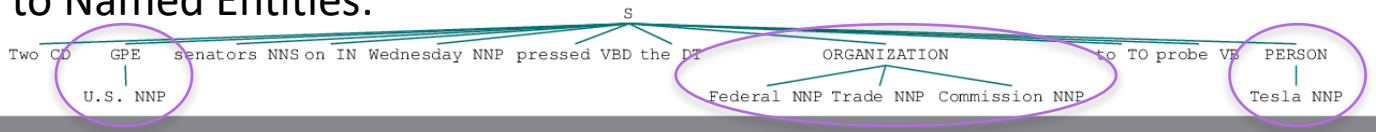
- Let us consider a recent news headline:

  “Two U.S. senators on Wednesday pressed the Federal Trade Commission to probe Tesla”

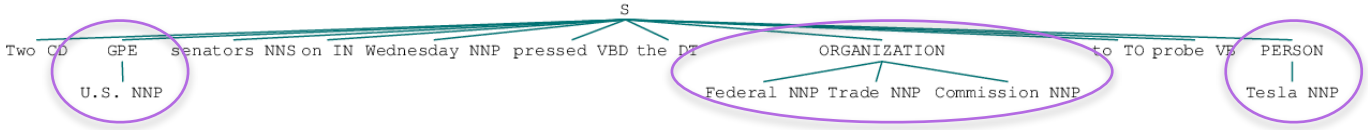
- After POS Tagging and Chunking we get:



- where we correctly identified 3 noun phrases which nltk is then able to map to Named Entities:



Named Entity Recognition



- it correctly identifies the US as a Geo-Political Entity and the FEC as an Organization, but it thinks that Tesla is a Person instead of an Organization.

Code - Named Entity Recognition

https://github.com/DataForScience/NLP_LL

Lesson 4: Topic Modelling

4.1 - Explicit Semantic Analysis
4.2 - Document Clustering
4.3 - Latent Semantic Analysis
4.4 - Latent-Dirichlet Allocation
4.5 - Non-Negative Matrix Factorization

Lesson 4: Topic Modelling

4.1 - Explicit Semantic Analysis

4.2 - Document Clustering

4.3 - Latent Semantic Analysis

4.4 - Latent-Dirichlet Allocation

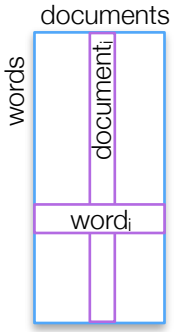
4.5 - Non-Negative Matrix Factorization

Topics

- A common application of NLP is to Search and Information Extraction.
- Can we automatically define the subject of a document?
- We saw in the previous lesson that some words are more meaningful than others.
- Based on the importance of each word for a specific document, it should be possible to characterize its topic.

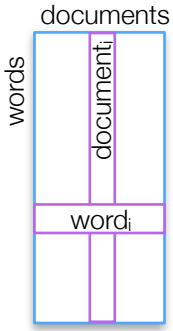
Term-Document Matrix

- We already know how to represent documents in terms of the TFIDF weights of each of their words.
- If we similarly use a vector representation where each element corresponds to a specific word, we can represent a corpus of documents as a matrix where each column is a document.
- Conversely, each word can be thought of as being represented by a row vector defined by its importance over all documents.



Explicit Semantic Analysis

- In the term document matrix, each word is defined, explicitly, by its contribution for each document.
- We can infer the meaning of each word by the concepts (documents) it contributes to.
- Typically, the English Wikipedia is used as the knowledge base (corpus).
- Using the TD matrix we can represent any (other) document by the sum (or average) over all the words it contains. This is similar to what we did with one-hot encodings to define bag-of-words.



Explicit Semantic Analysis

- The similarity of words or new documents can be measured using the cosine similarity:

$$\text{sim}\left(\vec{u}, \vec{v}\right) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \, ||\vec{v}||}$$

- Explicit semantic analysis, despite its simplicity, has been shown to improve the performance of different kinds of systems for search.
- The main disadvantage of ESA is that it requires the use of a large knowledge base corpus (the entire English Wikipedia!), resulting in high dimensional representations of words and documents.

Lesson 4: Topic Modelling

4.1 - Explicit Semantic Analysis

4.2 - Document Clustering

4.3 - Latent Semantic Analysis

4.4 - Latent-Dirichlet Allocation

4.5 - Non-Negative Matrix Factorization

Document Clustering

- Using the cosine similarity:

$$\text{sim}\left(\vec{u}, \vec{v}\right) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \, ||\vec{v}||}$$

- where $\vec{u}$ and $\vec{v}$ are document vectors, we can easily measure the similarity between every pair of documents in our corpus to generate a square similarity matrix.
- There is a large literature on clustering algorithms for matrices.
- The simplest approach is to impose a minimum similarity cutoff and consider any pair of documents with higher similarity to be part of the same cluster.

Hierarchical Document Clustering

- Algorithm
  - Assign each document to its own cluster.
  - Calculate the similarity matrix between all clusters.
  - Identify the two most similar clusters and merge them.
  - Recompute the similarity matrix for this reduced set of clusters.
  - Repeat until we reach the desired number of clusters.
- Library implementations will usually merge all the clusters until there is just one left so that any cutoff can be imposed a posteriori.

Lesson 4: Topic Modelling

4.1 - Explicit Semantic Analysis

4.2 - Document Clustering

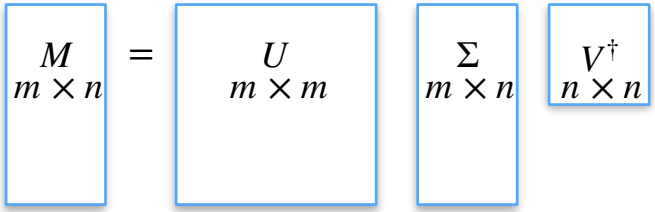
4.3 - Latent Semantic Analysis

4.4 - Latent-Dirichlet Allocation

4.5 - Non-Negative Matrix Factorization

Singular Value Decomposition (SVD)

- Any matrix M, such as the TD matrix seen above, can be decomposed into three matrices of the form:



- where

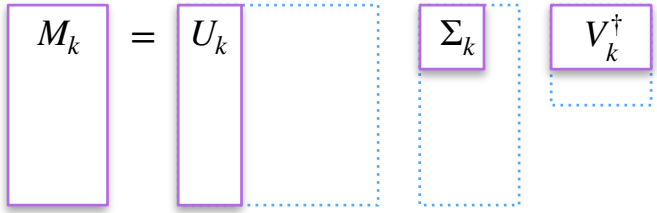
  U is a unitary matrix ($UU^\dagger = 1$)

  Σ is diagonal with non-negative elements

  $V^\dagger$ is a unitary matrix

Singular Value Decomposition (SVD)

- The diagonal elements of Σ, σ_i are called the singular values of the matrix M and are conceptually similar to eigenvalues in the case of square matrices.
- σ_i are sorted from largest to smallest.
- The original matrix M can be approximated by keeping only the top k dimensions of the SVD decomposition.



- Naturally, the larger the value of k the better the approximation.

Latent Semantic Analysis (LSA)

- While ESA explicit defines each word based on the documents in contributes to in the knowledge base, LSA tries to implicitly determine a “latent” representation for each word and document.
- LSA defines the term-document matrix for the corpus under consideration (as opposed to an external knowledge base).
- The DT matrix is approximated using a k dimensional singular value decomposition
- Each of the k latent dimensions used represents a latent dimension (topic) of the underlying dataset.

Latent Semantic Analysis (LSA)

- The U_k represents the distribution of each topic among the words in our vocabulary, while the V_k matrix describes the distribution of each document across topics.
- Documents and words can be more effectively clustered in the singular space.
- New documents (or queries) can be mapped into the singular space using:

$$\hat{v} = \Sigma_k^{-1} U_k^{\dagger} v$$

where v is the column vector representing the original query and $\hat{v}$ the transformed document vector.

Lesson 4: Topic Modelling

4.1 - Explicit Semantic Analysis

4.2 - Document Clustering

4.3 - Latent Semantic Analysis

4.4 - Latent-Dirichlet Allocation

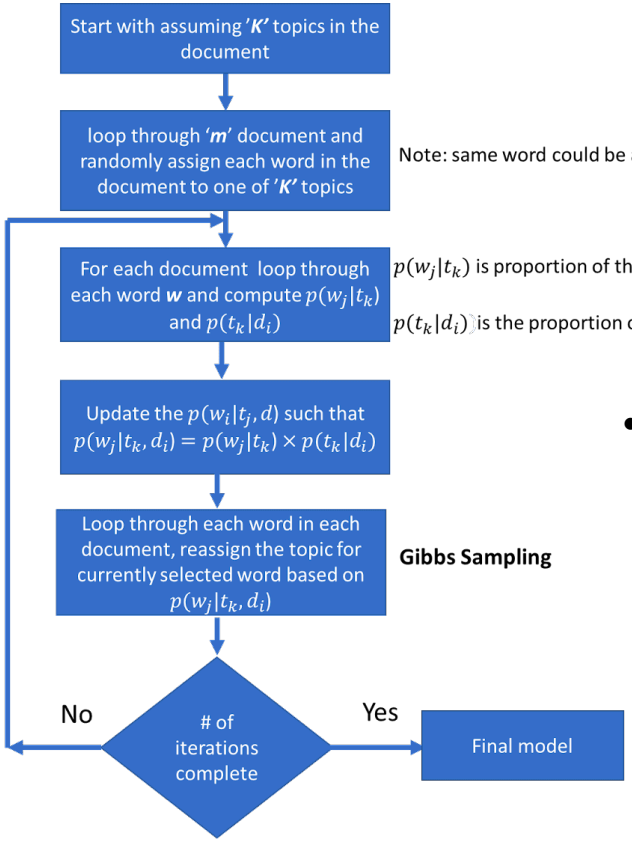
4.5 - Non-Negative Matrix Factorization

Latent Dirichlet Allocation

- Latent Dirichlet Allocation (LDA) is a generative (statistical) model that defines each document to be a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.
- It was first introduced in the context of NLP in 2003.
- LDA is based upon two assumptions:
  - Documents are mixtures of topics
  - Topics are mixtures of words
- The algorithm proceeds by assigning words and documents to topics in an iterative way

Latent Dirichlet Allocation

https://www.mygreatlearning.com/blog/understanding-latent-dirichlet-allocation/



- The algorithm proceeds for a fixed number of iterations or until the probability distributions converge

Lesson 4: Topic Modelling

4.1 - Explicit Semantic Analysis

4.2 - Document Clustering

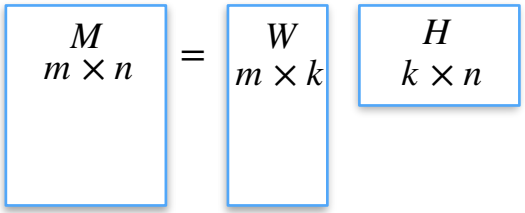
4.3 - Latent Semantic Analysis

4.4 - Latent-Dirichlet Allocation

4.5 - Non-Negative Matrix Factorization

Non-Negative Matrix Factorization

- Matrix factorization methods, like SVD, have a long history of application to natural language processing.
- Another common factorization method used to identify a latent structure to a dataset is non-negative matrix factorization (NMF).



- NMF directly approximates the matrix M for a given value of k.

Non-Negative Matrix Factorization

- In this formulation, it has the advantage of being easily interpretable:
  - Columns of W are the underlying basis vectors for each topic.
  - Columns of H are the contribution of each topic to a specific document.
- The value of W and H are found through an optimization procedure where we minimize the error of the approximation:

$$\min_{W,H} \| M - WH \|_F^2$$

- Fortunately for us, sklearn has a very robust implementation we can just use!

Code - Topic Modelling

https://github.com/DataForScience/NLP_LL

Lesson 5: Sentiment Analysis

5.1 - Quantify Words and Feelings

5.2 - Negations and Modifiers

5.3 - Corpus Based Approaches

Lesson 5: Sentiment Analysis

5.1 - Quantify Words and Feelings

5.2 - Negations and Modifiers

5.3 - Corpus Based Approaches

Positive and Negative Words

- We often use words to describe how we are feeling or how we feel about something
- People associate specific connotations and meanings to words
- Some are obvious:
  - Love, yes, friendship, good, … transmit positive feelings
  - Hate, no, animosity, bad, … transmit negative feelings
- Others less so:
  - Blue, maybe, indifference, …

Positive and Negative Texts

- We can use the words in a text to determine the sentiment behind the text
- The simplest approach:
  - count positive P and negative N words
  - define sentiment as:

$$\text{sentiment} = \frac{P - N}{P + N}$$

- This effectively weighs positive words as $+1$ and negative words as -1 and defines sentiment as how dominant one sentiment is over another
- Despite it's simplicity, this approach is surprisingly powerful

Word Valence

- Many lexicons of positive and negative words are available online:
  - Opinion Lexicon: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
  - Opinion Finder: https://mpqa.cs.pitt.edu/opinionfinder/
  - Valence Aware Dictionary and sEntiment Reasoner (VADER) https://github.com/cjhutto/vaderSentiment
- Even commercial products use similar approaches:
  - Linguistic Inquiry and Word Count (LIWC) http://liwc.wpengine.com/

VADER

- VADER includes over 7 thousand unique tokens and provides 10 hand-curated individual valence scores for each of them in addition to the mean score and standard deviation

	word	mean	std	scores
5499	rebelling	-1.1	1.51327	[-3, -1, -2, -1, -2, -1, -1, -3, 2, 1]
2187	disliked	-1.7	0.64031	[-2, -3, -2, -1, -1, -1, -2, -2, -1, -2]
5289	pressurizers	-0.7	0.90000	[-1, 0, -2, -1, -2, -1, 0, 0, 1, -1]
5369	promiscuities	-0.8	1.32665	[-1, 0, -2, -2, -1, 2, -3, -1, 0, 0]
5738	rigorous	-1.1	1.51327	[-2, -3, 1, 0, 1, 0, -1, -1, -3, -3]
2131	discard	-1.0	0.44721	[-1, -1, -1, -1, 0, -1, -1, -1, -2, -1]
1968	despair	-1.3	1.90000	[2, -1, -3, -1, -3, 1, -3, 1, -3, -3]
765	antagonized	-1.4	0.66332	[-2, -1, -2, -2, -1, 0, -2, -1, -1, -2]
1175	boycott	-1.3	0.45826	[-2, -1, -1, -1, -1, -2, -1, -1, -2, -1]
2085	dignify	1.8	0.74833	[1, 2, 2, 1, 3, 3, 1, 2, 2, 1]

- nltk provides support for VADER through the SentimentIntensityAnalyzer module in nltk.sentiment

VADER

- support for VADER through the SentimentIntensityAnalyzer module in nltk.sentiment
- SentimentIntensityAnalyzer provides a polarity_scores() method that takes a string as input (not tokens)
- polarity_scores() returns the polarity values across 4 dimensions:
  - neg - negative
  - neu - neutral
  - pos - positive
  - compound - a combination of all three
- scores are normalized such that neg + neu + pos = 1 and compound ranges from -1 to 1.

Lesson 5: Sentiment Analysis

5.1 - Quantify Words and Feelings

5.2 - Negations and Modifiers

5.3 - Corpus Based Approaches

Negations and Modifiers

- So far, our approach to sentiment analysis has relied on considering just individual words
- However it's clear that context matters.
- "not pretty" is very different from "pretty"
- n-grams are a natural extension, but increase significantly the memory requirements
- An intermediate approach is to consider modifier words like "not", "much", "little", "very", etc.
- By keeping track of the modifiers we can generate n-grams on the fly

Modifiers

- While sentiment words contribute additively to the sentiment score, modifiers have a multiplicative effect
- If we define the weights of each modifier

not	-1
very	1.5
somewhat	1.2
pretty	1.5
…	…

- We just have to check the previous word whenever we encounter one of the sentiment words in our lexicon
- Special care must be taken whenever a modifier word can also be a sentiment word (such as “pretty”).

Modifiers

- We must keep two separate dictionaries of words: modifiers and valence words.
- For each word we encounter, we first check whether it is a modifier and only then whether it is valence word.
- Words that are in neither list act as a signal to end the current n-gram
- With this simple approach we can handle even long sequences of modifiers, double negatives, etc.

Lesson 5: Sentiment Analysis

5.1 - Quantify Words and Feelings

5.2 - Negations and Modifiers

5.3 - Corpus Based Approaches

Corpus-based Approaches

- Sentiment analyses often rely on dictionaries of words and valences
- In many cases, these lists are curated manually with varying degrees of care
- Can we automatically generate them?
- With the advent of the Web, large corpora of product reviews became available
- Each review associates a piece of text with a numerical evaluation (typically 1-5 stars)

Corpus-based Approaches

- Corpus based approaches to sentiment analysis rely on these datasets to generate the lexicons used
- These approaches leverage sophisticated supervised machine learning techniques to automatically determine the weight that should be assigned to each word
- Modifiers can be identified using Part-of-Speech (POS) tagging

Corpus-based Approaches

- Hand curated lexicons are naturally subjective and potentially incomplete
- Lexicons generated automatically from large corpora can cover a wider range of languages and subjects
- Hand curation is more powerful when only smaller datasets are available
- Automatic generation relies on large datasets and their inherent biases. They are typically only applicable to specific domains and may appear to be more objective.
- Your results will only be as good as your lexicon. Make sure to understand its biases and limitations.

Code - Sentiment Analysis

https://github.com/DataForScience/NLP_LL

Lesson 6: Text Classification

6.1 - Feed Forward Networks

6.2 - Convolutional Neural Networks

6.3 - Applications

Lesson 6: Text Classification

6.1 - Feed Forward Networks

6.2 - Convolutional Neural Networks

6.3 - Applications

Text Classification

- Our prototypical example will be Text Classification
- We'll learn how to classify IMDB reviews as Positive or Negative, a simplified version of Corpus Based Sentiment Analysis
- Reviews can have arbitrary lengths and vocabulary

"<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of norman and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

"<START> worst mistake of my life br br i picked this movie up at target for 5 because i figured hey it's sandler i can get some cheap laughs i was wrong completely wrong mid way through the film all three of my friends were asleep and i was still suffering worst plot worst script worst movie i have ever seen i wanted to hit my head up against a wall for an hour then i'd stop and you know why because it felt damn good upon bashing my head in i stuck that damn movie in the <UNK> and watched it burn and that felt better than anything else i've ever done it took american psycho army of darkness and kill bill just to get over that crap i hate you sandler for actually going through with this and ruining a whole day of my life"

Text Classification

- For convenience, we'll consider only the 10,000 most frequent words and truncate the reviews at 500 words.
- Removed words are marked by a special **<UNK>** token

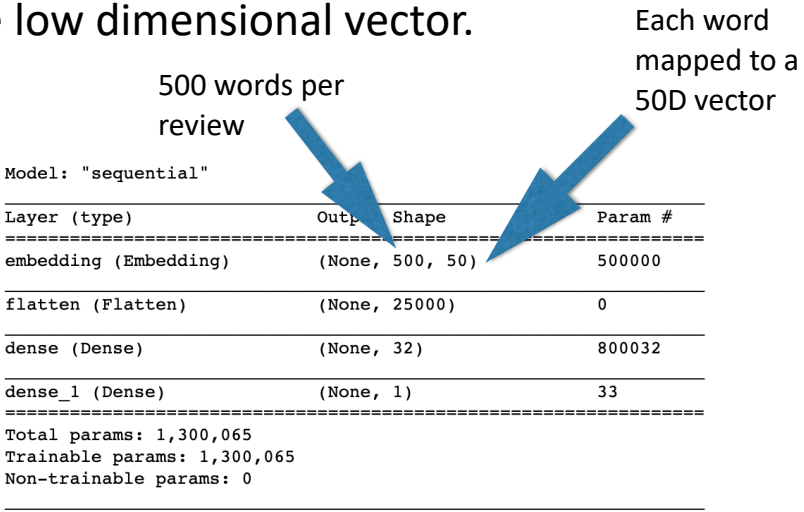
Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 50)           500000
_________________________________________________________________
flatten (Flatten)            (None, 25000)             0
_________________________________________________________________
dense (Dense)                (None, 32)                800032
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,300,065
Trainable params: 1,300,065
Non-trainable params: 0
_________________________________________________________________
```

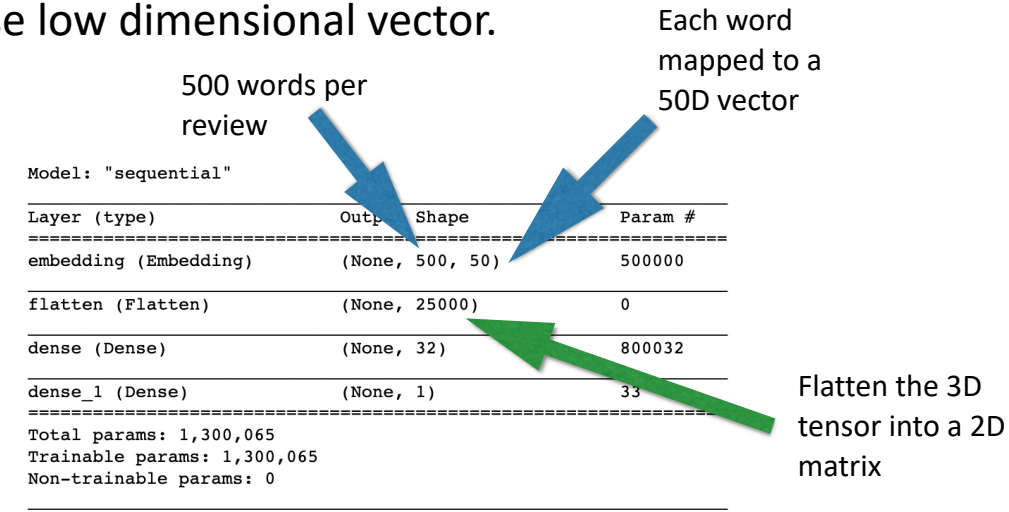
Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.



Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.

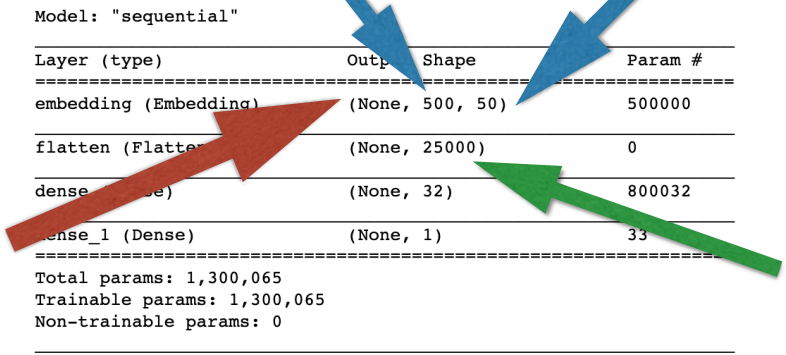


Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.

500 words per review

Each word mapped to a 50D vector



None input dimension means that batch_size is defined at training time

Flatten the 3D tensor into a 2D matrix

Lesson 6: Text Classification

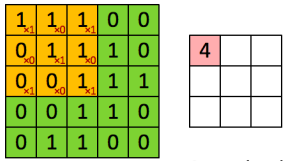
6.1 - Feed Forward Networks

6.2 - Convolutional Neural Networks

6.3 - Applications

Convolutional Neural Network

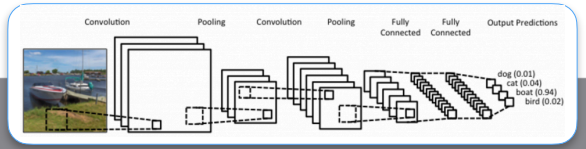
- Originally developed for Image Processing
- A Convolution Layer computes a value along a moving window as it slides through the image
- The output of the Convolution is smaller than the original image while still capturing relevant information



Image

Convolved Feature

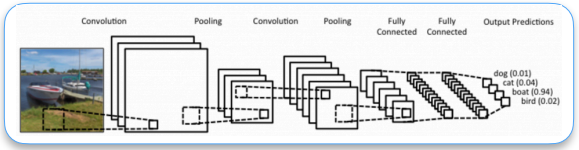
- Different convolution operations produce different effects on the original image:



Convolutional Neural Network

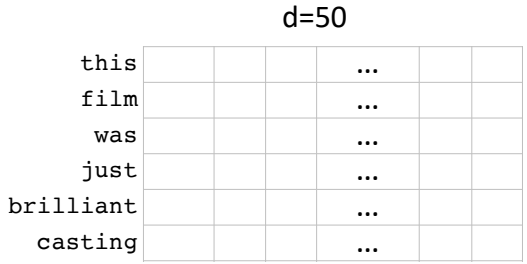
http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

- Different convolution operations produce different effects on the original image:
  - Extract Edges, Blur, Emboss, etc
- Convolution layers are used to extract features from the original image

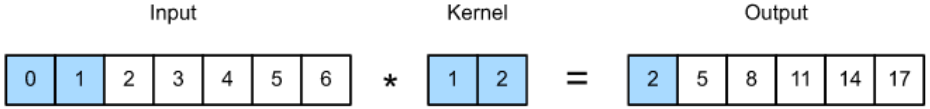


Convolutional Neural Network

- Images are just arrays of numbers, just like our input matrices of words!



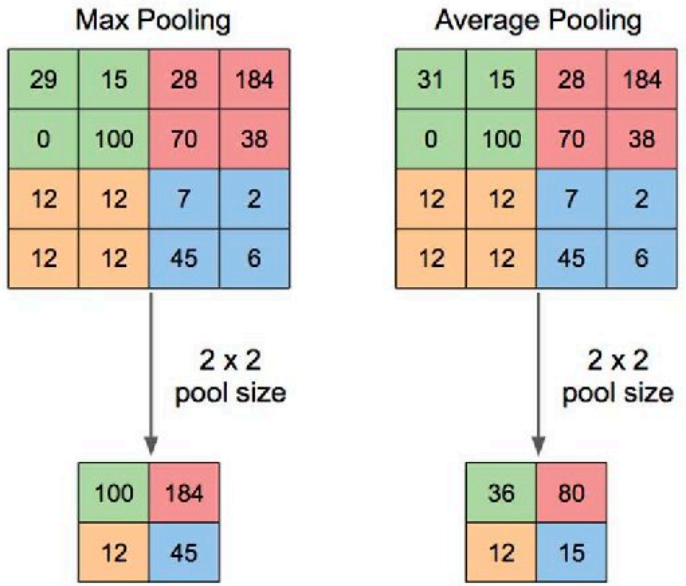
- The Kernel for each Conv1D layer can be learned by the Network itself



- And the output layers will have smaller dimension than the input

Pooling Layers

- Pooling layers reduce the dimensionality of the data by generating summaries of the values.
- Most common examples are MaxPooling which returns the Maximum value over a certain area and AveragePooling which returns the average value.
- The main goal is to produce a version of the data that has less noise and better defined features than the original



Convolutional Neural Network

Each word vector gets transformed from 50D to 32D by the convolution layer

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 50)           500000
_________________________________________________________________
conv1d (Conv1D)              (None, 500, 32)           4832
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 250, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                256032
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 760,897
Trainable params: 760,897
Non-trainable params: 0
_________________________________________________________________
```

Each review vector gets transformed from 500D to 250D through MaxPooling

Lesson 6: Text Classification

6.1 - Feed Forward Networks

6.2 - Convolutional Neural Networks

6.3 - Applications

Applications

- The Feed-Forward Networks and Convolutional Neural Networks architectures we explored are just simple examples of what's possible when we combine NLP and Machine Learning approaches.
- Text Classification is a huge and growing field with many applications in business, industry and academia
- We've touched only the fundamental ideas and principles and there's much more to learn

Applications

- Spam Detection - Spam messages are a huge issue to email providers and users alike. Effective classification can result in a significant improvement in service quality
- Tagging Products - We could easily modify the approaches we just demonstrated to classify the movie genre based on the review text
- Search Optimization - Identify query intention based on previous search history
- Language or Dialect Detection - Languages from the same family and Dialects within a language can have many words in common requiring sophisticated methods to distinguish in an automated fashion

Applications

- Bias detector - Identifying inadvertent political or other biases in public communications
- Inappropriate Content - Detecting inappropriate content or bullying in user posted content in online platforms
- Chat Bots - Select the correct answer based on the user query and past interactions

Code - Text Classification

https://github.com/DataForScience/NLP_LL

Lesson 7: Sequence Modeling

7.1 - Recurrent Neural Networks (RNN)

7.2 - Gated Recurrent Unit (GRU)

7.3 - Long-Short Term Memory (LSTM)

7.4 - Auto-Encoder Models

Lesson 7: Sequence Modeling

7.1 - Recurrent Neural Networks (RNN)

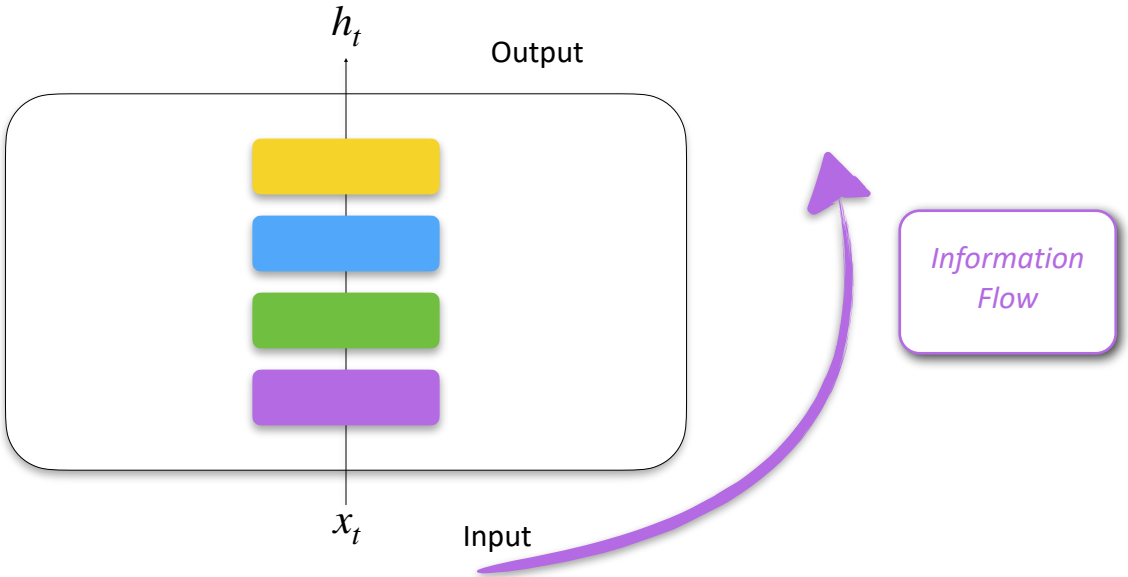
7.2 - Gated Recurrent Unit (GRU)

7.3 - Long-Short Term Memory (LSTM)

7.4 - Auto-Encoder Models

Feed Forward Network (FFN)

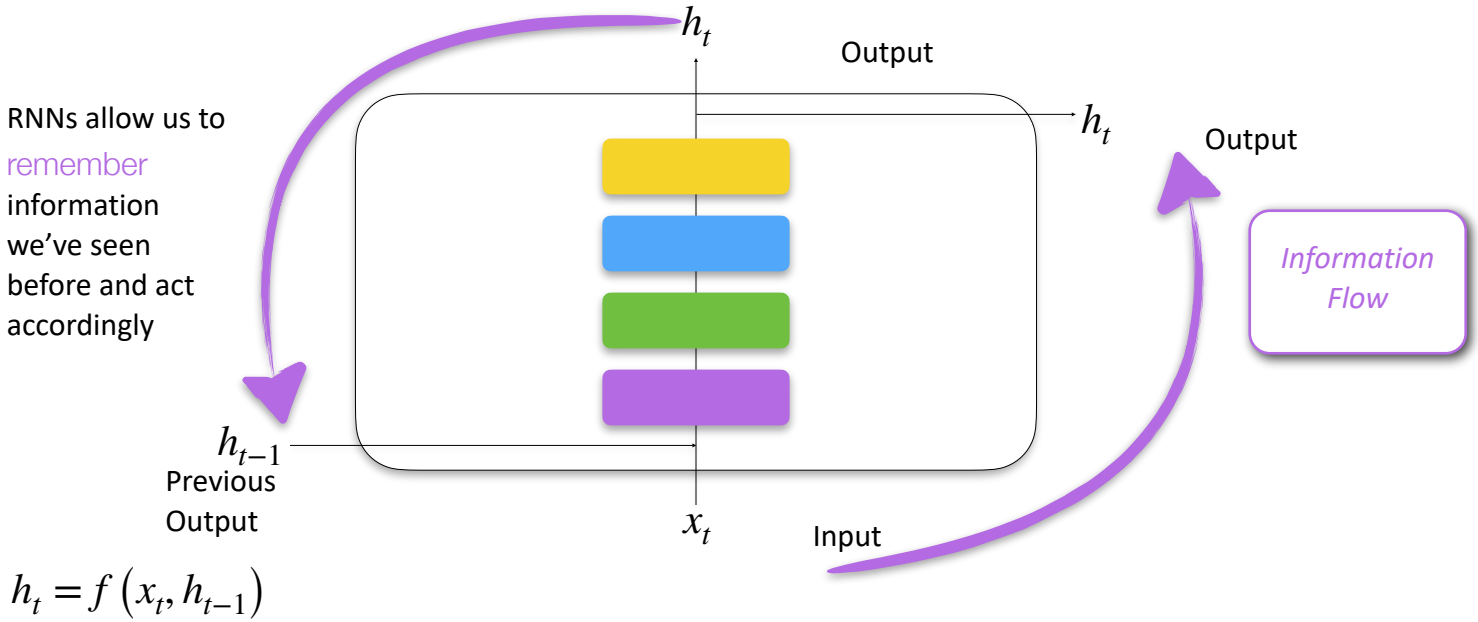
- FFNs provide a straight path from input to output



$$h_t = f(x_t)$$

Recurrent Neural Network (RNN)

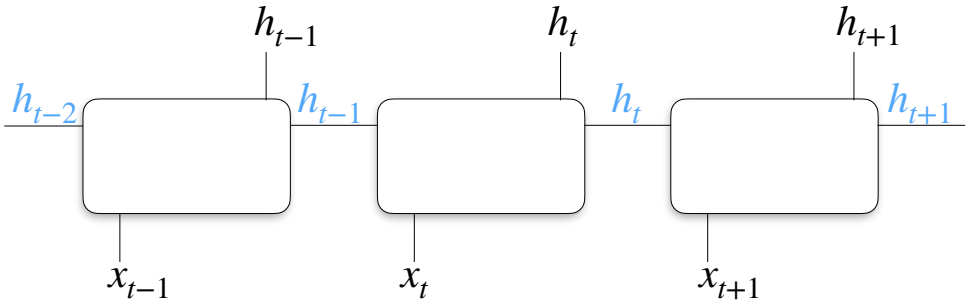
- RNNs follow a different paradigm



$$h_t = f\left(x_t, h_{t-1}\right)$$

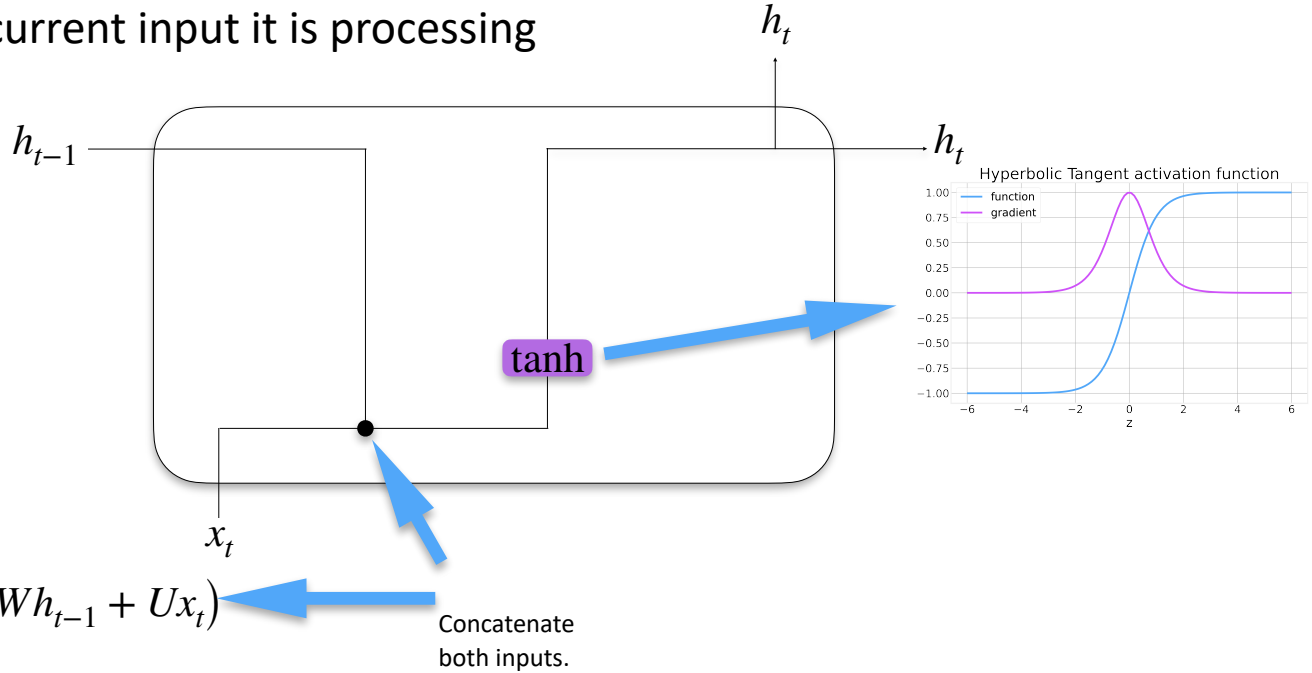
Recurrent Neural Network (RNN)

- Each output depends (implicitly) on all previous outputs.
- RNNs are particularly useful to model sequential systems, like time series, audio or streams of text
- Input sequences generate output sequences (seq2seq)



Recurrent Neural Network (RNN)

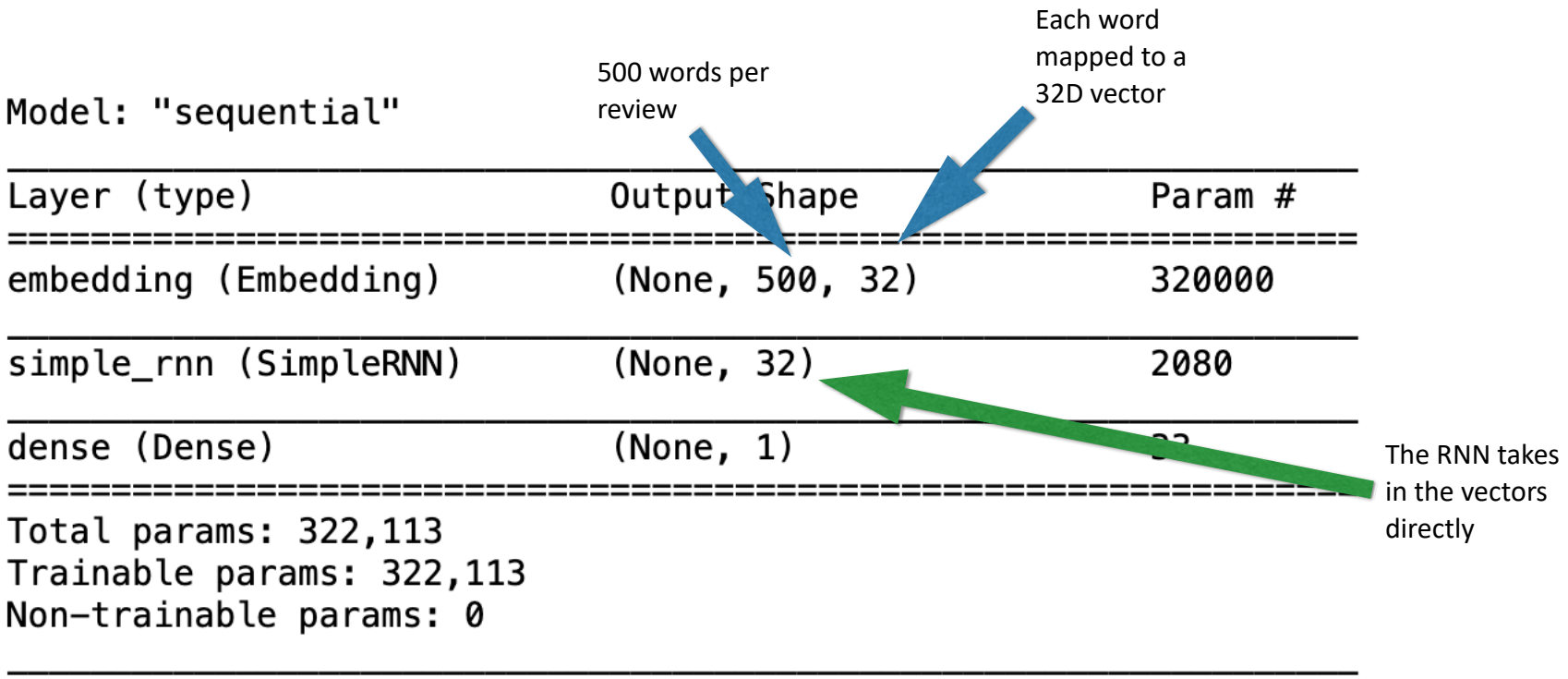
- SimpleRNN simply concatenates the last output it produced to the current input it is processing



$$h_t = \tanh\left(Wh_{t-1} + Ux_t\right)$$

Concatenate both inputs.

Recurrent Neural Network (RNN)



Lesson 7: Sequence Modeling

7.1 - Recurrent Neural Networks (RNN)

7.2 - Gated Recurrent Unit (GRU)

7.3 - Long-Short Term Memory (LSTM)

7.4 - Auto-Encoder Models

Gated Recurrent Unit (GRU)

- Introduced in 2014 by K. Cho
- Meant to solve the Vanishing Gradient Problem
- Can be considered as a simplification of LSTMs
- Similar performance to LSTM in some applications, better performance for smaller datasets.

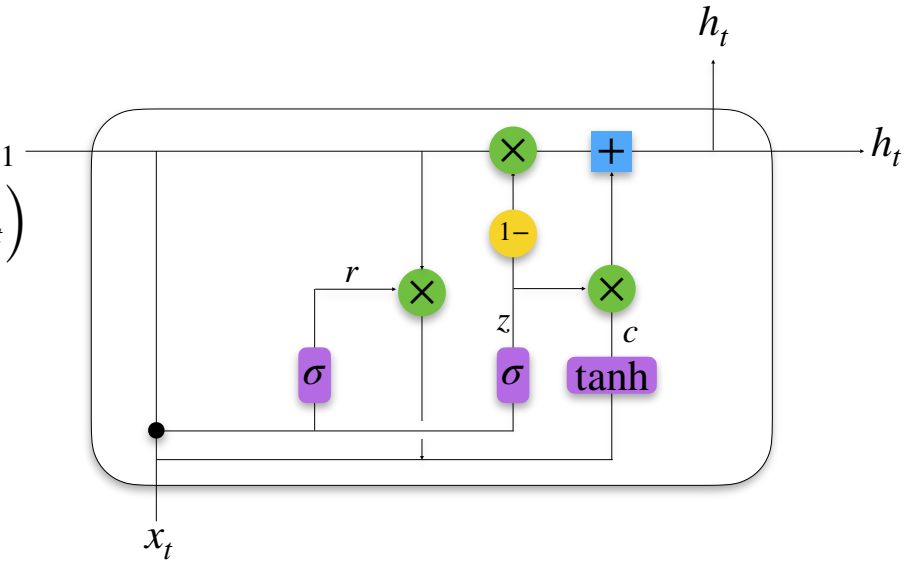
Gated Recurrent Unit (GRU)

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$



Element wise addition

Element wise multiplication

1– 1 minus the input

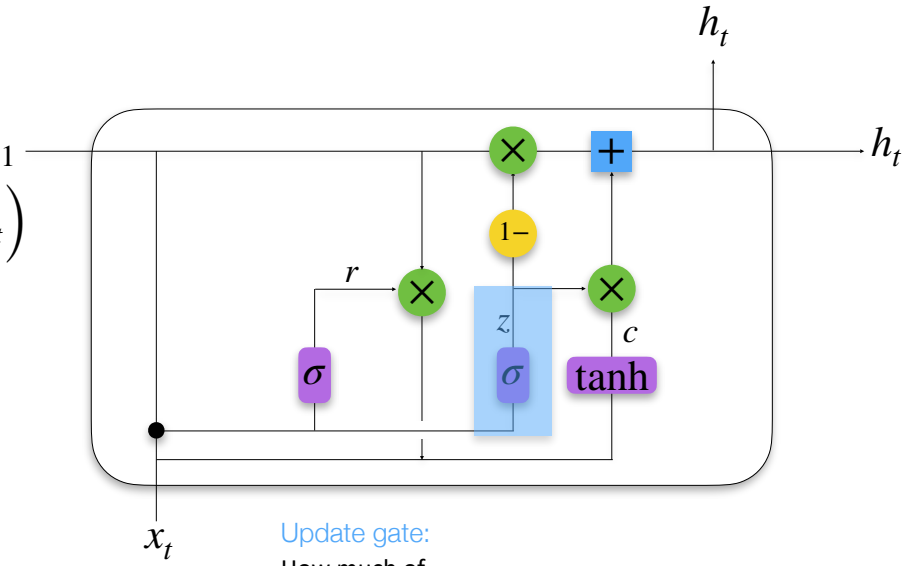
Gated Recurrent Unit (GRU)

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$



Element wise addition

Element wise multiplication

1– 1 minus the input

Update gate:
How much of the previous state should be kept?

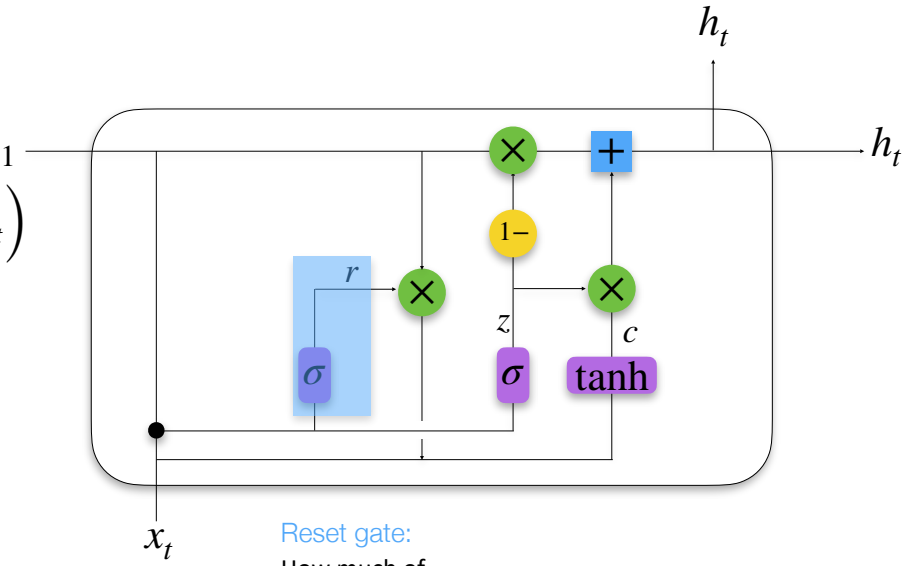
Gated Recurrent Unit (GRU)

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$



Element wise addition

Element wise multiplication

1– 1 minus the input

Reset gate:
How much of the previous output should be removed?

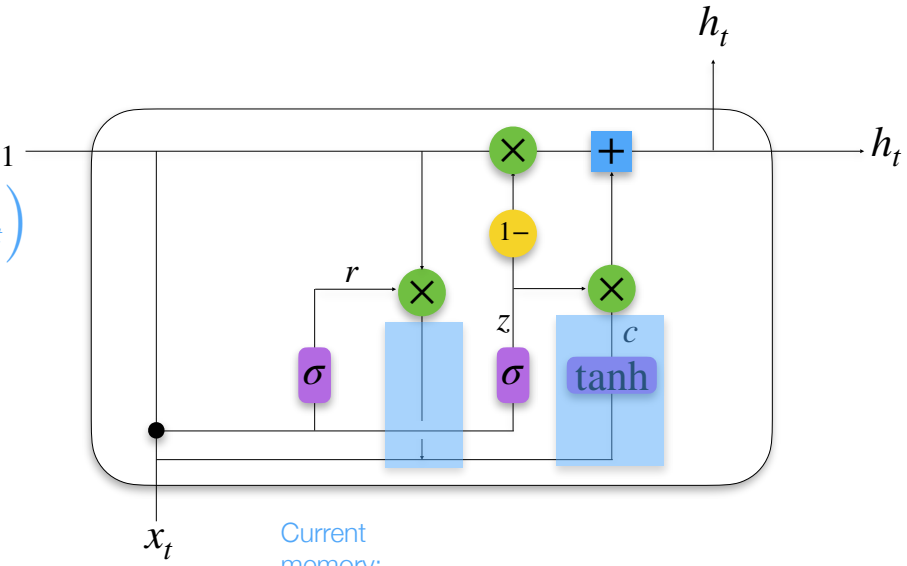
Gated Recurrent Unit (GRU)

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$



Element wise addition

Element wise multiplication

1– 1 minus the input

Current memory: What information do we remember right now?

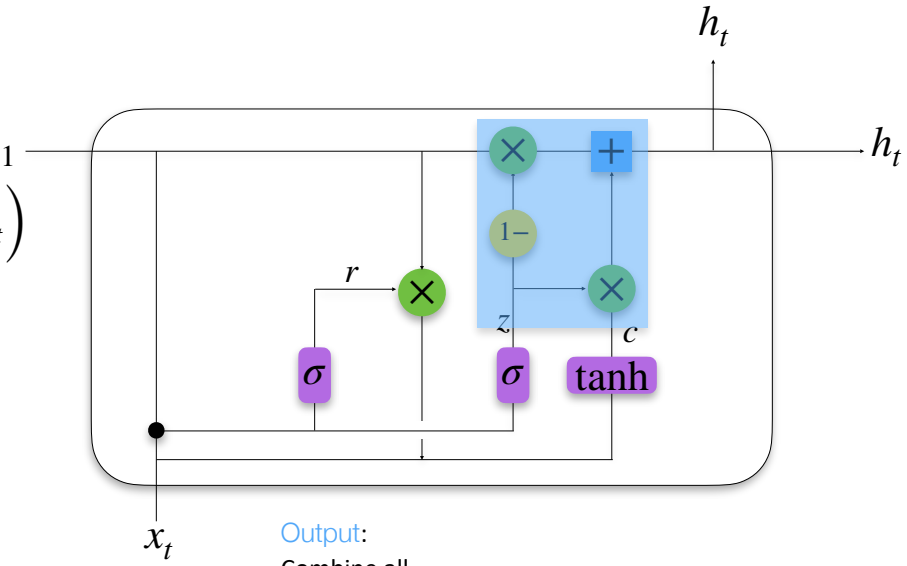
Gated Recurrent Unit (GRU)

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$



Element wise addition

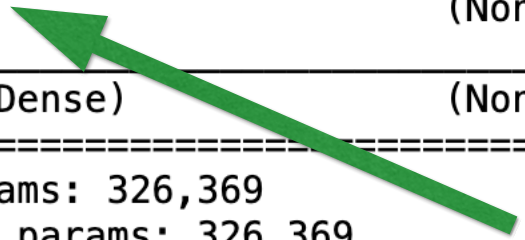
Element wise multiplication

1– 1 minus the input

Output:
Combine all available information.

Gated Recurrent Unit (GRU)

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           320000
_________________________________________________________________
gru (GRU)                    (None, 32)                6336
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 326,369
Trainable params: 326,369
Non-trainable params: 0
_________________________________________________________________
```



Just plugin a GRU instead of the RNN

Lesson 7: Sequence Modeling

7.1 - Recurrent Neural Networks (RNN)

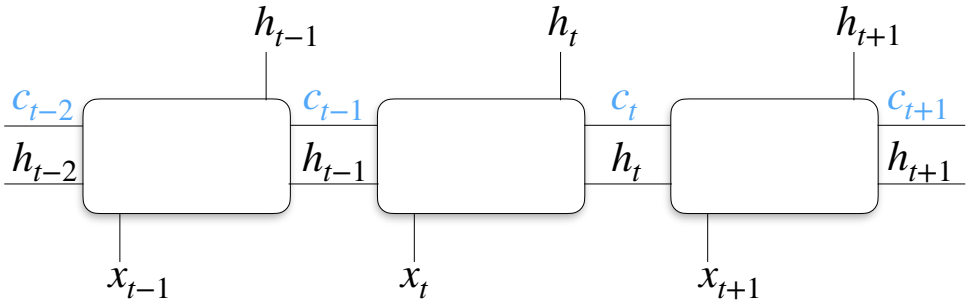
7.2 - Gated Recurrent Unit (GRU)

7.3 - Long-Short Term Memory (LSTM)

7.4 - Auto-Encoder Models

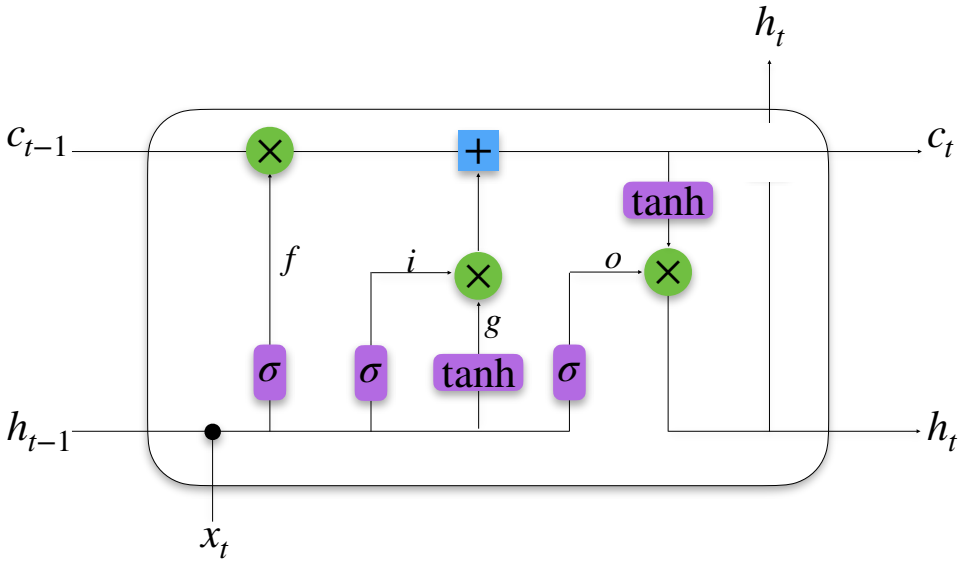
Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (memory)?
- How much information is kept, can be controlled through gates.
- First introduced in 1997 by Hochreiter and Schmidhuber



Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$
$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$
$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$
$$h_t = \tanh\left(c_t\right) \otimes o$$



Element wise addition

Element wise multiplication

1– 1 minus the input

Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$

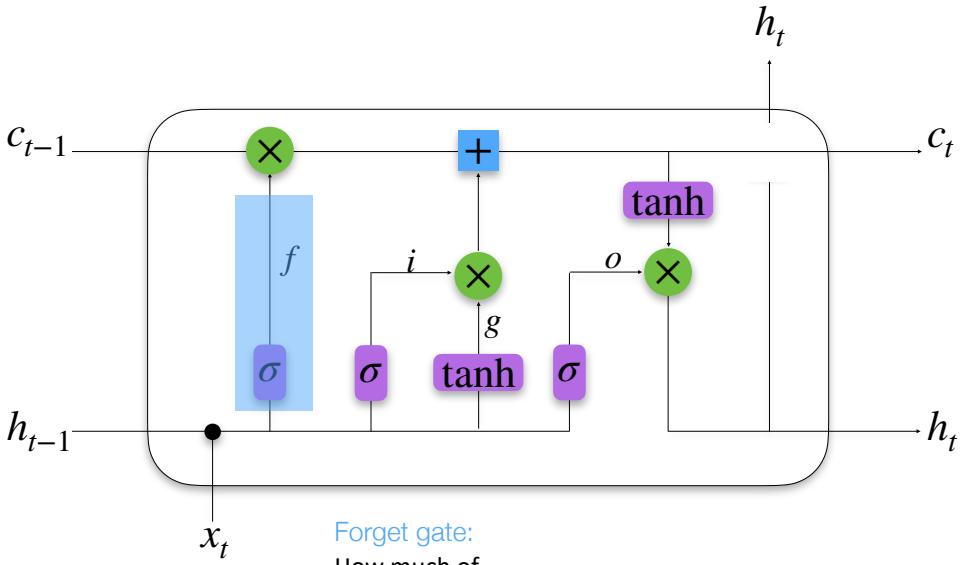
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$

$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$

$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

$$h_t = \tanh\left(c_t\right) \otimes o$$



Forget gate:
How much of the previous state should be kept?

Element wise addition

Element wise multiplication

1– 1 minus the input

Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$

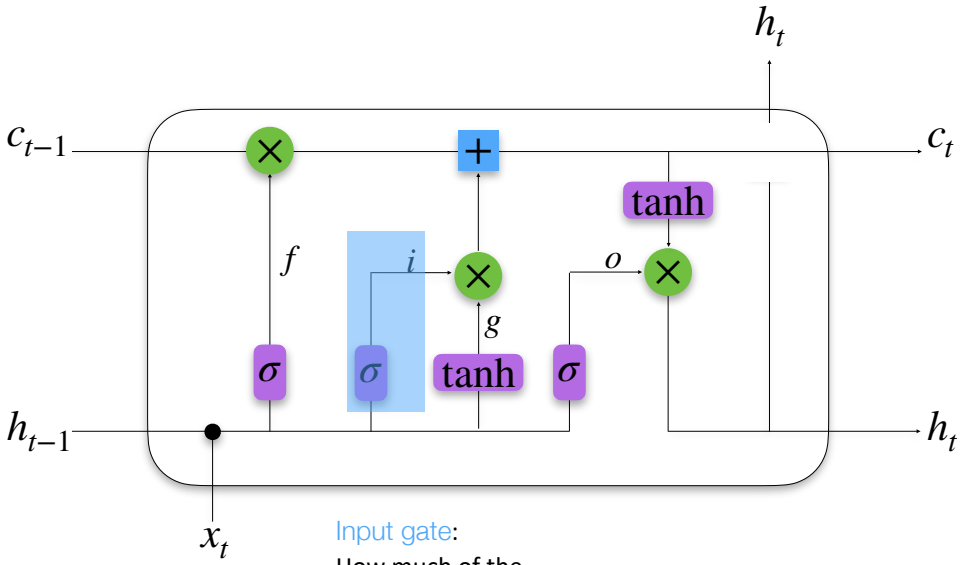
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$

$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$

$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

$$h_t = \tanh\left(c_t\right) \otimes o$$



Input gate:
How much of the previous output should be remembered?

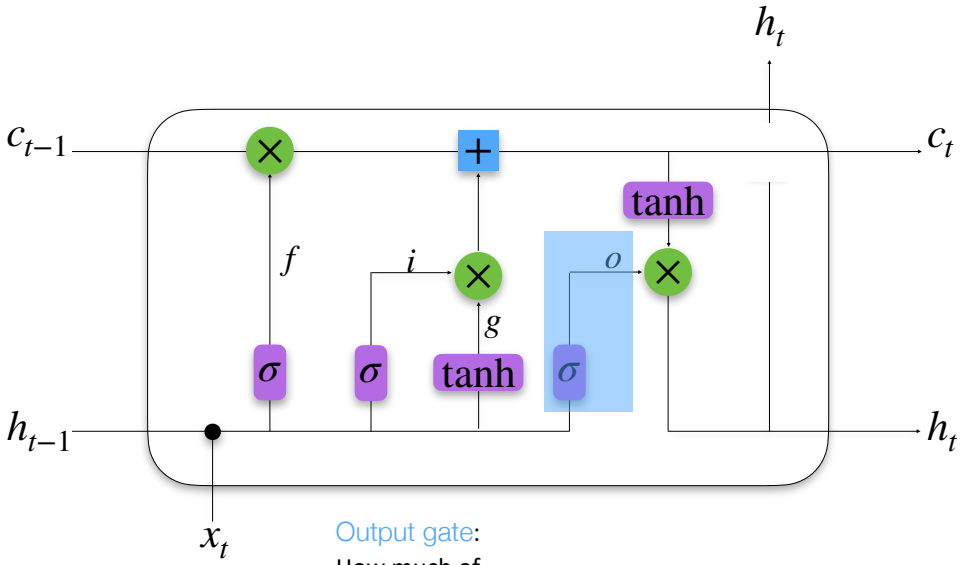
Element wise addition

Element wise multiplication

1– 1 minus the input

Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$
$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$
$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$
$$h_t = \tanh\left(c_t\right) \otimes o$$



Output gate: How much of the previous output should contribute?

Element wise addition

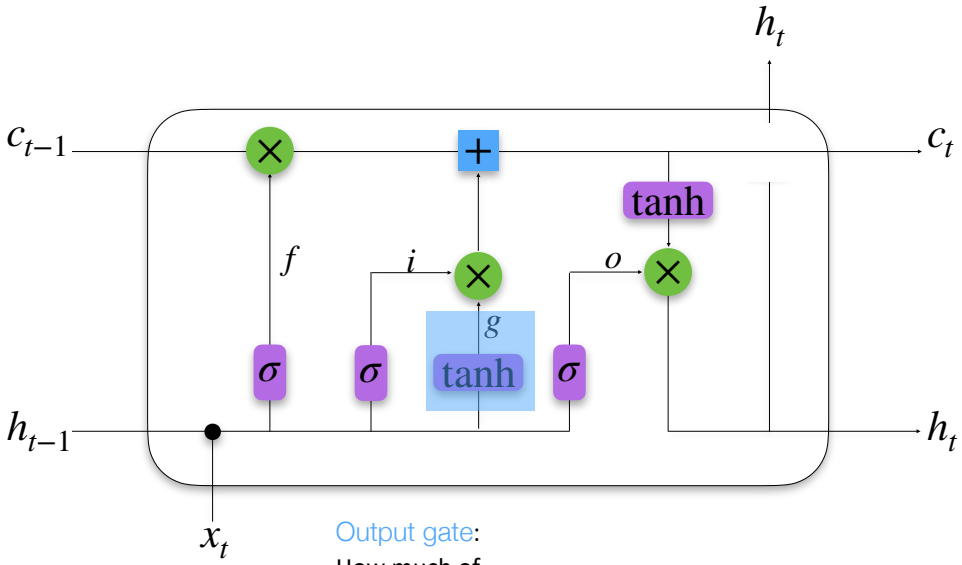
Element wise multiplication

1– 1 minus the input

All gates use the same inputs and activation functions, but different weights

Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$
$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$
$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$
$$h_t = \tanh\left(c_t\right) \otimes o$$



Element wise addition

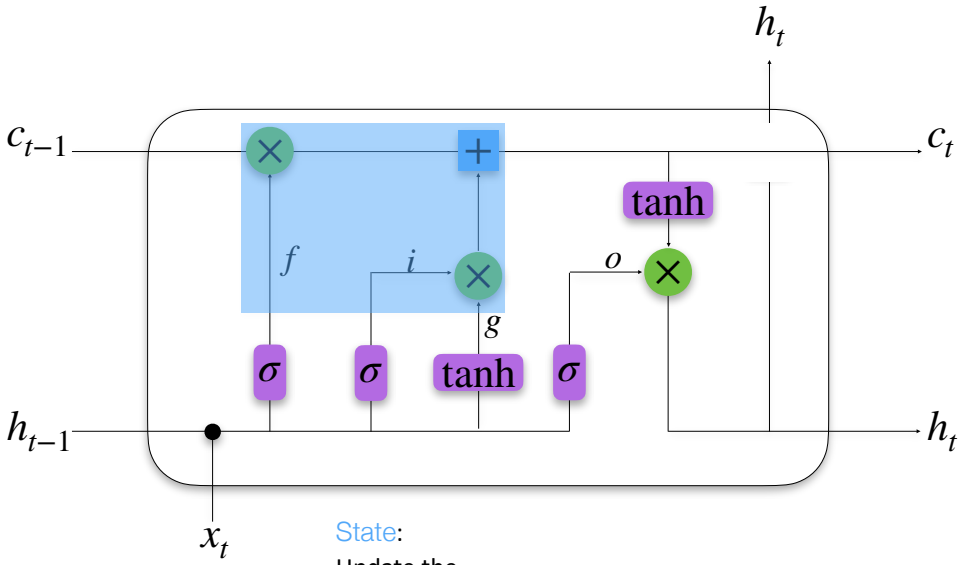
Element wise multiplication

1– 1 minus the input

Output gate: How much of the previous output should contribute?

Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$
$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$
$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$
$$h_t = \tanh\left(c_t\right) \otimes o$$



Element wise addition

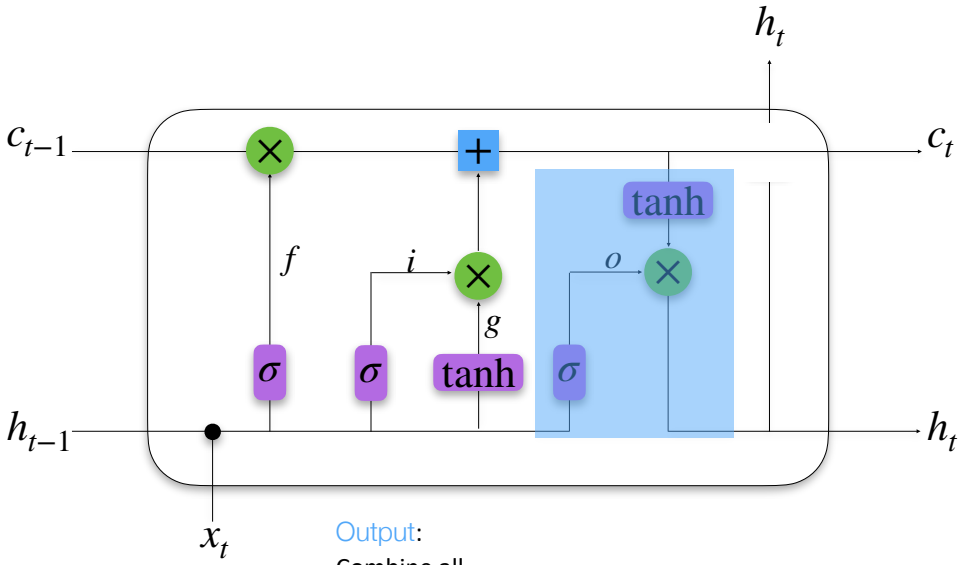
Element wise multiplication

1– 1 minus the input

State:
Update the
current state

Long-Short Term Memory (LSTM)

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right)$$
$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right)$$
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right)$$
$$g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$
$$c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$
$$h_t = \tanh\left(c_t\right) \otimes o$$



Element wise addition

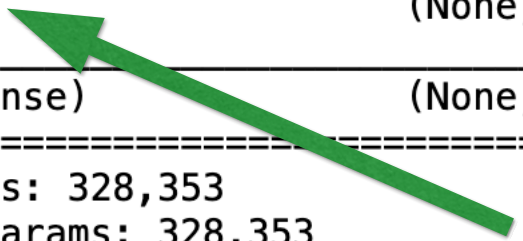
Element wise multiplication

1– 1 minus the input

Output:
Combine all available information.

Long-Short Term Memory (LSTM)

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           320000
_________________________________________________________________
lstm (LSTM)                  (None, 32)                8320
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```



Just replace the GRU or RNN with an LSTM

Lesson 7: Sequence Modeling

7.1 - Recurrent Neural Networks (RNN)

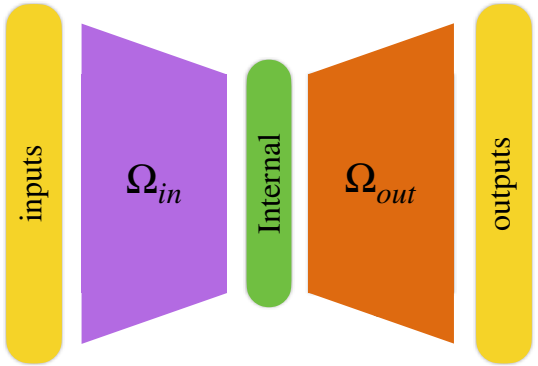
7.2 - Gated Recurrent Unit (GRU)

7.3 - Long-Short Term Memory (LSTM)

7.4 - Auto-Encoder Models

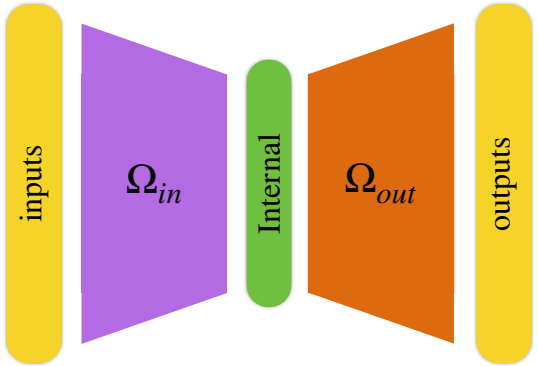
Auto-Encoders

- Auto-Encoders use the same values for both inputs and outputs
- The Internal/hidden layer(s) have a smaller number of units than the input



Auto-Encoders

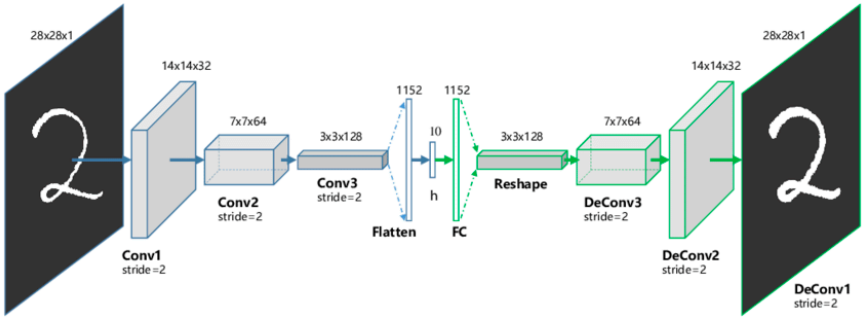
- The fundamental idea is that the Network needs to learn an internal representation of its inputs that is smaller but from which it is still possible to reconstruct the input.



- Think of it as “zipping” and “unzipping” the input values.

Auto-Encoders

- After training, the parts of the network that generate the internal representation can be used as inputs to other Networks
- This is similar to what we did when we reused the word embeddings generated by training a word2vec network
- Auto-encoders can be arbitrarily complex, including many layers between the input and the internal representation (or Code) and are often used in Image Processing to generate efficient representations of complex images



Code - Sequence Modeling

https://github.com/DataForScience/NLP_LL

Lesson 8: Applications

8.1 - word2vec embeddings

8.2 - GloVe

8.3 - Transfer Learning

8.4 - Language Detection

Lesson 8: Applications

8.1 - word2vec embeddings

8.2 - GloVe

8.3 - Transfer Learning

8.4 - Language Detection

Word Embeddings

- The distributional hypothesis in linguistics states that words with similar meanings should occur in similar contexts.
- In other words, from a word we can get some idea about the context where it might appear.

___ ___ house __ ____.
___ ___ car __ _____.

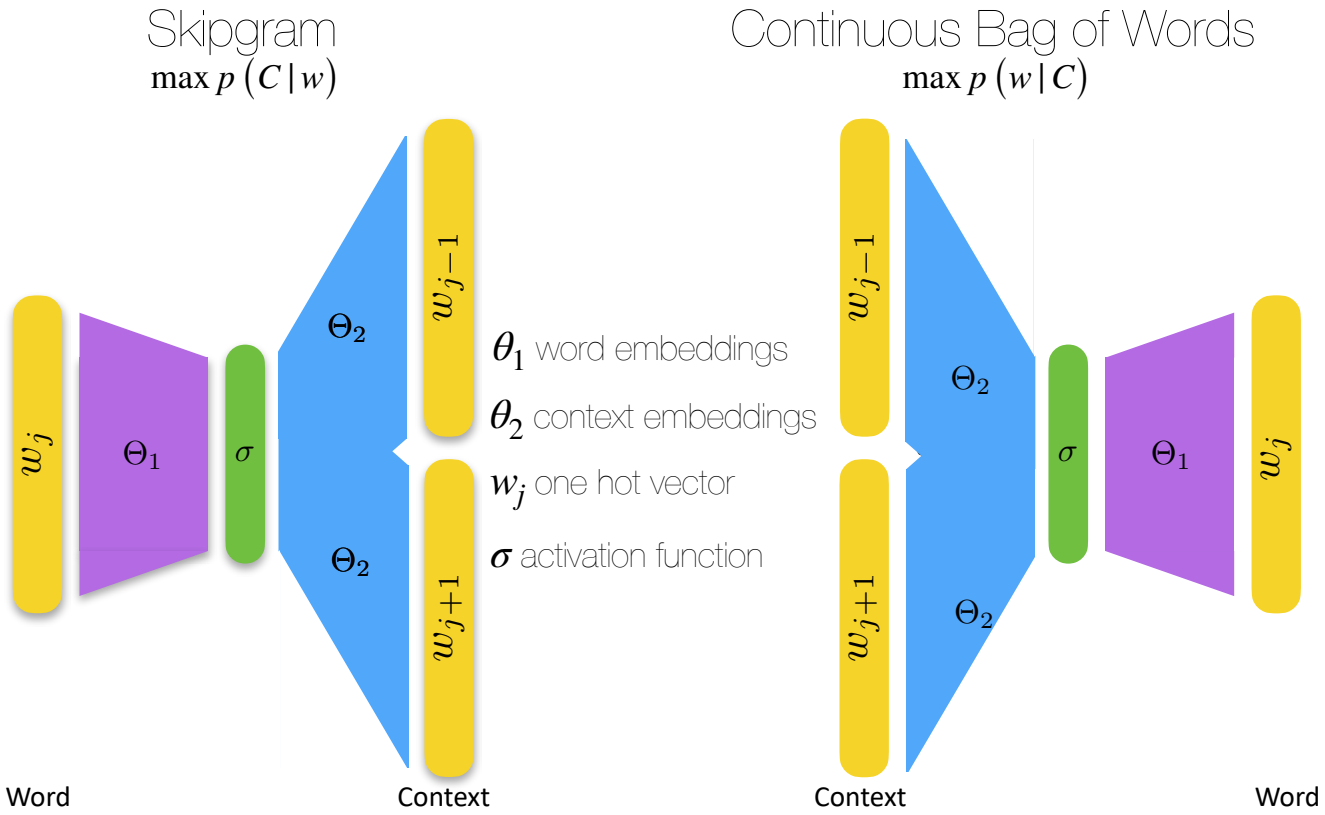
$$\max p\left(C \mid w\right)$$

- And from the context we have some idea about possible words.

The red _____ is beautiful.
The blue _____ is old.

$$\max p\left(w \mid C\right)$$

word2vec


Skipgram
$\max p(C \mid w)$
Continuous Bag of Words
$\max p(w \mid C)$
w_j
Θ_1
σ
Θ_2
Θ_2
w_{j-1}
w_{j+1}
θ_1 word embeddings
θ_2 context embeddings
w_j one hot vector
σ activation function
Word
Context
Context
Word

Variations

- Hierarchical Softmax:
  - Approximate the softmax using a binary tree
  - Reduces the number of calculations per training example from V to $\log_2 V$ and increases performance by orders of magnitude.

Variations

- Negative Sampling:
  - Under sample the most frequent words by removing them from the text before generating the contexts
  - Similar idea to removing stop-words — very frequent words are less informative.
  - Effectively makes the window larger, increasing the amount of information available for context

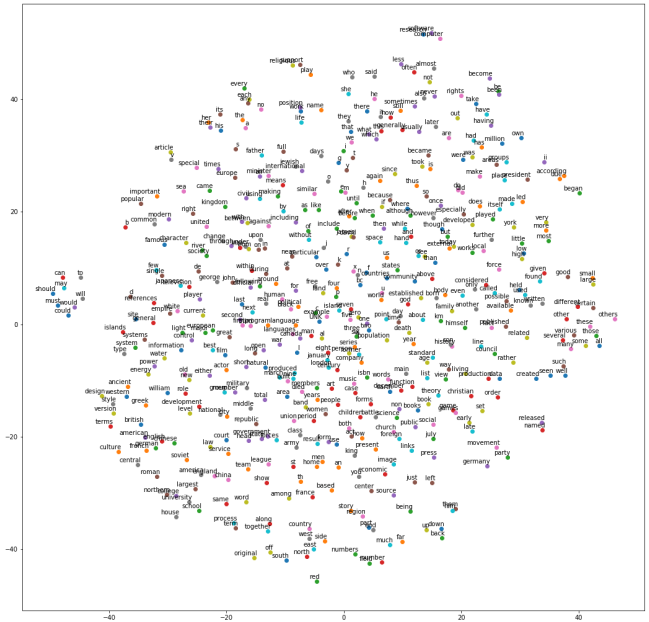
word2vec details

- The output of this neural network is deterministic:
  - If two words appear in the same context (“blue” vs “red”, for e.g.), they will have similar internal representations in θ_1 and θ_2
  - θ_1 and θ_2 are vector embeddings of the input words and the context words respectively
- Words that are too rare are also removed.
- The original implementation had a dynamic window size:
  - for each word in the corpus a window size k' is sampled uniformly between 1 and k

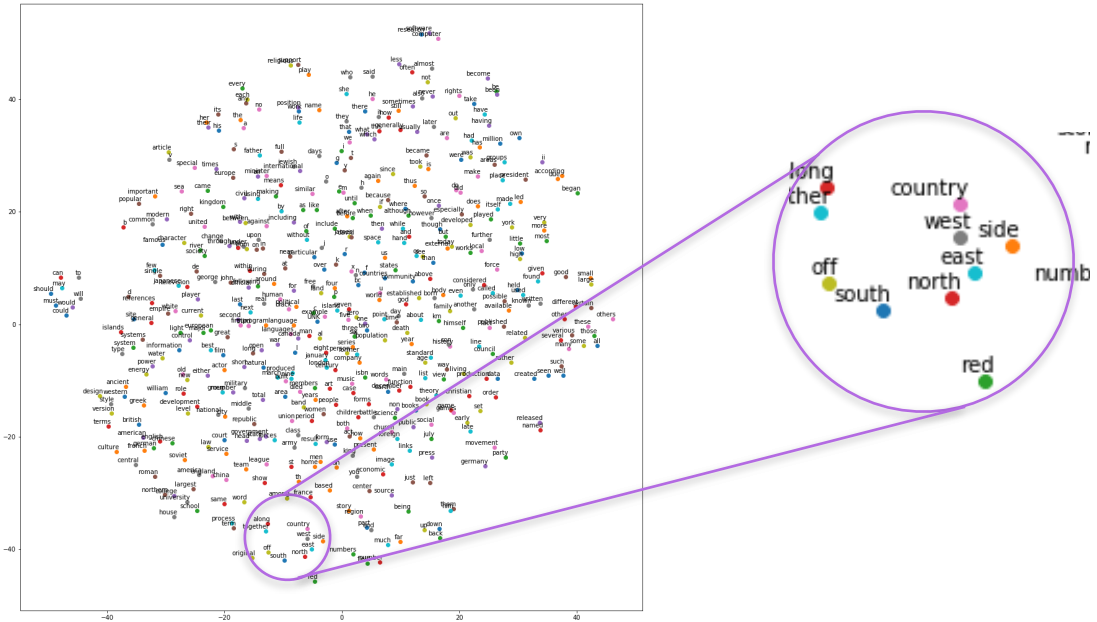
Online resources

- C - https://code.google.com/archive/p/word2vec/ (the original one)
- Python/tensorflow - https://www.tensorflow.org/tutorials/word2vec
- Python/gensim - https://radimrehurek.com/gensim/models/word2vec.html
- Pretrained embeddings:
  - 30+ languages, https://github.com/Kyubyong/wordvectors
  - 100+ languages trained using wikipedia: https://sites.google.com/site/rmyeid/projects/polyglot

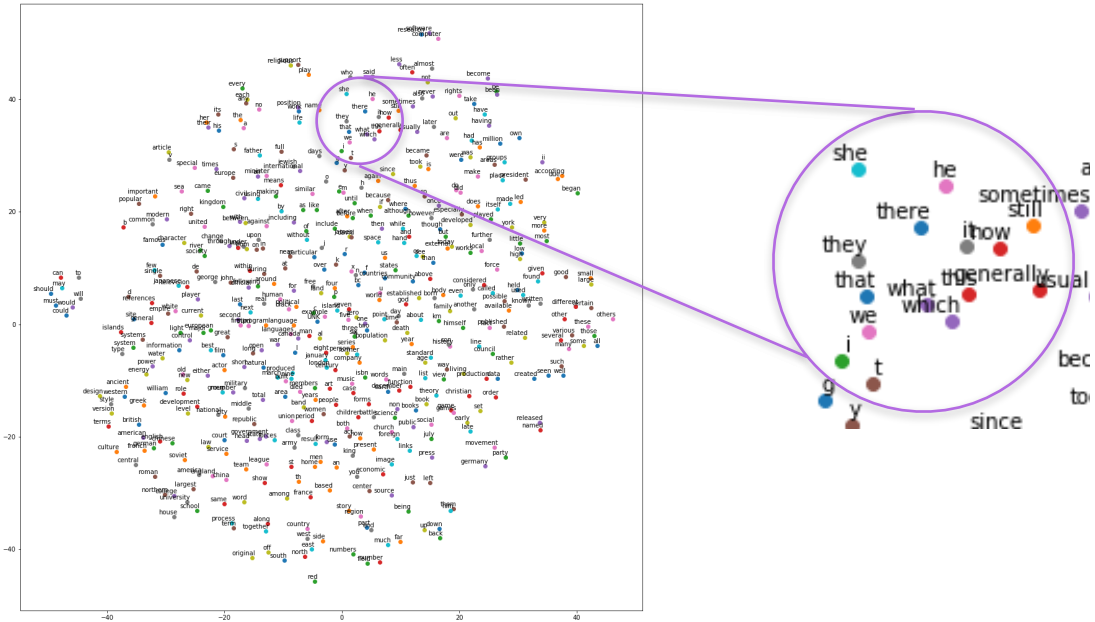
Visualization



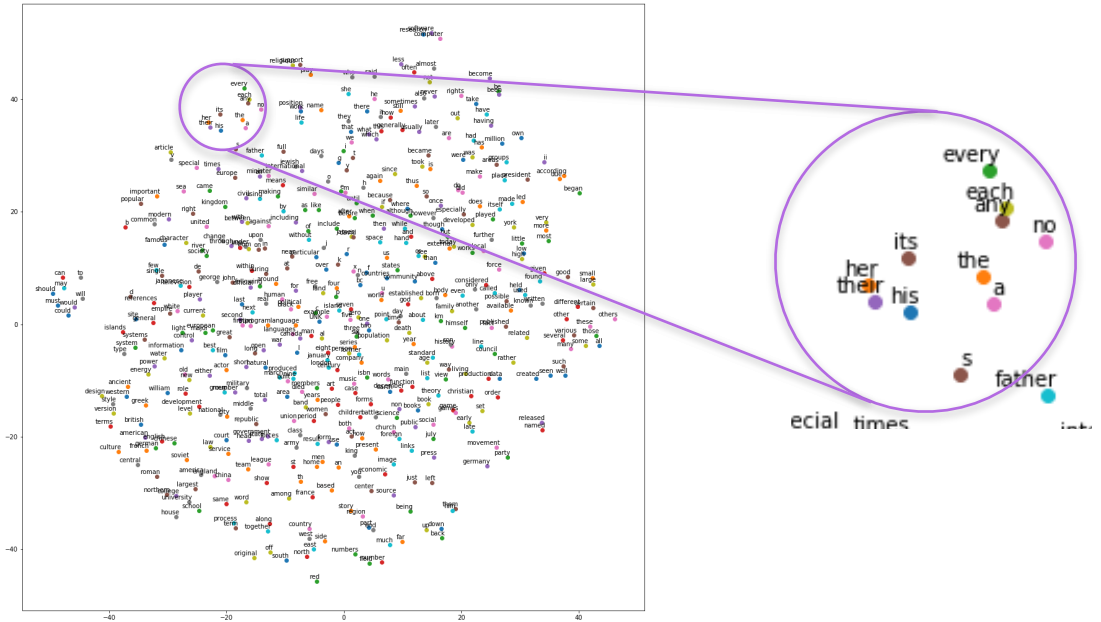
Visualization



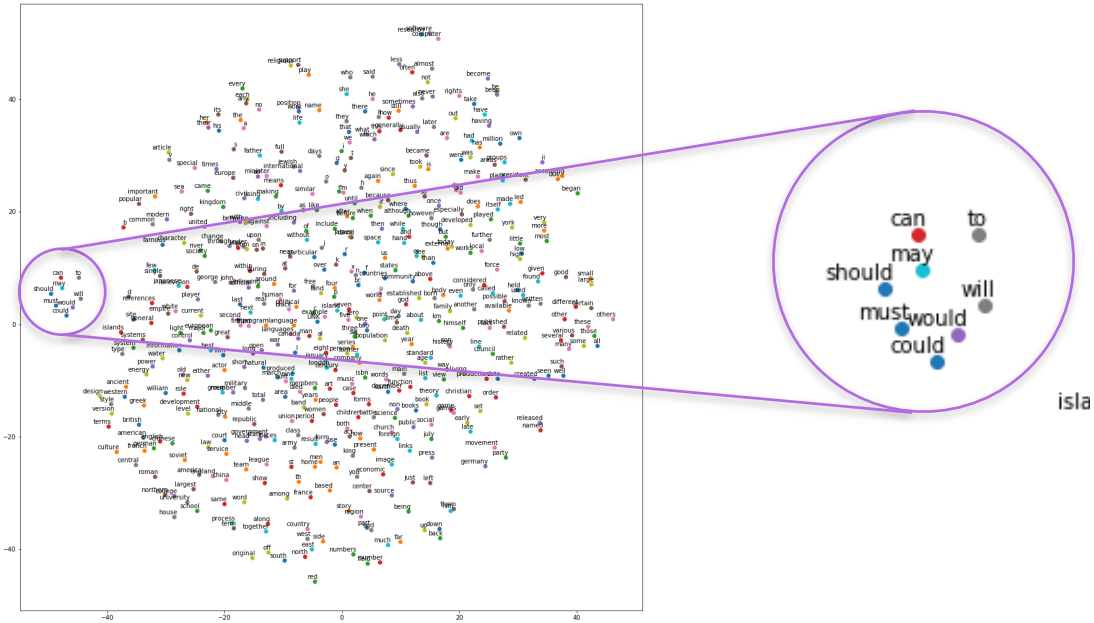
Visualization



Visualization



Visualization



Analogies

- The embedding of each word is a function of the context it appears in:

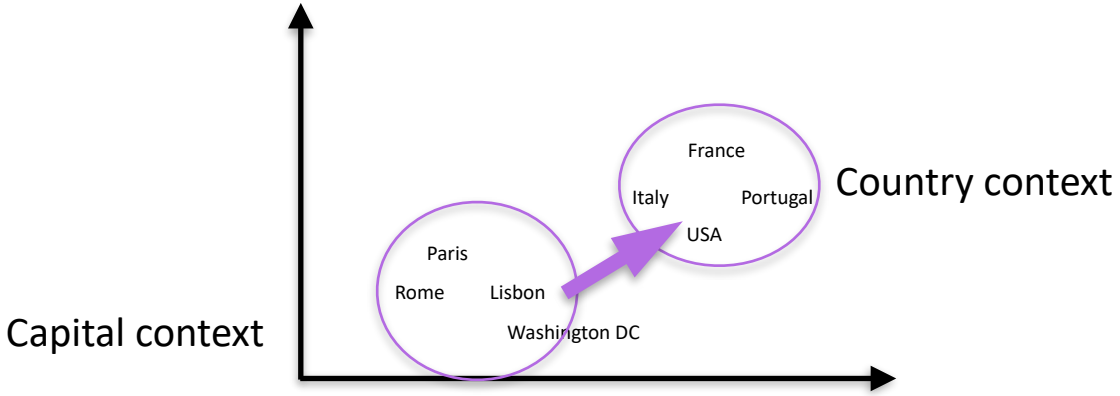
$$\sigma\left(red\right) = f\left(\text{context}\left(red\right)\right)$$

- words that appear in similar contexts will have similar embeddings:

$$\text{context}\left(red\right) \approx \text{context}\left(blue\right) \implies \sigma\left(red\right) \approx \sigma\left(blue\right)$$

- "Distributional hypotesis" in linguistics

Analogies



$$\sigma\,(France) - \sigma\,(Paris) = \sigma\left(Italy\right) - \sigma\,(Rome)$$

$$\sigma\,(France) - \sigma\,(Paris) + \sigma\,(Rome) = \sigma\left(Italy\right)$$

$$\overrightarrow{b} - \overrightarrow{a} + \overrightarrow{c} = \overrightarrow{d}$$

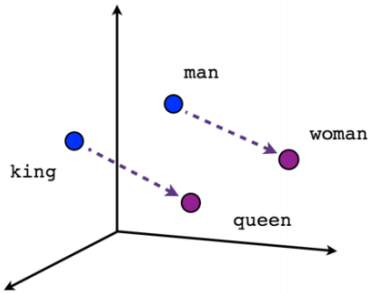
Analogies

$$\vec{b} - \vec{a} + \vec{c} = \vec{d}$$

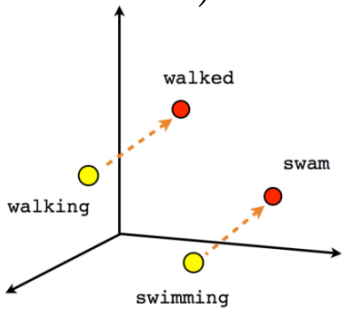
$$d^{\dagger} = \text{argmax}_x \frac{\left(\vec{b} - \vec{a} + \vec{c}\right)^T}{\left\| \vec{b} - \vec{a} + \vec{c} \right\|} \vec{x}$$

$$d^{\dagger} \sim \text{argmax}_x \left(\vec{b}^T \vec{x} - \vec{a}^T \vec{x} + \vec{c}^T \vec{x} \right)$$

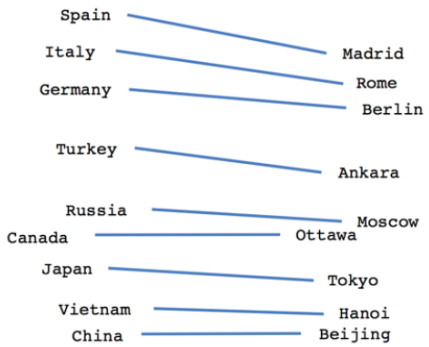
What is the word d that is most similar to b and c and most dissimilar to a?



Male-Female



Verb tense



Country-Capital

Lesson 8: Applications

8.1 - word2vec embeddings

8.2 - GloVe

8.3 - Transfer Learning

8.4 - Language Detection

Global Vectors (GloVe)

- An alternative to word2vec developed by the Stanford NLP Group in 2014
- Explicitly models word cooccurrences by a cooccurrence matrix X where rows correspond to input words and columns to context words
- X_{ij} is the number of times word i occurred with context word j
- Contexts are defined through a sliding window

Global Vectors (GloVe)

- Embedding vectors for word i is defined as:

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log\left(X_{ik}\right)$$

- Embedding vectors are found through an optimization procedure with a cost function of the form:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_k - \log\left(X_{ij}\right)\right)^2$$

$$f\left(X_{ij}\right) = \begin{cases} \left(\frac{X_{ij}}{X_{max}}\right) & \text{if } \mathrm{X_{ij}} < \mathrm{x_{max}} \\ 1 & \text{otherwise} \end{cases}$$

Global Vectors (GloVe)

- $f\left(X_{ij}\right)$ prevents common word pairs from skewing our cost function
- Explicitly considers the context in which each word appears
- Word-context matrix is large, but sparse
- Relatively fast to train
- Highlights the importance of relative frequency of word

Online resources

- The original implementation: https://nlp.stanford.edu/projects/glove/
- Python package: https://pypi.org/project/glove/
- Pre-trained vectors: https://github.com/stanfordnlp/GloVe

Lesson 8: Applications

8.1 - word2vec embeddings

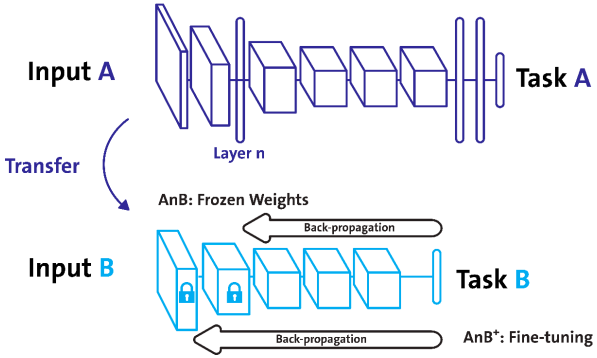
8.2 - GloVe

8.3 - Transfer Learning

8.4 - Language Detection

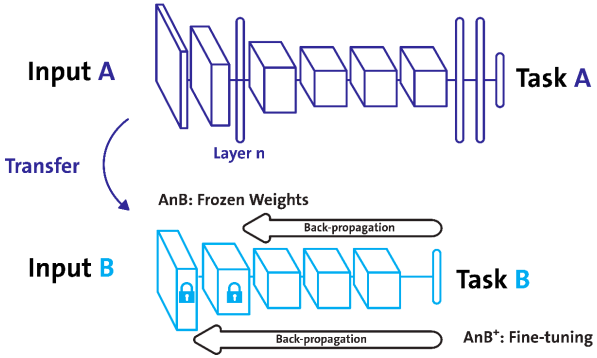
Transfer Learning

- Transfer Learning is the process of putting the knowledge learned by one network to use in another. Like when you make use concepts from a different field to solve a problem
- In a more general case, entire layers of a Deep Learning Network that was trained for Task A can be repurposed for use in Task B without any modifications
- This is particularly common in large scale systems that are extremely expensive (in both time and money) to train from scratch



Transfer Learning

- We can take advantage of the huge amounts of work put in by Google, Stanford, etc to generate high quality embeddings to save time and effort when developing our models
- In the case of small systems with relatively few training examples, specially trained embeddings tend to over perform these high quality ones.



Lesson 8: Applications

8.1 - word2vec embeddings

8.2 - GloVe

8.3 - Transfer Learning

8.4 - Language Detection

Language Detection

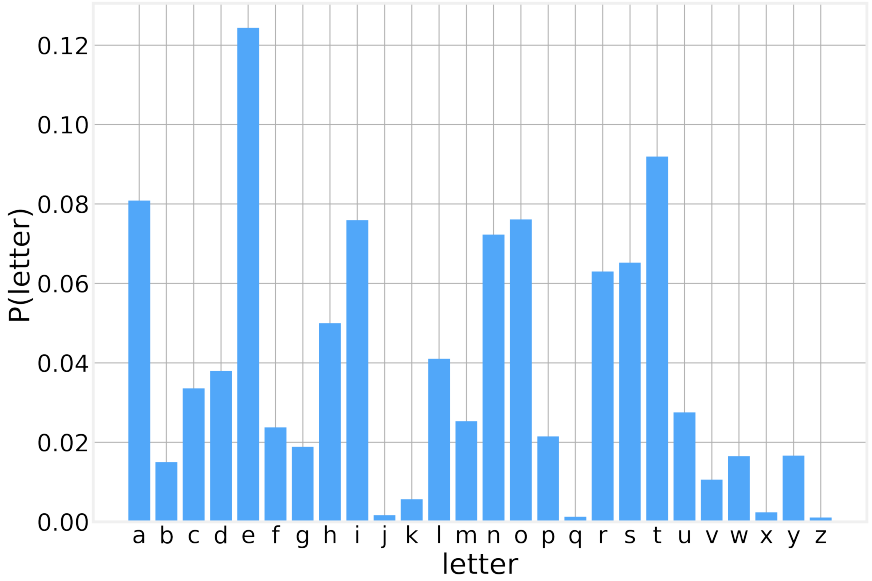
- Sometimes a given corpus will contain documents written in different languages (comment boxes in international websites, for example)
- We intrinsically assumed that each individual documents is comparable with all others.
- When multiple languages are present within the same corpus (on Twitter, in the comment boxes of an international website, etc) this is no longer true.

Language Detection

- One way to handle this would be to cluster documents, as documents in the same language will naturally cluster together. However:
  - there might be multiple clusters using the same language
  - We might not be interested in doing all this work for languages other than, say, English
- Language detection allows us to preprocess our corpus so that we can focus on specific languages, sort documents by language (to use different stopword lists), etc.
- Languages can be characterized by their character (letter) distribution.

Character Probabilities

- The character level distribution for English, obtained using Google Books 1-gram dataset is:



Language Detection

- We measured the probability distribution of letters in the english language. In effect, we calculated:

$$P\left(letter \mid english\right)$$

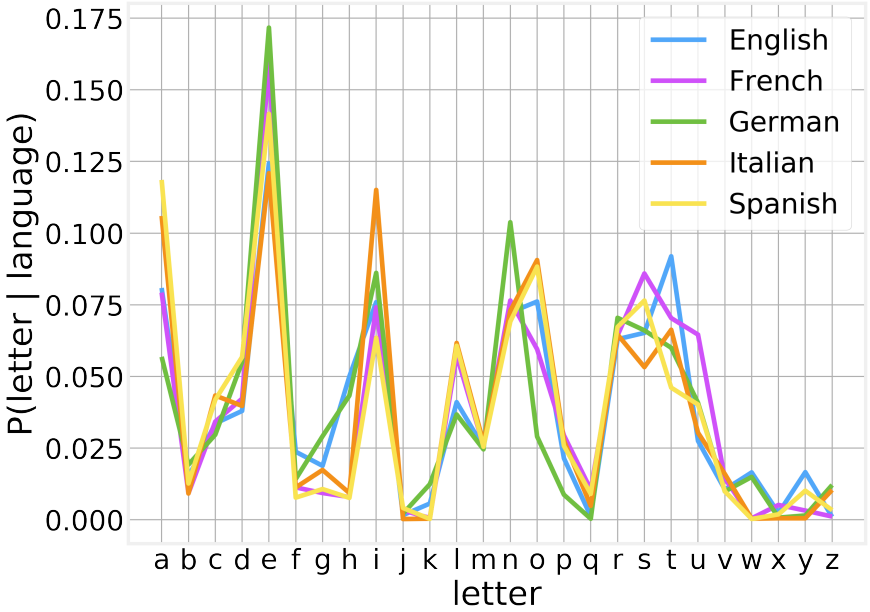
- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:

$$P\left(letter \mid language\right)$$

- Google Books covers several different languages, among which we find 5 different European languages: English, French, German, Italian and Spanish.

Language Detection

- Character distributions for different languages look different in at least a few of the characters due to the idiosyncrasies of each language, even in the case of closely related languages.



Conditional Probabilities

- Using these conditional probabilities, and Bayes Theorem, we can easily build a language detector. For that we just need to calculate:

$$P\left(language \mid text\right)$$

- Which we can rewrite as:

$$P\left(language \mid letter_1, letter_2, \cdots, letter_n\right)$$

- If we treat each letter independently, we obtain:

$$P\left(language \mid letter_1, letter_2, \cdots, letter_n\right) = \prod_i P\left(language \mid letter_i\right)$$

Conditional Probabilities

- This is known as the Naive Bayes Approach and is an obvious oversimplification: It completely ignores correlations present in the sequence of letters.
- All we have to do now is apply Bayes Theorem to our original table:

$$P\left(language \mid letter\right) = \frac{P\left(letter \mid language\right) P\left(language\right)}{P\left(letter\right)}$$

Naive Bayes

- And if we assume that all languages are equally probable (non-informative prior):

$$P\left(language\right) = \frac{1}{N_{langs}}$$

- Naive Bayes approaches (and many others) use terms of the form:

$$\prod_i P\left(A \mid B_i\right)$$

- which implies multiplying many small numbers. To avoid numerical complications, it is best to use, instead:

$$\sum_i \log P\left(A \mid B_i\right)$$

- Which is commonly referred to as the "Log-Likelihood".

Naive Bayes

- Our expression then becomes:

$$\mathscr{L}\left(language \mid letter_1, letter_2, \cdots, letter_n\right) = \sum_i \log\left[\frac{P\left(letter_i \mid language\right) P\left(language\right)}{P\left(letter_i\right)}\right]$$

- Or more simply:

$$\mathscr{L}\left(language \mid text\right) = \sum_i \log\left[\frac{P\left(letter_i \mid language\right) P\left(language\right)}{P\left(letter_i\right)}\right]$$

- And finally:

$$\mathscr{L}\left(language \mid text\right) = \sum_i \mathscr{L}\left(language \mid letter_i\right)$$

- Providing us with a quick and easy way to determine which language is more likely to be the correct one.

Code - Applications

https://github.com/DataForScience/NLP_LL

Thank you!

Bruno Gonçalves

www.data4sci.com/newsletter

https://github.com/DataForScience/NLP_LL