Analysis of Text Data FS20

Pascal Baumann
pascal.baumann@stud.hslu.ch

24. Juni 2024

For errors or improvement raise an issue or make a pull request on the github repository.

Contents

1 Introduction

Text analysis consist of a series of operations completed by one or more pieces of software on a sample of written human language, with the goal of extracting useful information.

Applications of text analysis include:

- Analysis
  - spell checkers
  - keyword extraction
  - authorship attribution
  - document retrieval
  - text classification
  - text mining
  - sentiment analysis
  - content-based recommendation
- Text Analysis and Generation
  - machine translation
  - automatic question answering
  - automatic summarisation
- Text Generation
  - database report generation
  - weather forecast generation
- Analysis, Generation and Interaction
  - dialogue systems
  - assistive technology for teaching or writing

Analysis of Text Data lies at the intersection of Machine Learning, Computational Linguistics and Human-Computer Interaction. Artificial Intelligence is applied to human language, while human speech data is a related problem which makes use of different technologies, like signal processing.

1.1 Basic Concepts in NLP

Machine Learning is a powerful tool in NLP, thus the same vocabulary is used. Most of the time, supervised learning is used for Text Analysis, where the labelled data is classified by human experts and called the reference or ground truth.

Significance addresses the key question if the difference between the two systems is really due to the fact that one is better than the other or if it can be explained by randomness. This is easier to compute when cross-validation is used, and paired t-tests can be used to compare two systems. Performance scores vary depending on the data set it is difficult to predict actual performance on a data set of a different nature than the test set, this is called the problem of portability.

1.2 Key Jargon from Linguistics

- letter
- syllable
- morpheme: a meaningful morphological unit of a language that cannot be further divided

- word
- phrase: a group of words standing together as a conceptual unit
- clause: a group of words in a sentence that contains a subject and a subject
- sentence:
  - simple sentence: one independent clause
  - compound sentence: at least two independent clauses
  - complex sentence: at least one independent clause and one or more dependent clauses
- text
- corpus: collection of text, pl. corpora
- utterance: an uninterrupted chain of spoken or written language
- lexeme: basic abstract unit of meaning that roughly corresponds to a set of forms taken by a single root word — for example run, runs, ran and running are forms of the same lexeme
- lemma: one form chosen by convention as the canonical form of a lexeme — go, goes, went, gone, going for the lexeme *go*

1.3 Utterance Decoding

Decode its propositional content or logical form first and then combine them. Draw inference to solve ambiguities, guess what is left unsaid and, in general, maximise the likelihood given the context.

1.4 Lexical Analysis

The goal of lexical analysis is to understand word forms.

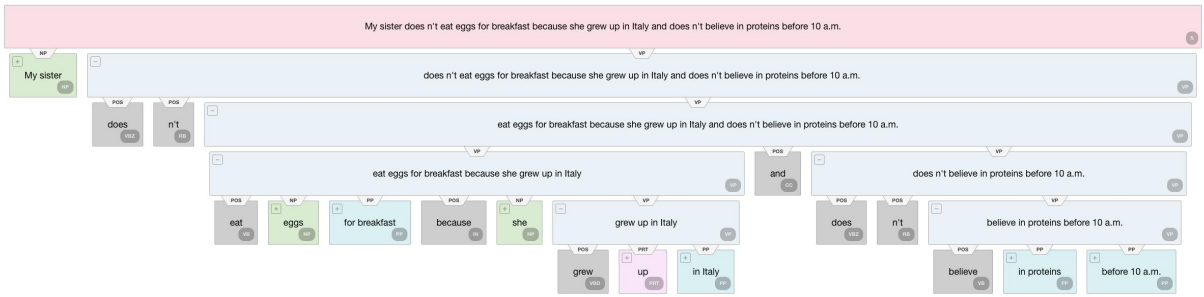
1. tokenisation: breaking a text into word tokens
2. lemmatisation: finding the base form of each word token or lemma
3. Part of Speech tagging: finding the part of speech each word token corresponds to
4. Named Entity Recognition: identification of proper nouns of people, places, organizations

1.5 Syntactic Analysis

- sentence segmentation or splitting
- identifying phrases or chunking
- figuring out logical functions
- building parse trees or parsing

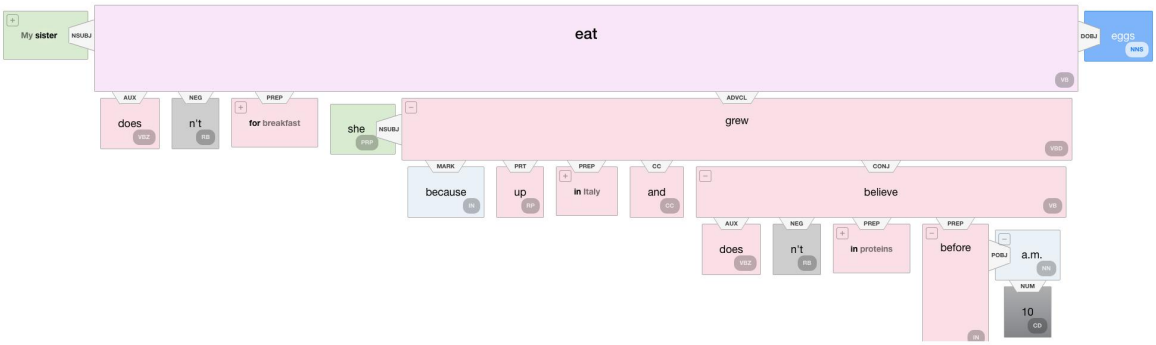
1.5.1 Constituency Parsing

With syntactic parsing the connection between words can be understood. Constituency parsing breaks a sentence into its basic building blocks.



1.5.2 Dependency Parsing

Dependency Parsing helps in understanding what depends on what. The assumption is that the sentence is centred around the verb and what comes first in a sentence is more important than any information that comes after. So anything of the sentence depends on that word or some lower-level information.



1.6 Understanding Meaning

1.6.1 Semantic Analysis

- word-level
  - word sense disambiguation
  - co-occurrence analysis
- sentence-level or text-level
  - semantic role labelling
  - co-reference resolution

1.6.2 Discourse Analysis

- topics
- sentiments
- speech or dialogue acts
- argumentative structures

1.7 Data for Text Analysis

Text analysis using machine learning requires large amounts of training data and finding suitable data is often a bottleneck due to expense or limited rights. An annotated corpus typically contains a selection of texts based on explicit criteria, metadata like author, date, source, title, sectioning, and annotations in a more or less standardised format.

1.8 Preprocessing

Most text processing tasks begin with a set of standard preprocessing steps

- Tokenisation: segmentation into tokens
- Token normalisation
- Segmentation into sentences

9 words, 8 unique ones, 10 tokens (including the period)

- There are about 170'000 unique words in the English language at the moment

1.8.1 Tokens and Types

Tokens are the words on the page, while the type is the word forms. Counting the types requires lemmatisation, which is finding the lemma or base form for each word.

1.8.2 Tokenisation and Normalisation

Not as straightforward as one may think

- punctuation
  - periods and commas appear within words or abbreviations
  - special tokens in mail addresses or tweets
  - apostrophes
- capital letters
- compound words
  - problem complicated without dashes
  - German words
  - proper names

1.8.3 Lemmatisation and Stemming

Stemming is the process of reducing inflected or derived words to their word stem, also called base or root form. Lemmatisation refers to the process of determining the lemma of a word based on its intended meaning. Lemmatisation is closely related to stemming, but a stemmer *operates without knowledge of the context of the word*. Lemmatisation, on the other hand, depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as the larger context surrounding that sentence.

1.8.4 Sentence Segmentation

Can be easier or more difficult depending on the source text formatting. If no particular information is available from the layout, punctuations and casing can be used. While question and exclamation marks are quite reliable indicator of sentences periods are not. A good approach is to try to combine Tokenisation and Sentence Splitting, focusing on full stops.

2 Text Classification

The goal of text classification is to assign text documents to one or more categories. There is a predefined set of classes, in which previously unseen documents are assigned to. If there are only two classes this is a binary classification problem.

To tackle this problem there are different strategies: (a) have hard-coded rules carefully crafted by an expert on the basis on combinations of words or other features, (b) through supervised machine learning with understanding building on semantic representation of texts and labels or (c) supervised machine learning without understanding where word-based features are derived from text and the relationship between features and labels from the training texts.

Stop-words are words like conjunctions or prepositions, which may not have meaning given the task at hand.

2.1 Formalising Text Classification

Input

- a set of documents D
- a fixed set N of classes C
- a training set of M hand-labelled documents $(d_1C_1), (d_2C_2), \dots (d_MC_M)$

Output

- a mapping $D \to C$ that associates a predicted class $c \in C$ to each document $d \in D$

2.2 Naïve Bayes Classifier

For a document d and a class c

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Maximum A Posteriori (MAP) classifier:

$$c_{\text{MAP}} = \underset{c \in C}{\text{argmax}} \frac{P(d|c)P(c)}{P(d)} = \underset{c \in C}{\text{argmax}} P(d|c)P(c)$$

Dropping the denominator does not change c_{MAP} because $P(d)$ has no effect on argmax. In practice, the evidence a machine can observe is not the human-readable document d, but a number of features $x_1, \dots, x_N$ obtained based on d and a MAP-classifier

$$c_MAP = \underset{c \in C}{\text{argmax}} P(x_1, x_2, \dots, x_N|c)P(c)$$

Given a vocabulary of V words a feature can be if a word appears in a document d or how often it appears

- **Bernoulli Model** represent d as $(e_1, \dots, e_V)$ where $e_i = 1$ if the word i is in d and $e_i = 0$ otherwise

- **Multinomial Model** represent d as $f_1, \ldots, f_V$) where f_i is the number of occurrences of word i in d

The Naïve Bayes independence assumption is that given a class c the **features are independent**

$$P(x_1, x_2, \ldots, x_N|c) = P(x_1|c) \cdot P(x_2|c) \cdots P(x_N|c) = \prod_{k=1}^{n} P(x_k|c)$$

$P(c)$ and $P(x_k|c)$ need to be computed, $P(c)$ can be estimated based on the frequency of each class in the training data, and $P(x_k|c)$ depends on the chosen feature representation.

In the so called Bag-Of-Word Model x_k is the feature representation of the words in the documents. The position of the individual words can usually be ignored.

2.3 Multinomial Naïve Bayes

Represent every token in d as a feature vector $x_i = f_i$, where f_i is the number of occurrences of token i in d. Then $P(x_i|c)$ can be estimated as $\frac{\text{number of occurrences of token } i \text{ in } c}{\text{total number of tokens in } c}$. For simplicity $P(w|c)$ can be written to refer to the probability of finding token w in class c.

1. Normalise the training data (remove stop words, remove punctuation, set all characters to lowercase)
2. Assemble vocabulary (list of unique meaningful words)
3. Count the number of occurrences of each word in each class and divide by the total number of words in each class

2.4 Practical Complications

If a word w from the vocabulary (which contains V tokens) is never in class c in the training data, we will estimate $P(w|c) = 0$, which causes $c_{\text{corpus}} = \underset{c \in C}{\text{argmax}} P(c) \prod_{k=1}^{N} P(x_k|c) = 0$.

Remark. *Zero probabilities cannot be conditioned away, no matter the other evidence!*
- Dan Jurafsky

The typical solution to this is to do Laplace Smoothing like this

$$P(w|c) \text{ can be estimated as } \frac{\text{number of occurrences of token } w \text{ in class } c + 1}{\text{total number of tokens in class } c + V}$$

This prevents probabilities of zero from occurring.

2.5 Metrics

True Positive	Sample c classified correctly
False Positive	Non-c sample incorrectly classified as c
True Negative	Non-c sample classified correctly
False Negative	Sample c classified incorrectly as non-c

Accuracy	$\frac{\sum TP}{\sum \text{all elements}}$
Precision	$\frac{TP}{TP+FP}$

Recall $\frac{TP}{TP+FN}$

3 Sentiment Analysis

Sentiment analysis tries to understand the emotional state of the author of a text, and is also called opinion mining but can be extremely hard. In principle text classification and sentiment analysis are very different, but there are some striking similarities as the text can reflect a positive or a negative emotional state which can be utilised.

3.1 Framing Sentiment Analysis

Sentiment analysis is commonly framed as **attitude detection**

- *Attitude* is an enduring disposition toward someone or something with an emotional connotation
- The attitude can be broken down in quintuplets
  1. opinion holder
  2. target entity
  3. aspect: specific feature of the target that the opinion is about
  4. type: most commonly positive, negative, neutral and with an indication of strength
  5. time when this opinion was expressed
- knowing the opinion without knowing the opinion target is of limited use

3.2 Tokenisation Challenges

- Isolating emoticons
- Respecting domain-specific markup like `<strong>really bad idea</strong>`
- Capturing masked curses
- Selective true-casing
- Normalised lengthening, as "YAAAAAAY" is equivalent to "YAY" no matter how it is written
- Capturing important multi-word expressions and idioms

3.2.1 Dealing With Negations

A simple effective workaround by Das and Chen is to prepend `NOT_` to every word between a negation and a clause-level punctuation mark.

3.2.2 More Complications

- Modal adverbs like *quite possibly, totally*
- Thwarted expectations "*It was hailed as brilliant, unprecedented artistic achievement worthy of multiple Oscars*"
- Non-literal language "*Like 50 hours long*"

3.3 Various Degrees of Complexity

In principle, all quintuplets in a document should be discovered, which would allow representing unstructured information in a structured form. However, this is extremely hard to do. In practice, simplified forms of sentiment analysis are typically solved

- Document-level sentiment classification
- Sentence-level sentiment classification
- Aspect-level sentiment classification

Though simplified, these forms of sentiment analysis are still hard to do with subtlety and sarcasm complicating the task.

3.3.1 Document-level Sentiment Classification

Does the document reflect an overall positive or negative view? The approach is the same as for text classification

1. tokenisation
2. feature extraction
3. classification (Naïve Bayes, MaxEnt, SVM)
4. Two or Three classes { positive | negative | (neutral) }

3.4 Generative versus Discriminative Models

In Naïve Bayes, for a document d and each class c:

- Estimate $P(d|c)$ and $P(c)$ directly from the training data
- Learn a model of $P(d \cap c) = P(d|c) \cdot P(c)$
- Use Bayes' rule to compute $P(c|d) = \dfrac{P(d|c) \cdot P(c)}{P(d)}$
- Pick the class that maximises $P(c|d)$ over all classes

This is called a **generative model**: Which class c is most likely to have generated my test sample d, given the $P(d \cap c)$ that was estimated.
The alternative is a **discriminative model** that models $P(c|d)$ directly.

3.5 Discriminative Model: Maximum Entropy

Entropy in information theory refers to a measure of the information content, where highly probable events carry very little information and if all events are equally likely there is maximum entropy. A prediction on which one will occur is impossible and when a event occurs there is a lot of learning.

- Suppose movie reviews get classified into { positive | negative | (neutral) }
- In the training corpus 90% of the sample containing the word `love` are tagged as positive
- Let it further be that a test sample with the word `love` is 90% likely to be positive and equally likely to be either negative (10%) or neutral (10%)
- All other things unknown are assumed to be equally likely (maximum entropy)

Unlike in text classification, topic words are not as important as **sentiment words** that carry an emotional charge, but the mapping between emotional states and the written word can be exceedingly subtle.

- good, splendid, nice
- bad, racist, dismal

3.6 Unsupervised Sentiment Classification

1. Use a Part-Of-Speech tagger to identify phrases with adjectives and adverbs
2. Estimate the semantic orientation
3. Classify based on the average semantic orientation

Semantic orientation is computed based on Pointwise Mutual Information (PMI)

$$\text{PMI}(w_1, w_2) = \log_2\left(\frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right) = \log_2\left(\frac{P(w_1|w_2)}{P(w_1)}\right) = \log_2\left(\frac{P(w_2|w_1)}{P(w2)}\right)$$

PMI is computed using approximate probabilities with normalised word counts in a corpus. Or the web can be used as a corpus using the number of hits from a search engine when w_1 and w_2 are searched. Following Barrière (2016) the number of hits of `the, a, of` are used, which results in a total of $2.527 \cdot 10^{10}$ hits.

3.6.1 Example PMI of Academy and Awards

- $w_1 =$ `Academy`, $w_2 =$ `Awards`
- $P(w_1, w_2) \propto \#$`Academy` AND `Awards` $\approx 1.01 \cdot 10^9$hits
- $P(w_1) \propto \#$`Academy` $\approx 1.94 \cdot 10^9$hits
- $P(w_2) \propto \#$`Awards` $\approx 5.57 \cdot 10^9$hits
- $\text{PMI}(w_1, w_2) \approx \log_2\left(\frac{1.01 \cdot 10^9 \cdot 2.527 \cdot 10^{10}}{1.94 \cdot 10^9 \cdot 5.57 \cdot 10^9}\right) \approx 1.24$

Two words with high PMI are very likely to occur jointly, while two words with low PMI are very likely to occur separately. PMI can also be used to compute Semantic Orientation (SO), by computing the PMI of selected bigrams with reference words.

4 Representation and Querying

The user has information needs and expresses them as a text query, the system matches the query with the documents and returns the results. The system is using system-specific document or query representations to make comparison possible. The problem is, that computers have difficulties to work with text directly, so the text has to be transformed into something the machine can work with, usually a vector of numbers.

4.1 Document Representation

The idea is to count the words that are shared between documents. The higher the score, the more words are shared, the higher the apparent similarity between documents. But lexemes and capitalisation makes this task harder.

Figure 1: Preprocessing pipeline for text

The actual steps taken in the preprocessing depends heavily on the use case.

4.1.1 Clean Text

- Remove markup
- Expand contractions
- TrueCase
- Remove stop-words
- Only keep specific word forms

4.1.2 Tokenise

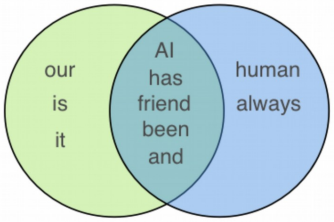
- Tokenise
- Lemmatisation
- Stemming
  Reduces inflected words to their stem, the root or base forms, even if the stem itself is not a valid word in the language

4.2 Scoring with Jaccard Similarity

The idea to count the words that are shared between documents to account for similarity stays the same. But the Jaccard Similarity is defined as the size of intersection divided by the size of the union of two sets.

Sentence 1: AI is our friend and it has been friendly
Sentence 2: AI and humans have always been friendly

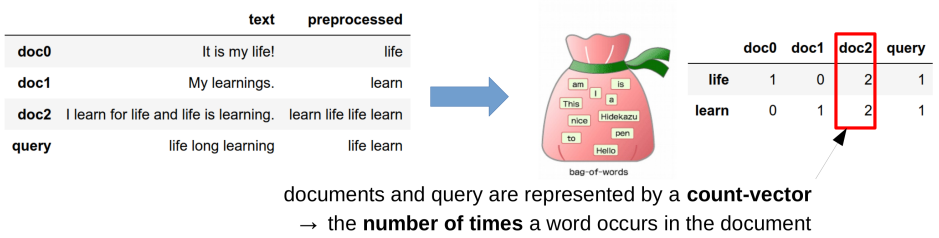
$$J(\text{document}_1, \text{document}_2) = \frac{\text{document}_1 \cap \text{document}_2}{\text{document}_1 \cup \text{document}_2}$$



Jaccard similarity has the issue of preferring short documents and that it ignores the number of times a word is occurring.

4.3 Bag of Words Model

Focuses on the number of occurrence of words, with the intuition that if a queried word occurs more in the document it is more relevant.



However the Bag of Words model does not consider the ordering of words in a document, and the words must precisely match.

4.4 Term Frequency Weighting and Transformation

The raw term frequency is not necessarily the most desirable attribute. A document where a query word occurs more should be handled as more relevant, but a document with 50 occurrences should not be treated 50 times more important than a document with only five occurrences. Thus the relevance should increase sublinear with term frequency

$$\mathrm{TF}_{\mathrm{document}} = \sum_{\mathrm{word} \in \mathrm{query} \cap \mathrm{document}} \log\left(1 + \mathrm{count}(\mathrm{word}, \mathrm{document})\right)$$

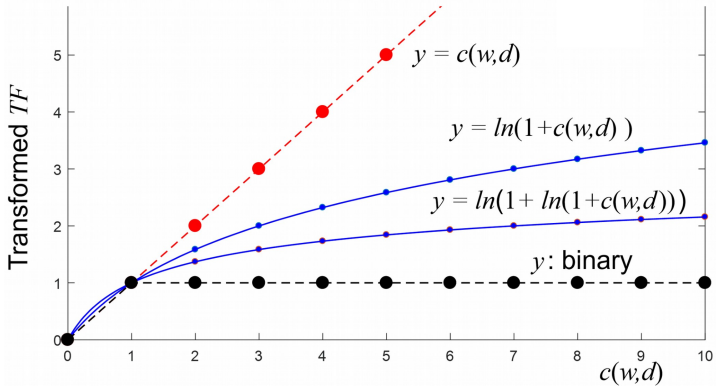


Figure 2: A sublinear transformation avoids the dominance of single words

4.4.1 (Best Matching) BM25 Term Frequency Transformation

$$y = \frac{(k+1) \cdot c(w,d)}{k + c(w,d)}$$

4.5 Inverse Document Frequency

The problem with term frequency lies in the fact that highly frequent words dominate the analysis, but rare and domain specific words may be more informative semantically. Thus positive weights for frequent words but lower weights than for rare words are desirable.

DF_w: Document Frequency, the number of documents containing word w, is the inverse measure of the informativeness of word w

N: total number of documents in the collection

$$\mathrm{IDF}_{word} = \log\left(\frac{N}{\mathrm{DF}_{word}}\right)$$

The logarithm is used to dampen the effect of IDF.

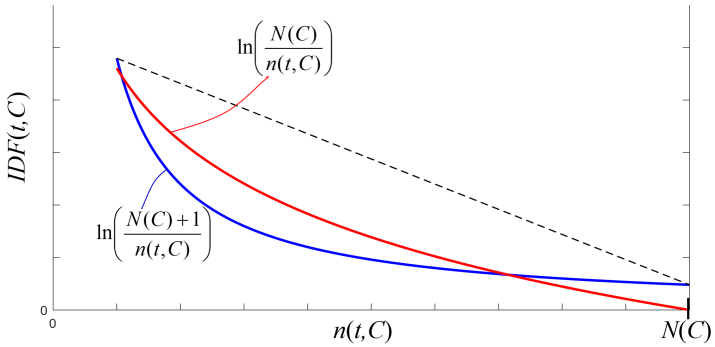


Figure 3: Popular terms are penalised

4.6 TF-IDF

$$\text{TF-IDF}_{word} = \text{TF}_{word,doc} \cdot \text{IDF}_{word,coll}$$

Increases with the number of word occurrences within a document and the rarity of the word in the collection. Ranking of documents for a query is done by

$$\text{Score}_{query,doc} = \sum_{word \in query \cap doc} \text{TF-IDF}_{word,doc,coll}$$

	TF weights
Binary	$0/1$
Counts	$c(t,d)$
Frequency	$\frac{c(t,d)}{\sum_{\tau \in d} c(\tau,d)}$
Log Counts	$1 + \log\left(c(t,d)\right)$
$\vdots$	$\vdots$

	IDF weights
Unary	1
IDF	$\log\left(\frac{N(C)}{n(t,C)}\right)$
IDF. *	$1 + \log\left(\frac{N(C)}{n(t,C)}\right)$
IDF Smoothed	$\log\left(\frac{N(C)}{1+n(t,C)}\right)$
$\vdots$	$\vdots$

4.7 Document Length Normalisation

Penalise long documents with a document length normaliser, as long documents have a better chance to match any query and are thus less specific. But avoid over-penalisation as documents can be long since they use more words, which should result in more penalisation, or as they have more content, which should be less penalised.

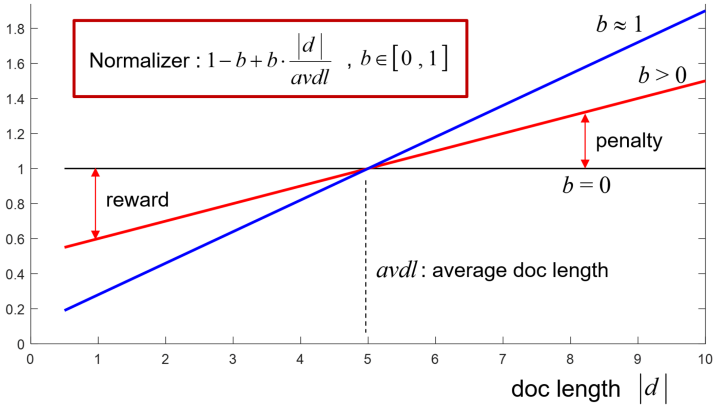


Figure 4: Average document length is used as a pivot point

4.7.1 State of The Art Okapi BM25+

Given a Query Q containing keywords $q_1, \ldots, q_n$, document D's score is

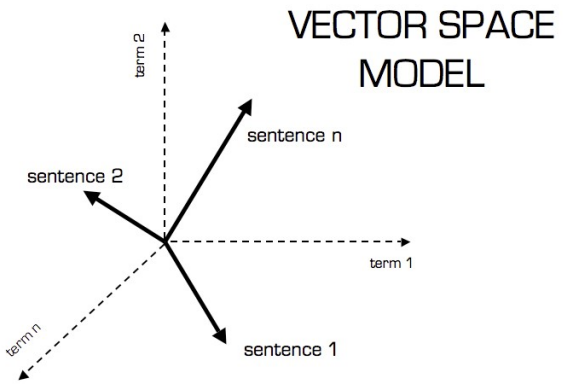
$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \left[\frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)} + \delta \right]$$

$$\text{IDF}(q_i) = \log \frac{N-n(q_i)+0.5}{n(q_i)+0.5}$$

$f(q_i,D)$	Term frequency of q_i in document D
$\lvert D\rvert$	length of document D
avgdl	average document length over corpus
δ	free parameter, default $\delta = 1.0$
k_1	free parameter, $k_1 \in [1.2, 2.0]$
b	free parameter, default $b = 0.75$
N	number of documents in corpus
$n(q_i)$	number of documents containing q_i

4.8 Vector Space Model

Used to correlate words, sentences and documents by semantic similarity.



For scoring the cosine similarity between documents and a query vector is preferred over the euclidean distance

similar scores	score vectors are in the same direction, angle between them is almost zero making the cosine lying near one or 100%
unrelated scores	score vectors are nearly orthogonal and the cosine is thus near zero
opposite scores	score vectors in opposite directions, cosine is near minus one or -100%

The cosine similarity is calculated by

$$\cos(d_j, q) = \frac{\boldsymbol{d_j} \cdot \boldsymbol{q}}{\|\boldsymbol{d_j}\|\|\boldsymbol{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

L^2 norm (Euclidean norm) is the square root of the dot product of the vector $\boldsymbol{p}$ with itself, and scales a vector to its unit-length

$$\|\boldsymbol{p}\| = \sqrt{p_1^2 + p_2^2 + \cdots + p_n^2} = \sqrt{\boldsymbol{p} \cdot \boldsymbol{p}}$$

Normalising all vectors to unit vectors this way allows for the dot-product to measure similarity, and do it in a more efficient way than cosine similarity. The dot product between two vectors $\boldsymbol{d_j}$ and $\boldsymbol{q}$ is high if w_{ij} and w_{iq} have similar characteristics, and is equivalent to the cosine similarity if the vectors are normalised.

5 Word Embeddings

An embedding is a **representation of words as dense vectors** such that the properties of the words and the relationships between the words are preserved. **Count-based methods** compute statistics of word co-occurrence in large text corpora, and then map these count-statistics down to a small, dense vector for each word. Predictive models use a (shallow) neural network to predict words from its neighbours. The learned weights act as word vector representations.

Word embeddings are automatically learned and allow for vector-oriented reasoning based on the offsets between words.

5.1 Context-Counting Word Vectors

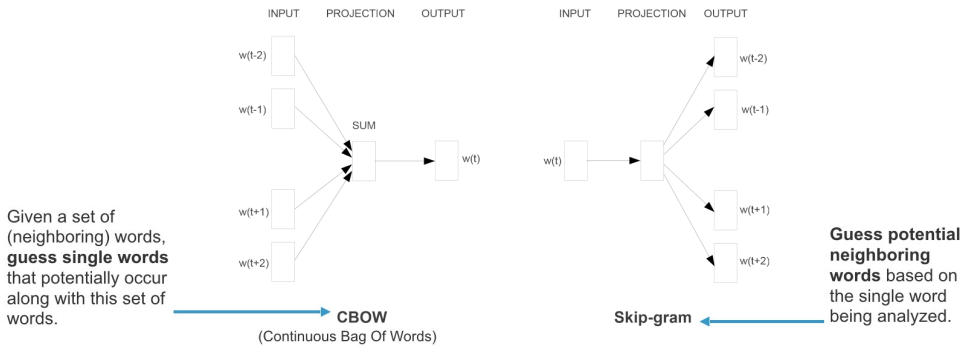
Represent mutual information of a word with other words by global co-occurrence counts using a pre-defined window size.

	START ___ was	it ___ the	was ___ best	the ___ of	best ___ times	of ___ it	times ___ was	was ___ worst	worst ___ times	of ___ END
it	1	0	0	0	0	0	1	0	0	0
was	0	2	0	0	0	0	0	0	0	0
the	0	0	1	0	0	0	0	1	0	0
best	0	0	0	1	0	0	0	0	0	0
of	0	0	0	0	1	0	0	0	1	0
times	0	0	0	1	0	0	0	0	0	1
worst	0	0	0	1	0	0	0	0	0	0

The problem is that this matrix increases with vocabulary and windows size, which results in a very high dimensional, but sparse feature matrix. Single value decomposition (SVD) is used to project the co-occurrence matrix to a smaller dimension and this sub-matrix is taken as the word embedding matrix.

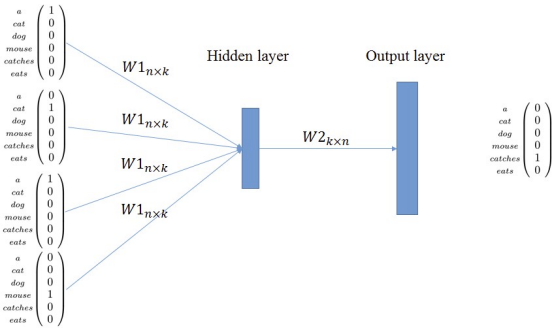
5.2 Word2Vec

Word2Vec is a simple and computationally efficient way to learn embeddings. It is a prediction based method and comes in two flavours, the Continuous Bag of Words (CBOW) and the Skip-gram model.



5.2.1 CBOW

Given a set of context words, predict the centre word that potentially occurs with the context words. The input are m one-hot vectors of context window words.



5.2.2 Skip-Gram

Given a center word, predict context words that potentially occur with the centre word. Input is the one-hot vector for word w.

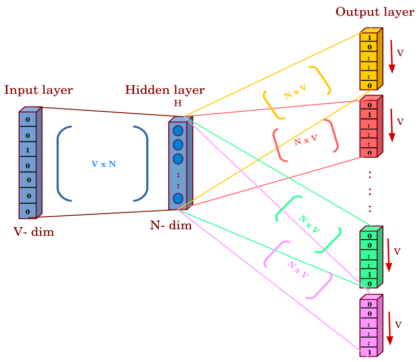
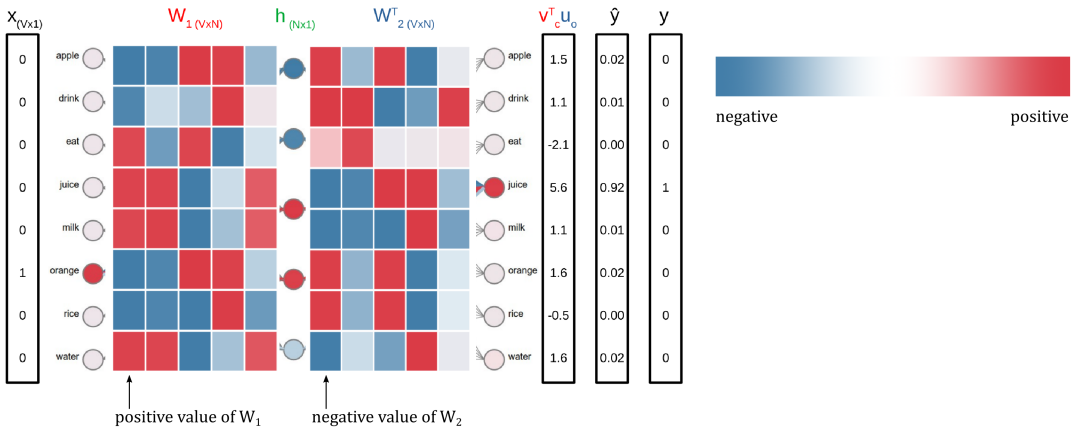


Figure 5: Source: An Information Retrieval and Recommendation System for Astronomical Observatories



5.2.3 Softmax

$$\hat{y} = p(o|c) = \frac{e^{v_c^T u_0}}{\sum_{w \in V} e^{v_c^T u_w}}$$

Inner product $v_c^T u_0$

- v_c^T := embedding of word w if word is center word
- u_0 := embedding of word w if word is contex word
- Measures the similarity of the two embedding representations of v_c^T and u_0

The exponentiation makes all values positive and large values even larger, while the division by the sum over all words in the vocabulary makes $\hat{y}$ a probability.

5.3 GloVe

Combines count based co-occurrence statistics with predictive context window method. GloVe is faster to train than Word2Vec and scales to huge corpora.

5.3.1 Out of Vocabulary Words

While using Word2Vec or GloVe out-of-vocabulary words will be encountered, as the possible vocabulary is infinitely large. Subword methods are based on the assumption that the semantic representation can be reconstructed from known parts of an unknown word.

FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. Each word is represented with a bag of n-grams in addition to the word itself. It has proven to be more accurate than Word2Vec, but models are quite big and require a lot of computation for training.

5.4 Practical Advice

Use pre-trained word embeddings, unless there is a corpus of billions of words in the training set, where it is appropriate to start randomised. If your training set is small do not fine-tune the word embeddings. If the training set is large (more than one million words or with domain specific content), it probably will work better when fine-tuned to the task. It is generally advisable to compare results for both methods.

6 Word Sense Disambiguation

6.1 Homonyms

Homographs	words written the same way
Homophones	words that sound the same

In general Homographs cause problems with machine translation, while Homophones cause problems with text-to-speech. Words exhibiting these characteristics are called polysemous; words with multiple related senses.

6.1.1 Zeugma Test

Apply a word to two others in different senses and test if it makes sense.

"Routers are used to cut channels in wood and to connect different networks."

6.2 Synonyms and Antonyms

Synonyms are lexical items that have the same meaning in some or all contexts. Perfect synonymy seldom occurs, one or a few lexemes are typically preferable to other synonyms, and may convey the intended meaning better. This could be notions of politeness, register, acceptability, convention or nuance.

For example *automobile* is more formal than *car*.

Antonyms are words with opposite senses

Reversives	rise/fall, in/out, up/down
Opposite ends of a scale	long/short, hot/cold

6.3 Hyponymy and Hypernymy

One sense is a hyponym of another if it denotes a subclass. Sense A is a hyponym of sense B if being an A entails being a B.

- A:B = hyponym:hypernym = subtype:supertype
- Also known as the IS-A relationship

Hypernymy is the opposite relationship.

6.4 Meronymy and Holonymy

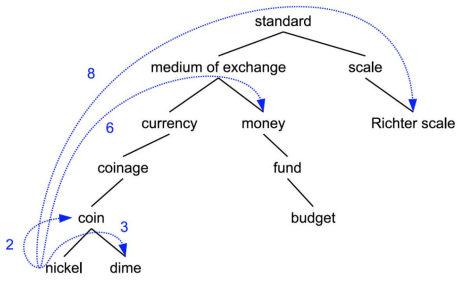
Part-whole and whole-part relationship. A meronym is a part of its holonym, for example a *wheel* is a meronym of *car*. There is a HAS-A relationship between a holonym and its meronym.

6.5 Word Similarity Based on WordNet

Look at the shortest path between sense nodes.

Example: path-based similarity

$\text{simpath}(c_1,c_2) = 1/\text{pathlen}(c_1,c_2)$



simpath(*nickel,coin*) = 1/2 = .5
simpath(*fund,budget*) = 1/2 = .5
simpath(*nickel,currency*) = 1/4 = .25
simpath(*nickel,money*) = 1/6 = .17
simpath(*coinage,Richter scale*) = 1/6 = .17

6.6 Word Vectors and Word Meaning

- One-hot vectors: No notion of word meaning
- Co-occurence matrices: words are represented by the weighted occurrence of other words in their context

- Pointwise mutual information (PMI) matrices: each element (i, j) signifies whether words i and j are more likely to occur jointly than to be independent
  - Negative PMI tends to be unreliable, as words co-occur less than expected with a corpus too small
  - In practice positive PMI (PPMI) is used

$$\text{PPMI}(w_1, w_2) = \max\left(\log_2\left(\frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)\right)$$

6.7 Word Sense Disambiguation

Extrinsic evaluation

- Embed WSD in another task
- Measure the task performance without and with WSD and see whether it improves

Intrinsic evaluation

- Use a labeled corpus
- Measure the accuracy, evaluated based on held-out data from the labelled corpus

As a baseline choose the most frequent sense or use the Lesk Algorithm.

6.7.1 Simplified Lesk Algorithm

The principle is to choose sense with most word overlap between gloss and context. Disambiguate the word bank in the sentence:

"The bank can guarantee that deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities."

The bank can guarantee that deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

6.7.2 Word Sense Disambiguation Based on Word Embeddings

Embedding vectors like Word2Vec or GloVe can be used for word sense disambiguation.

- Capture context of each word sense
  - Find the gloss of the word sense in WordNet
  - Get the embedding vectors of the words in the gloss
  - Average out the embedding vectors and obtain an embedding vector for the context of each target word sense
- Do the same for the sentence containing the target word

- Compare the embedding vector for the target sentence to the embedding vectors for the target word senses
- Pick the word sense whose embedding vector is the most similar to the one of the target sentence

```
for sense in range (1, len(word_vectors)):
        print ('context '+ str(sense) + ': ' + get_gloss(target_word, sense))
```

```
context 1: sloping land (especially the slope beside a body of water)
context 2: a financial institution that accepts deposits and channels the
↪  money into lending activities
context 3: a long ridge or pile
context 4: an arrangement of similar objects in a row or in tiers
context 5: a supply or stock held in reserve for future use (especially i n
↪  emergencies)
context 6: the funds held by a gambling house or the dealer in some gambl
↪  ing games
context 7: a slope in the turn of a road or track; the outside is higher
↪  than the inside in order to reduce the effects of centrifugal force
context 8: a container (usually with a slot in the top) for keeping money at
↪  home
context 9: a building in which the business of banking transacted
context 10: a flight maneuver; aircraft tips laterally about its longitud
↪  inal axis (especially in turning)
```

7 Part-of-Speech Tagging

PoS tagging means identifying the grammatical category of each word token.

- nouns
- verbs
- adjectives
- adverbs
- preposition
- particles
- conjunctions
- pronouns
- numerals
- interjections

7.1 Corpora

There are corpora with manually tagged PoS, which are both in part in NLTK

- Brown corpus
- Penn Treebank

The Penn Treebank tags are widely used for English language PoS tagging.

Number	Tag	Description	Number	Tag	Description
1	CC	Coordinating conjunction	20	RB	Adverb
2	CD	Cardinal number	21	RBR	Adverb, comparative
3	DT	Determiner	22	RBS	Adverb, superlative
4	EX	Existential there	23	RP	Particle
5	FW	Foreign word	24	SYM	Symbol
6	IN	Preposition or subordinating conjunction	25	TO	to
			26	UH	Interjection
			27	VB	Verb, base form
7	JJ	Adjective	28	VBD	Verb, past tense
8	JJR	Adjective, comparative	29	VBG	Verb, gerund or present participle
9	JJS	Adjective, superlative	30	VBN	Verb, past participle
10	LS	List item marker	31	VBP	Verb, non-3rd person singular present
11	MD	Modal	32	VBZ	Verb, 3rd person singular present
12	NN	Noun, singular or mass	33	WDT	Wh-determiner
13	NNS	Noun, plural	34	WP	Wh-pronoun
14	NNP	Proper noun, singular	35	WP$	Possessive wh-pronoun
15	NNPS	Proper noun, plural	36	WRB	Wh-adverb
16	PDT	Predeterminer			
17	POS	Possessive ending			
18	PRP	Personal pronoun			
19	PRP$	Possessive pronoun			

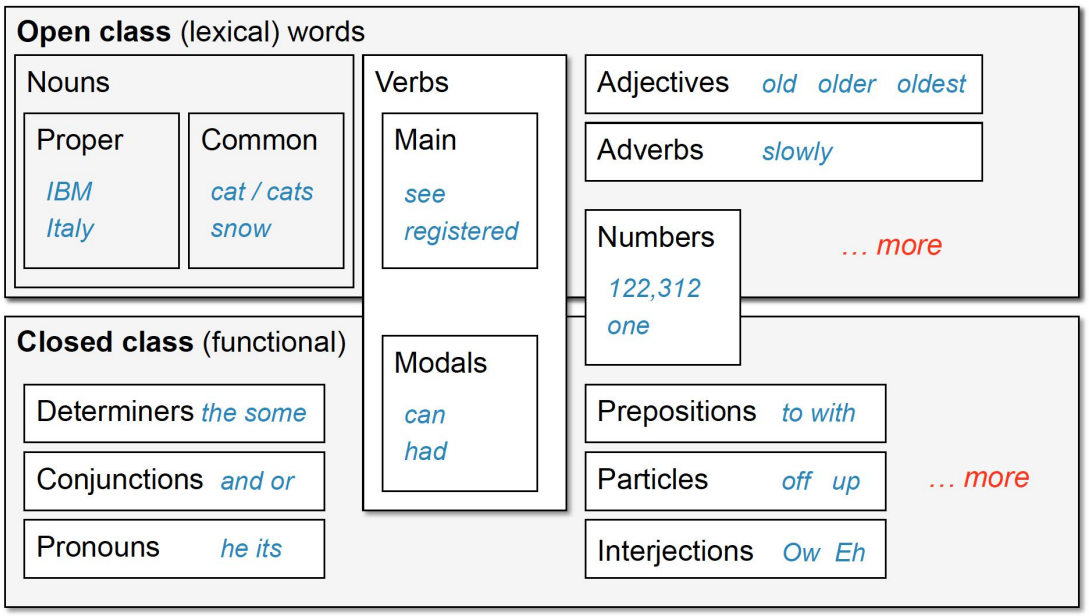


Figure 6: Open and closed classes

7.2 English Nouns

- Count nouns allow for grammatical enumeration

- Mass nouns are used for a homogeneous group of concepts which are not countable
- Proper nouns are used to mane people, places or things, and are generally capitalised

7.3 English Verbs

- Main verbs refer to actions and processes
- Auxiliary or Modal verbs change the meaning of main verbs
- There are four inflections
  - Non-third person singular (VBP)
  - Third person singular (VBZ)
  - Progressive (VBG)
  - Past participle (VBN)

7.4 Other English Part of Speech Components

- Adjectives characterise nouns
- Adverbs characterise verbs, adjectives, other adverbs, or word groups, they are separated into directional, degree, manner or temporal adverbs
- Prepositions specify relationships, be that spatial, temporal or agency
  arrive at the theatre on time
- Particles look like prepositions or adverbs but serve to modify the meaning of the verb that precedes them
  to find out
- Determiners appear directly before a noun
  - definite article: the
  - indefinite article: a, an
  - demonstratives: this, that, these, those
- Conjunctions join clauses
  - coordinating conjunctions link clauses of equal importance
    I respect his opinion but in this case he's factually wrong.
  - subordinating conjuctions a less important clause to complement a more important clause

7.5 Pronouns

- Relative pronouns introduce subordinate clauses that modify their antecedent
- Interrogative pronouns introduce questions
- Demonstrative pronouns identify or point at nouns
- Indefinite pronouns refer to nonspecific persons or things

7.6 Handling Ambiguity

For each ambiguous word token, choose the tag that is most frequent for that word token in the training corpus.

- PRP+VBD is very common, while PRP+VBN is exceedingly uncommon
- RB+RB is very common, also at the end of a sentence
- RB+JJ is also very common, but not at the end of a sentence
- IN+JJ and IN+RB are both fine, but there would have to be more word tokens afterwards

7.7 PoS Tagging as Sequence Classification

A sentence is a sequence of observations, thus tagging can use probabilistic methods to choose the most likely tag sequence. In practice, each token is classified independently while using information about the surrounding tokens as input features. For ambiguous tags the forward and backward sequence of the already identified tags can be used.

7.8 Markov Chains

Markov chains are models that identify the underlying probabilities of a sequence of random variables.

- a set of N state variables
- a transition probability matrix A, where $a_{i,j}$ is the probability of going from state i to state j
- an initial probability distribution over the N states
  - this is what is known *a priori* before observing the system
  - π_i is the probability that the chain will start in state i

A key assumption is that the next state only depends on the current state. For word sequences, this means that the assumption is that the next word only depends on the current word.

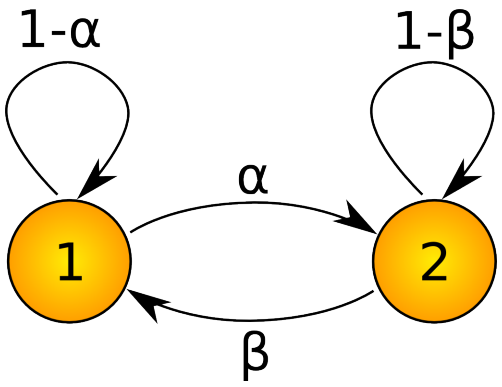


Figure 7: Two-state Markov Chain Model (QuantStart, 2020)

Notice that the probabilities sum to one for each state ($\alpha + (1 - \alpha) = 1$). The transition matrix A for this system is given by

$$\begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}$$

n steps of a Discrete State Markov Chain (DSMC) can be simulated simply b y repeated multiplication of the transition matrix with itself.

Unfortunately Markov Chains only work if the interest lies in the probability of observed events. In PoS tagging, interest lies in the probability of things that can't be observed directly. Namely, these are the tags, which are hidden. Fortunately **Hidden Markov Chains** enable the modelling of sequences with both observed events and related hidden events. In PoS tagging these correspond to words and tags.

- a set of N states
- a $N \times N$ **transition matrix** A where $a_{i,j}$ is the probability of transitioning from state i to state j
- a sequence O of T observations (drawn from a set of V possible observations): $O = o_1, o_2, \ldots, o_T$
- a $N \times V$ **emission matrix** B where $b_{i,t} = P(o_t|i)$ is the probability of state i generating observation o_t
- an initial probability distribution over the N states

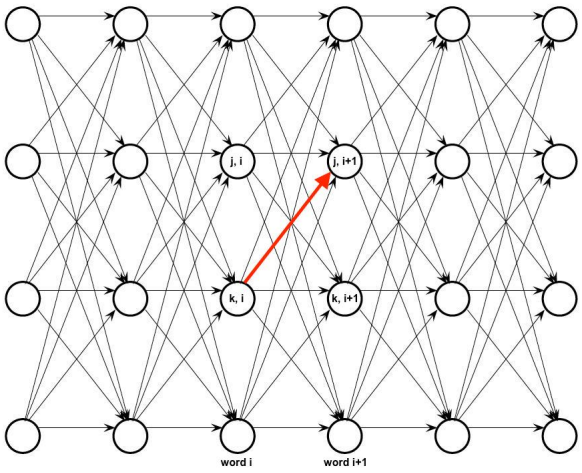
Let W be an observed word sequence and T be a hidden tag sequence. The interest lies in $\hat{T}$ over all possible tag sequences.

$$\hat{T} = \arg\max P(T|W) = \arg\max P(W|T)P(T) = \arg\max \prod_i P(w_i|t_i) \prod_i P(t_i|t_{i-1})$$

The process used to discover the tag sequence that maximizes $P(W|T)P(T)$ is called decoding and can be carried out using the Viterbi algorithm.

7.9 Viterbi Algorithm for Hidden Markov Models

The Hidden Markov Model may be represented as a graph called a trellis with a row for each tag and a column for each word



The ith column corresponds to the ith observation, the kth row corresponds to the kth hidden stage. The goal is to find the most probable path through the trellis, which refers to the path with the highest end-to-end probability computed as the products of the weights along the path.

The weight of the arrow from node (k, i) to node $(j, i+1)$ is the product of two probabilities

- the probability of going from state k to j $\quad a_{k,j} = P(j|k)$
- the probability of state j generating word $i+1$ $\quad b_{j,i+} = P(i+1|j)$

The path through the trellis with the highest end-to-end probability is the most likely tag sequence associated to the word sequence.

	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.7 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.8 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

8 Language Models

Language modelling is the task of predicting what word follows a sequence of words. The formal definition is: given a sequence of words $w^{(1)}, w^{(2)}, \ldots, w^{(t-1)}$ a language model computes the probability distribution of the next word $w^{(t)}$

$$P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(1)}\right)$$

where $w^{(t)}$ can be any word in a fixed vocabulary $V = (w_1, \ldots, w_{|V|})$.

We can also see a Language Model as a system that assigns a probability to any sequence of words. According to the chain rule the probability of a text $w^{(1)}, \ldots, w^{(T)}$ is given by

$$\begin{aligned} P\left(w^{(1)}, \ldots, w^{(T)}\right) &= P\left(w^{(1)}\right) \cdot P\left(w^{(2)} \middle| w^{(1)}\right) \cdot \ldots \cdot P\left(w^{(T)} \middle| w^{(T-1)}, \ldots, w^{(1)}\right) \\ &= \prod_{t=1}^{T} P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(1)}\right) \end{aligned}$$

The exact order of the words matters.

8.1 n-Gram Language Model

Some simplifications are needed for this idea to be practicable. Thus, the probability of a word $w^{(t)} \cdot P\left(w^{(t)}\right)$ only depends on the $n-1$ preceding words. This simplification is called a n-gram model where n-grams are chunks of n consecutive words. Each component is approximated with the following product

$$\begin{aligned} P\left(w^{(1)}, \ldots, w^{(T)}\right) &\approx \prod_{t=1}^{T} P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(i-(n-1))}\right) \\ P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(1)}\right) &\approx P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(i-(n-1))}\right) \end{aligned}$$

The n-gram probabilities are based on the Maximum likelihood estimate

$$P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(1)}\right) \approx \frac{\text{count}\left(w^{(t)}, w^{(t-1)}, \ldots, w^{(t-(n-1))}\right)}{\text{count}\left(w^{(t-1)}, \ldots, w^{(t-(n-1))}\right)}$$

8.1.1 Problems

- The context for a word is bigger than the n grams in the model
- Increasing n increases the sparsity of the model (typically n can't be bigger than five)

$$P\left(w^{(t)}\middle|w^{(t-1)},\ldots,w^{(1)}\right)\approx\frac{\text{count}\left(w^{(t)},w^{(t-1)},\ldots,w^{(t-(n-1))}\right)}{\text{count}\left(w^{(t-1)},\ldots,w^{(t-(n-1))}\right)}$$

- Numerator $w^{(1)},\ldots,w^{(t-1)},w^{(t)}$ might never occur in the training set
  Solution: Smoothing
- Denominator $w^{(1)},\ldots,w^{(t-1)}$ might never occur in the training set
  Solution: Backoff

8.1.2 Smoothing n-Gram Language Models

It can always happen that n-grams not seen in the training set appear in the data, Smoothing takes care of this by reassigning some probability to unseen n-grams. The simplest method for this is **Laplace smoothing**

$$P\left(\boldsymbol{w^{(t)}}\middle|w^{(t-1)}\right)\approx\frac{\text{count}\left(w^{(t-1)},w^{(t)}\right)+1}{\text{count}\left(w^{(t-1)}\right)+|V|}$$

Smoothing this way generally shifts the probability too much towards unseen n-grams. Compensate this by adding a fraction k (0.01, 0.03, 0.1, 0.3) instead of one. This method is called **add-k smoothing**

$$P\left(\boldsymbol{w^{(t)}}\middle|w^{(t-1)}\right)\approx\frac{\text{count}\left(w^{(t-1)},w^{(t)}\right)+k}{\text{count}\left(w^{(t-1)}\right)+k\cdot|V|}$$

Unfortunately, neither add-one nor add-k work well in practice.

8.1.3 Backoff in n-Gram Language Models

Use the count of the next lower-order n-gram, which means to use less context. Even better is to take backoff into account when n-grams are sparse, so use a linear combination of n-grams

$$P\left(w^{(t)}\middle|w^{(t-1)},w^{(t-2)}\right)=\lambda_1P\left(w^{(t)}\middle|w^{(t-1)},w^{(t-2)}\right)+\lambda_2P\left(w^{(t)}\middle|w^{(t-1)}\right)+\lambda_3P\left(w^{(t)}\right)\qquad\sum_i\lambda_i=1$$

8.2 Evaluation of a Language Model

- Extrinsic Evaluation
  - Put model A and model B through the same task
  - Run the task and get the accuracy for both models
  - Compare accuracies

  Cons: Time Consuming
- Intrinsic Evaluation
  - Easier and faster to use
  - Evaluation metric: Perplexity
    - Algorithm that scores better on the test set is considered better

  Cons: Bad approximation for real language (but it helps to get an idea on how good the model works)

8.2.1 Perplexity

Perplexity is the probability of the test set, normalized by the number of words

$$
\begin{aligned}
PP(W) = P(w^{(1)}, w^{(2)}, \ldots, w^{(N)})^{-\frac{1}{N}} &= \sqrt[N]{\frac{1}{P\left(w^{(1)}, w^{(2)}, \ldots, w^{(N)}\right)}} \\
&= \sqrt[N]{\prod_{t=1}^{N} \frac{1}{P\left(w^{(t)} \middle| w^{(t-1)}, \ldots, w^{(1)}\right)}} && \text{(chain rule)} \\
&= \sqrt[N]{\prod_{t=1}^{N} \frac{1}{P\left(w^{(t)} \middle| w^{(t-1)}\right)}} && \text{(for bigrams)}
\end{aligned}
$$

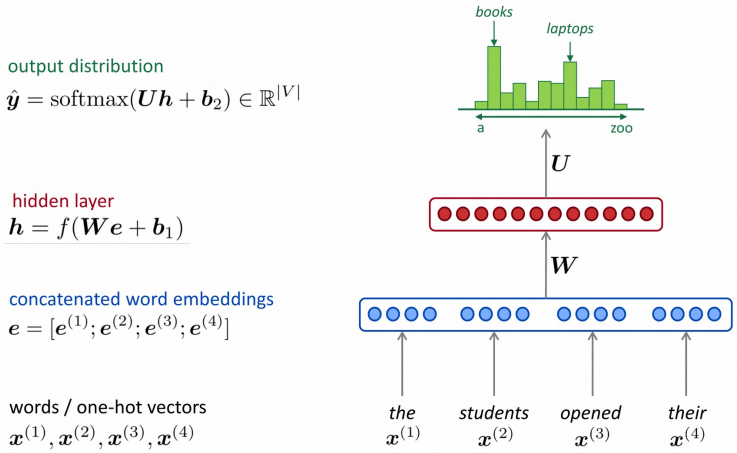
Minimizing perplexity is the same as maximizing probability.

Another evaluation metric is the word error rate (the number of insertions, deletions and substitutions normalised by sentence length needed to re-construct a reference sentence). The word error rate can be measured with the Levenshtein Edit Distance (which allows to weight insertions, deletions and substitutions differently) and is efficiently implemented using dynamic programming.

9 Recurrent Neural Networks and Language Models

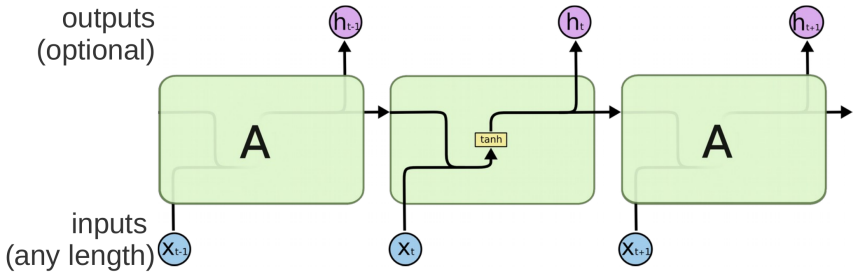
The task of a Language Model is at its core to take as an input a sequence of word $x^{(1)}, x^{(2)}, \ldots, x^{(t-1)}$ and output a probability distribution of the next word $P\left(x^{(t)} \middle| x^{(1)}, x^{(2)}, \ldots, x^{(t-1)}\right)$.

A fixed-window neural language model uses a fixed input size of preceding words to predict the next word



This model will provide an output distribution to any word combination (it might not be a good one) and while making the context window size n larger does not lead to larger sparse matrices. The size of the hidden layer is an independent hyper-parameter.

9.1 Recurrent Neural Network



$$h^{(t)} = \tanh\left(W_h h^{(t-1)} + W_e x^{(t)} + b_1\right)$$

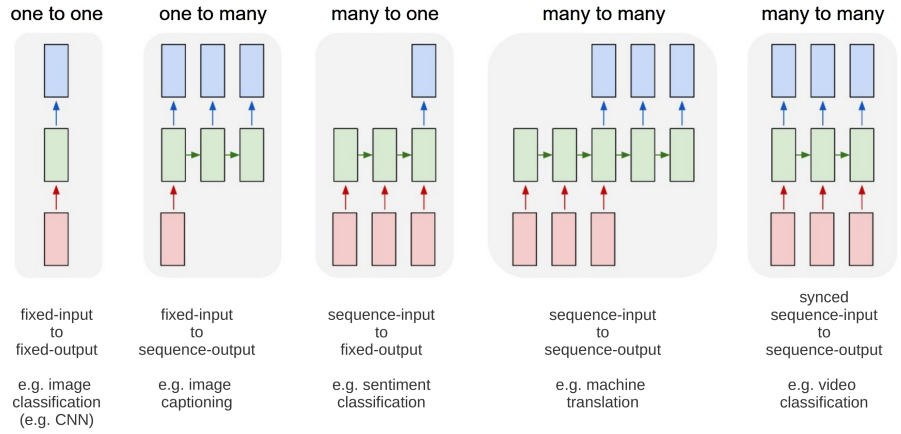
The hidden state vector $h^{(t)}$ has two inputs, the input word $x^{(t)}$ and the previous hidden state $h^{(t-1)}$. The core idea of RNNs is to apply the same weights repeatedly, thus the weight matrix W_h for weighing the previous hidden state vector $h^{(t-1)}$ and weight matrix W_e for weighing the one-hot encoded input word vector $x^{(t)}$ are shared.

Advantages

+ can process input sequences of any length
+ computation for timestep t can use information from any preceding timestep
+ model size does not increase for longer input
+ symmetry in how input is processed, as the same weights are applied for every timestep

Disadvantages

- recurrent computation is slow as no parallelisation is possible because inputs have to be handled sequentially by design
- difficult to access information from many time steps back in practice



Training a RNN amounts to feeding a big corpus of text into the RNN and computing the output distribution $\hat{y}^{(t)}$ at every step t. The loss function on step t is the cross-entropy between predicted probability distribution and the true word

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_w^{(t)} \log\left(\hat{y}_w^{(t)}\right) = -\log\left(\hat{y}_{x_{t+1}}^{(t)}\right)$$

The average of the loss is the overall loss for the training set

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T}\left(-\log\left(\hat{y}^{(t)}_{x_{t+1}}\right)\right)$$

Errors are backpropagated at each individual time step and then across all time steps, in a process that is called backpropagation through time. Computing gradients with regard to h_0 involves many factors of W_h and repeated gradient computations, which can lead to the exploding and vanishing gradient problem.

Measures against the problem

- **Exploding Gradients**
  - Gradient clipping to scale big gradients
- **Vanishing Gradients**
  - Activation Function
  - Weight initialisation
  - Network architecture

  see Pascanu et al., 2013 for an in-depth explanation

10 Seq2Seq and Attention for Machine Translation

Machine Translation is the task of translating a sentence x written in a **source language** to a sentence y written in the **target language**.

- Source sentence represented by input sequence $(x^{(1)}, x^{(2)}, \ldots, x^{(T)})$ of variable length T
- Target sentence represented by output sequence $(y^{(1)}, y^{(2)}, \ldots, y^{(T')})$ of variable length T'
- Elements of x and y come from the vocabulary of the respective language

10.1 Statistical Machine Translation

Learn a probabilistic model from data $P(y|x)$ by using the Bayes' rule. Break the problem into two probability functions and use statistical models to model each probability function separately

$$\text{argmax}_y P(y|x) = \text{argmax}_y P(x|y)P(y)$$

- Translation Model: Models how words and phrases should be translated (fidelity) and is learnt from **parallel data**
- Language Model: Models how to write good language (fluency) and is learnt from **monolingual data**
- Decoding Algorithm: Finds the translation that maximises the probabilities of the two models, this is usually done by beam search (a heuristic search algorithm)

Learn the translation model $P(x|y)$ by starting with a **large parallel corpus** like the Rosetta Stone. The language model can be broken down further into $P(x, a|y)$ where a is the **alignment**, which is the **correspondence between particular words** in the translated sentence pair.

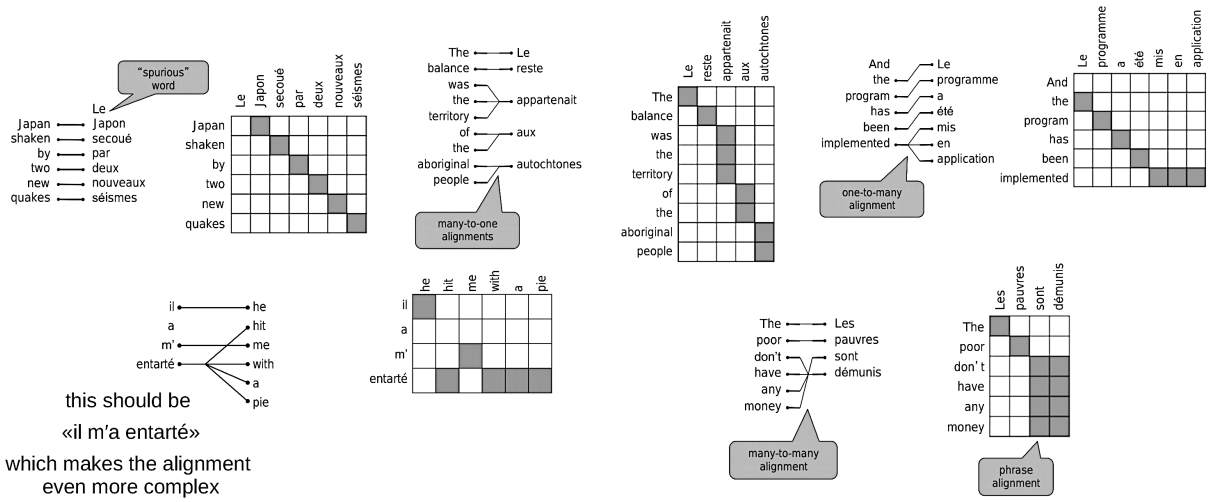
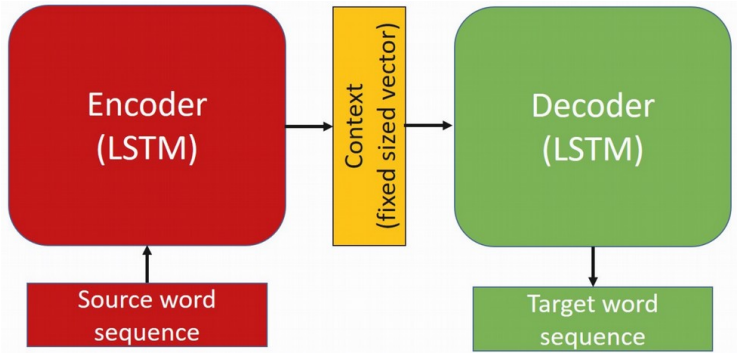


Figure 8: Examples showing the complexity of alignment

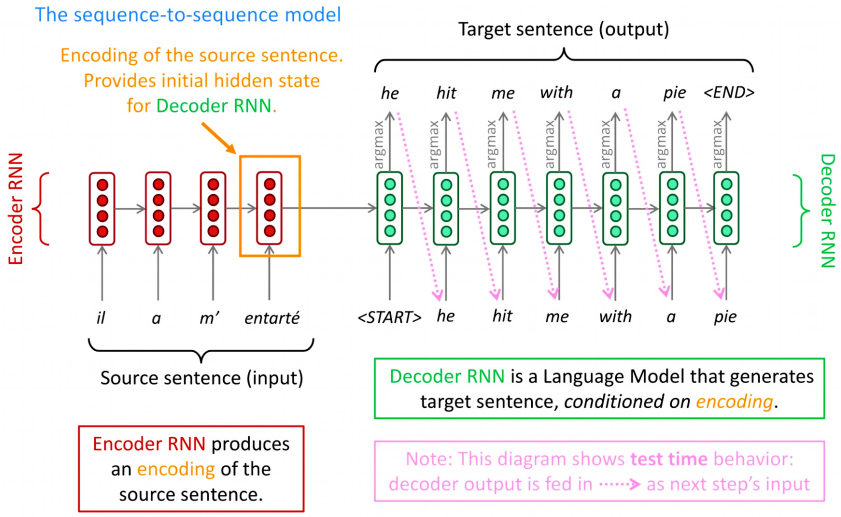
Statistical Machine Translation was a huge research field. But the best systems were extremely complex with hundreds of important detail and separately designed sub-components. A lot of human effort was needed to build and maintain the systems for each language pair.

10.2 Neural Machine Translation

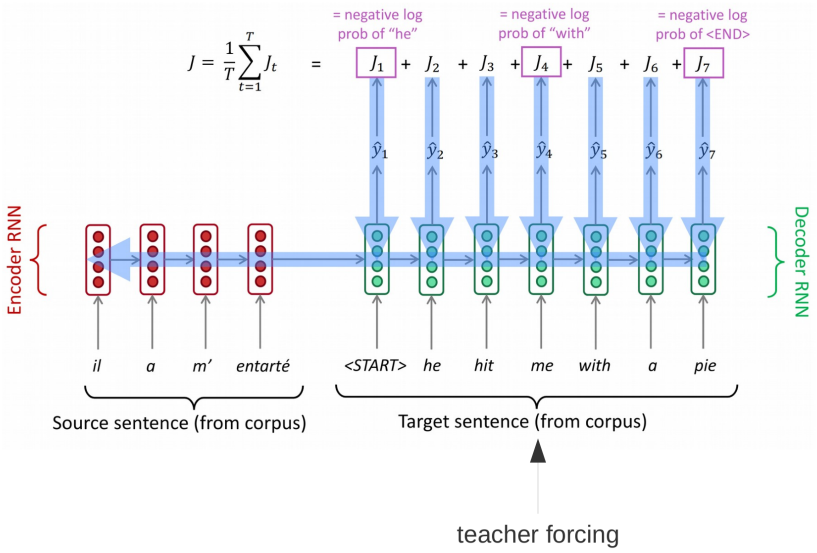
Neural Machine Translation is a way to do Machine Translation with a single neural network. The neural network architecture is called sequence-to-sequence and involves two RNNs. An **Encoder** network which processes the input sequence and produces a fixed size context vector, and a **Decoder** network which takes the context vector as input to produce the translated output word sequence.



The idea is to use an end-to-end neural network to directly implement $\text{argmax}_y P(y|x)$



This sequence to sequence (Seq2Seq) system is trained as single whole and the backpropagation operates from end to end. Using Teacher forcing: Supply expected translation (from corpus) instead of predicted output from previous timestep.



10.3 Scoring a Target Sequence

The output of the Decoder for the next word is a multinational probability distribution over the vocabulary. The idea behind the scoring is to maximise the probability for the complete translation of length T

$$P(y|x) = P(y^{(1)}|x) \cdot P(y^{(2)}|y^{(1)}, x) \cdot \ldots \cdot \underbrace{P(y^{(T)}|y^{(1)}, y^{(2)}, \ldots, y^{(T-1)}, x)}_{\text{probability of next target word given words so far and source sentence } x}$$

$$P(y|x) = \prod_{t=1}^{T} P(y^{(t)}|y^{(1)}, y^{(2)}, \ldots, y^{(t-1)}, x)$$

For numerical stability the logarithm is used to score a candidate solution

$$\begin{aligned} \text{score}\left(y^{(1)}, \ldots, y^{(t)})\right) &= \log\left(\prod_{t=1}^{T} P(y^{(t)}|y^{(1)}, y^{(2)}, \ldots, y^{(t-1)}, x)\right) \\ &= \sum_{t=1}^{T} \log\left(P(y^{(t)}|y^{(1)}, y^{(2)}, \ldots, y^{(t-1)}, x)\right) \end{aligned}$$

10.4 Beam Search

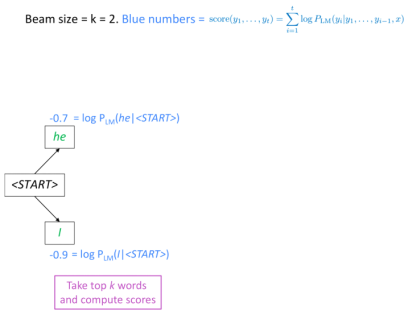
On each step of the decoder, keep track of the k **most probable partial translations**, which are called hypotheses. Beam search is a heuristic search strategy, where k is called the beam size (in practice around 5 to 10).

In order to choose the best k hypotheses for the next word in time step t we score the alternatives.

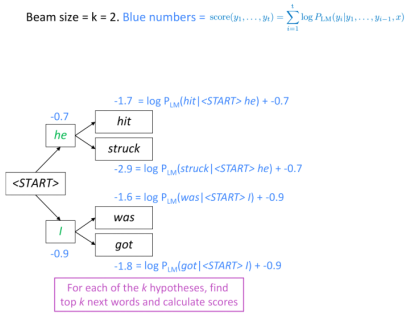
$$\text{score}\left(y^{(1)},\ldots,y^{(t)}\right) = \sum_{t=1}^{T} \log\left(y^{(t)}\middle|y^{(1)},\ldots,y^{t-1},x\right)$$

Scores are all negative, a higher score is better. Search for a high-scoring hypothesis while tracking the top k on each step.

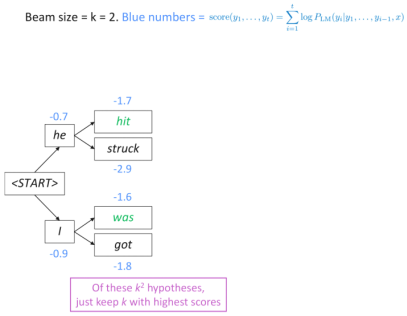
(a) Step 1



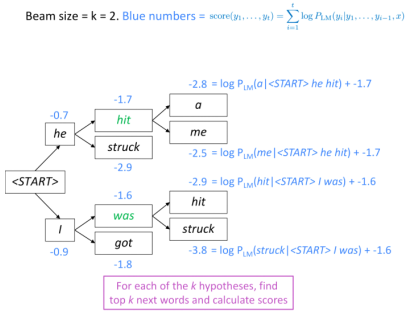
(b) Step 2



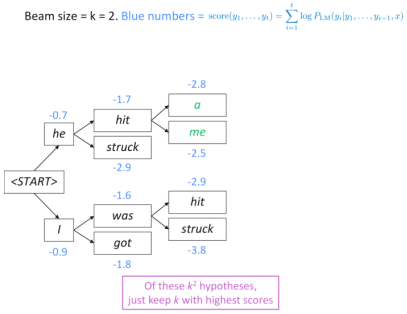
(c) Step 3



(d) Step 4



(e) Step 5



(f) Step 6

Figure 9: Beam Search full example

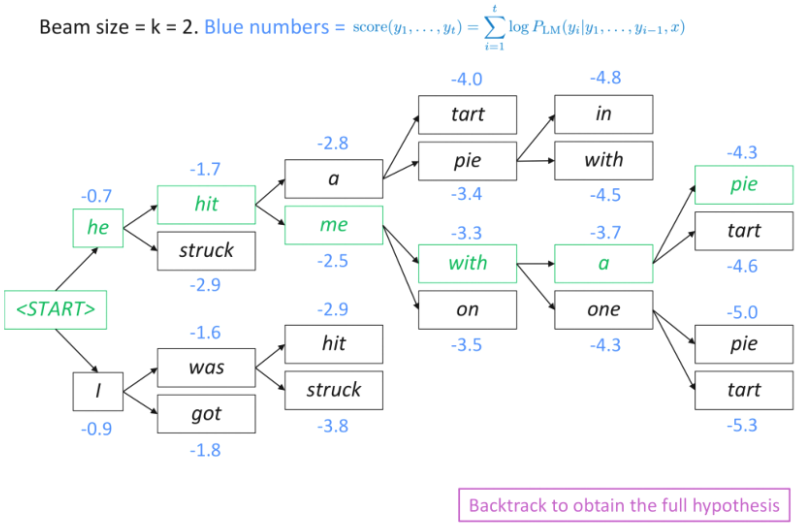


Figure 10: Full hypothesis of Beam Search

10.5 Hypotheses Selection

- In **greedy decoding**, the input gets decoded until the model produces the **<END> token**
- In **beam search decoding** different Hypotheses may produce the <END> token in different timesteps, if that happens the hypothesis is saved and other hypotheses are continued to be explored until a predefined cut-off at timestep T is reached or until at least n complete hypotheses are acquired.

  Then the hypothesis with the best score is selected, but longer hypotheses have lower scores (normalised by length)

$$\frac{1}{T}\sum_{t=1}^{T} \log\left(P(y^{(t)}|y^{(1)},\ldots,y^{(t-1)},x)\right)$$

10.6 Evaluating Machine Translation Systems using BLEU

The **Bi**lingual **E**valuation **U**nderstudy compares the **machine written translation** to one or several **human written translations** and computes a **similarity score**

- Measures the ratio of matched n-grams (overlap) to the total number of n-grams in the translated text (range [0..1] but often scaled to [0..100])
- Typically 1,2,3 and 4-gram precision which are equally weighted
- Penalty for translations that are too short using a brevity penalty (BP) factor

$$\text{BLEU} = \text{BP} \cdot e^{\sum_{n=1}^{N} w_n \log(p_n)}$$

$$p_n = \frac{\sum_{\text{n-gram}\in\hat{y}} \overbrace{\text{count}_{\text{clip}}(\text{n-gram})}^{\#\text{ matched n-grams}}}{\sum_{\text{n-gram}\in\hat{y}} \underbrace{\text{count}(\text{n-gram})}_{\#\text{ n-grams in translation}}}$$

$$\text{BP} = e^{\min\left(0,1-\frac{\text{len}_{\text{reference}}}{\text{len}_{\text{translation}}}\right)}$$

N n-grams considered up to length N

w_n n-grams specific weight (default $\frac{1}{N}$)

BLEU should be used on corpus level, not on individual sentences, and while it is useful, it is imperfect. There are many valid ways to translate a sentence, and a good translation can get a poor BLEU score because it has a low n-gram overlap with the provided human translations.

10.7 Notes on Machine Translation

- **Reliability**: NMT translations can be unreliable, offensively wrong, or utterly unintelligible. NMT systems have no guarantees about accuracy and can miss negations, whole words, or entire phrases.
- **Memory**: NMT systems are often built to translate one sentence at a time. As a result, they forget information gained from prior sentences.
- **Common sense**: NMT systems have very little common sense, that is information of the external context and knowledge about the world. Understanding what contexts are appropriate for certain translations is important to our understanding of situations, but these contexts are often difficult to capture in their entirety.

10.8 Attention

The source word encodings $x_1, x_2, \ldots, x_T$ are individually weighted depending on the target word y_{t-1}. Therefore, there are as many context vectors c_t as there are target words y_t and every context vector c_t for target word y_t is a weighted average of the source words encodings. Depending on the previous target word y_{t-1} the source word encoding will be weighted differently. This method is called global attention[1].

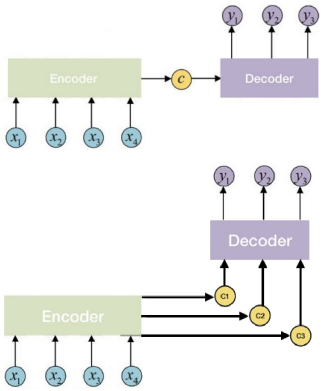
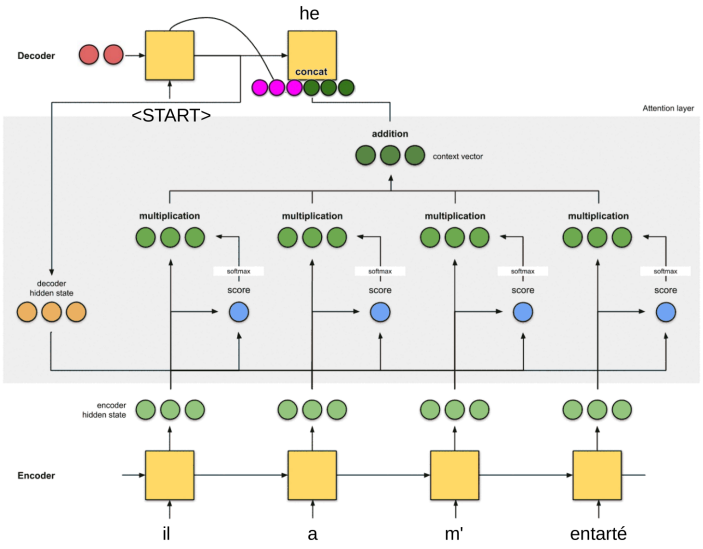


Figure 11: Schematic comparison between a plain seq2seq model (top) and a seq2seq model with attention (bottom)

[1] A good example of attention calculated through on a simple Input and Output



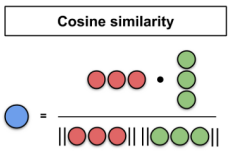
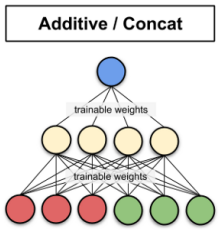
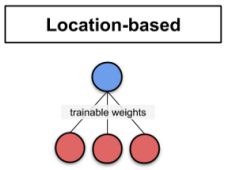
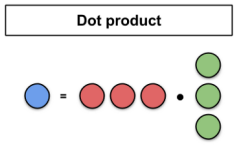
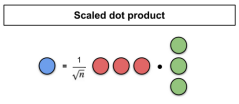
Name	Description	Score	Schema
Content-based	(Graves et al., 2014)	$\text{score}(s_t, h_i) = \text{cosine}\left[s_t, h_i\right]$	
Additive, Concat	W_a is a weight matrix and v_a a weight vector, which both are being trained (Bahdanau et al., 2014)	$\text{score}(s_t, h_i) = v_a^T \tanh\left(\text{W}_a[s_t; h_i]\right)$	
Location-based	Simplifies the SoftMax alignment to only depend on the target position (Luong et al., 2015)	$a_{t,i} = \text{softmax}\left(W_a s_t\right)$	
Dot-Product	(Luong et al., 2015)	$\text{score}(s_t, h_i) = s_t^T h_i$	
Multi-plicative	W_a is a trainable weight matrix in the attention layer (Luong et al., 2015)	$\text{score}(s_t, h_i) = s_t^T W_a h_i$	
Scaled Dot-Product	N is the dimension of the source hidden state. Motivation for scaling is to improve learning when input is large, as the SoftMax may have extremely small gradients (Vaswani et al., 2017)	$\text{score}(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	

Table 6: Attention score mechanisms

Attention ...

- significantly improves NMT performance, the intuition is that it is effective as it allows the decoder to focus on certain parts of the source.
- solves the bottleneck problem, as it allows the decoder to bypass the bottleneck and look directly at the source
- helps with the vanishing gradient problem, as it provides shortcuts to faraway states
- provides some interpretability, by revealing what the decoder is focussing on and by the network learning alignment

Attention is a technique to compute a weighted sum of the values, dependent on the query and the keys, which is a selective summary of the information contained in the values, where the query and the keys determine how much to focus on the individual values. Thus, Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), depending on some other representations (the query and the keys).

11 Understanding Syntax

Syntactic parsing refers to mapping the syntactic structures of sentences. Formal grammar is a set of rules used to build syntactically valid sentences. Nearly all natural languages seem to follow the Context-Free Grammar model, with Swiss-German being a notable exception needing a mildly context-sensitive grammar[2].

11.1 Context-Free Grammar

A context-free grammar includes

- a set of terminals T
  words you see on the page
- a set of non-terminals N
  labels given to words and group of words
- a designated start symbol $S \in N$
- a set of rules, so-called *productions*, of the form $A \rightarrow \beta$
  - A is a non-terminal (Left-Hand Side)
  - β is a sequence of terminals and non-terminals (Right-Hand Side)
  - $A \rightarrow \beta$ means replace A with β
- rules can be applied regardless of context (hence context-free grammar)
- when a rule is applied to a symbol, nothing matters other than the symbol the rule is applied to

[2] Shieber, 1985.

11.2 Key Jargon from Linguistics

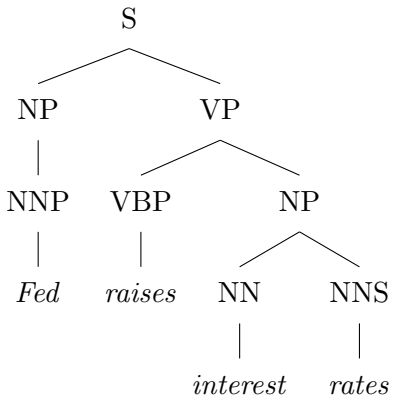
Smaller units combine into increasingly complex ones

- phrase, a group of words standing together as a conceptual unit
  - Noun Phrase, a phrase built around a noun
  - Verb Phrase, a phrase built around a verb
  - Prepositional Phrase, a phrase including a preposition and a noun phrase that indicates relations in space and time
  - Adjective or Adverb Phrase, a phrase built around adjectives and adverbs
- clause, includes at least a subject and a verb
- sentence, a group of clauses
  - simple sentence, only an independent clause
  - compound sentence, at least two independent clauses
  - complex sentence, one independent clause and one or more dependent clauses
  - compound-complex sentence, two or more independent clauses and one or more dependent clauses

A context-free grammar represents how words and phrases are organised in a sentence. If a sentence is a part of the language defined by a context-free grammar, the organisation of its constituents is known. In practice a probabilistic context-free grammar (PCFG) is needed, where every rule has a certain probability, with the sum over all rules being one.

11.3 Constituency Structure

- *interest* and *rates* go together as noun phrase
- *raises* and *interest rates* go together as a verb phrase
- *Fed* and *raises interest rates* go together as a sentence
- Each NP and VP unit is a **constituent**
- Constituents can be indicated with square brackets
  Fed [raises [interest rates]]



To determine what a constituent is a complex question and different linguists often disagree on. But there are some basic tests

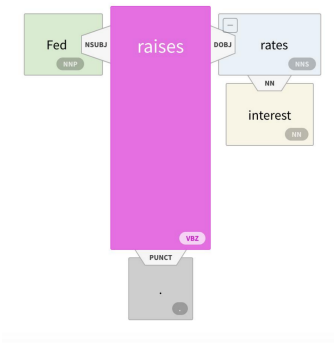
- Look for words that stay together when phrases move around
- Phrasal expansion and substitution
  I sat [on the box]
  I sat [on the box — right on top of the box — on top of it — there]
- Lots of other tests

11.3.1 Clause-Level Constituents

S	simple declarative clause
SBAR	clause introduced by a (possibly empty) subordinating conjunction
SBARQ	direct questions introduced by a wh-word or a wh-phrase
SINV	Inverted declarative sentence
SQ	Yes or No questions

11.4 Dependency Parsing

The basic idea behind dependency parsing is that every word is another word's dependent, except for the root. This root corresponds to the central theme of the sentence.



11.5 Key Algorithms

- Probabilistic Cocke-Kasami-Younger (CKY) Parser, Constituency Parsing
- Arc-Standard Transition-Based Parser, Dependency Parser
- Neural Dependency Parser

11.5.1 Arc-Standard Transition-Based Parser

At any given time, the parser has a configuration
- stack of words (left to right)
- buffer of words (left to right)
- (partial) dependency graph

The parser begins in an **initial configuration**

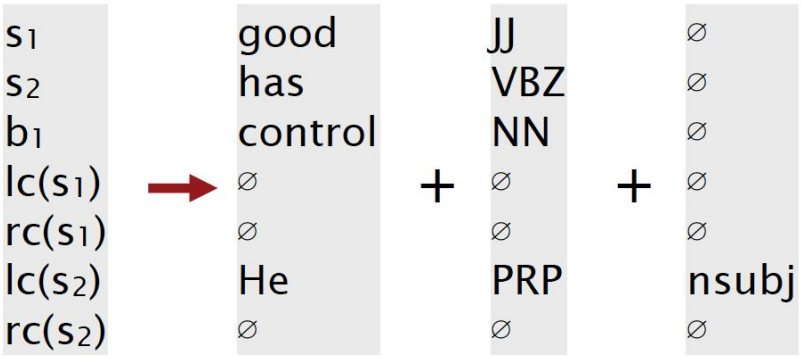
stack	ROOT symbol
buffer	all words
dependency graph	empty

The possible actions are

1. SHIFT: move a word from the top of the buffer to the top of the stack
2. LA_r: Left Arc
   - the neighbour of the head of the stack is a dependent of the head of the stack
   - add a corresponding link to dependency graph with label r
   - remove the neighbour from the stack
3. RA_r: Right Arc
   - the head of the stack is a dependent of its neighbour
   - add a corresponding link to dependency graph with label r
   - remove the head of the stack

11.5.2 Neural Dependency Parser

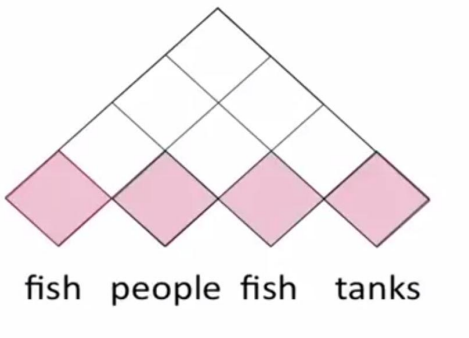
Represent words as d-dimensional word embeddings with $d \approx 1000$, do the same to represent PoS tags and already known dependency labels. Extract a set of tokens based on stack or buffer positions. Look up their embedding vectors and concatenate them with the PoS and label embedding vectors.



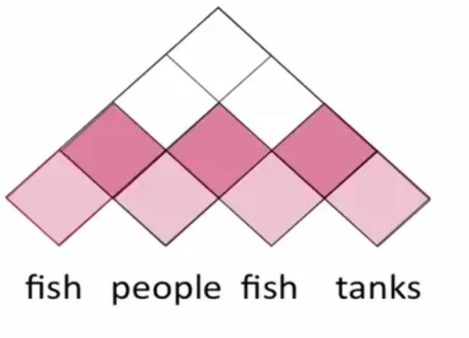
- Input layer, the concatenated embedding vectors
- Feedforward neural network with ReLu non-linearity
- Output Layer, Softmax gives a probability distribution
- Using Cross-Entropy Loss to measure the error

11.5.3 CKY Constituency Parsing

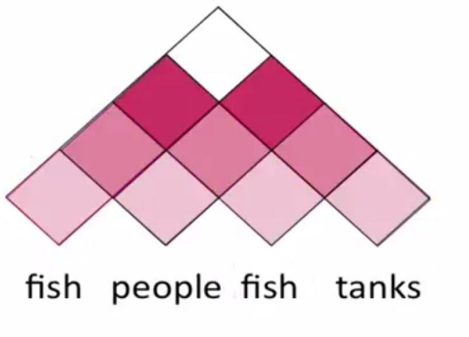
CKY uses a parse chart



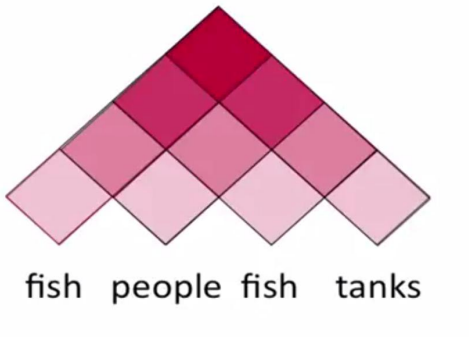
(a) Fill in the cells with the things that can be described in single words



(b) Fill in the second row cells that describe combinations of two words



(c) Fill in the third row cells that describe three-word constituents



(d) Fill in the fourth row cells that covers the whole span

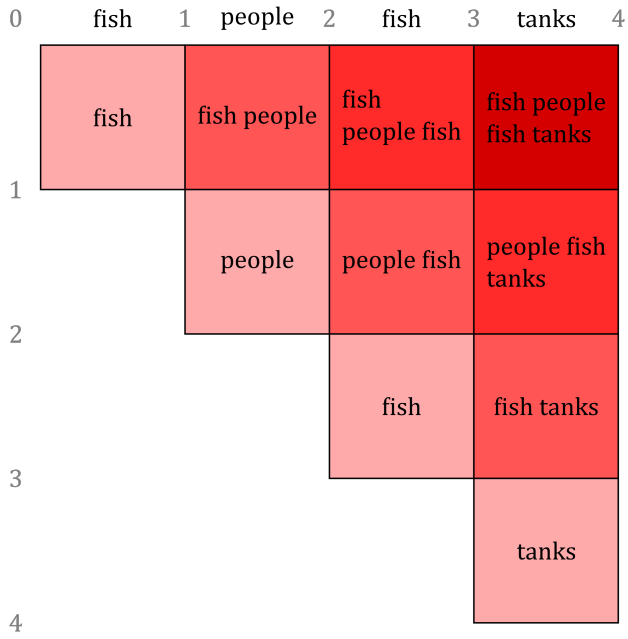


Figure 13: In practice it is easier to tilt the chart

12 Information Extraction

The goal of information extraction is to extract the unstructured information embedded in texts and transform it into structured data. Structured data can be used to populate a relational database to enable further processing or anything else that requires structured input data.

Typical information extraction tasks

- **Named Entity Recognition**
  to find names of people, places, organisations
- **Co-reference resolution and entity linking**
  to group together named entities into sets corresponding to real-world entities
- **Relation extraction**
  to find semantic relationships among entities in the text
- **Event extraction and co-reference**
  events are denoted by verbs or nouns, but temporal order matters very much
- **Template filling**

12.1 Named Entity Recognition

Fine-graded tags are often used

GPE	Geo-Political Entity
FAC	Facility
NORP	Nationalities, Religious or Political Groups

12.1.1 Named Entity Recognition as Sequence Labelling

NER is a word-by-word sequence labelling task where the tags capture the boundary and type. Use a sequence classifier to label the tokens with IO, IOB or IOBES tagging

I-TYPE	inside TYPE entity
O	outside any entity
B-TYPE	beginning of TYPE entity
E-TYPE	end of TYPE entity
S-TYPE	singleton TYPE entity

IO, when compared to IOB, has only I entities and no B-TYPEs, this makes things easier for machine learning system but at the cost of more ambiguity.

12.1.2 Features for NER

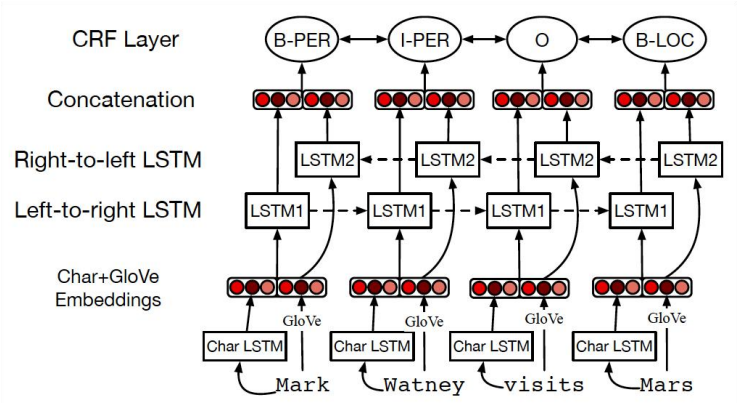
- Syntactic chunk labels, which allow parsing through basic constituent analysis
- Gazetteer, list of toponyms
- Word shapes, which are used to represent the abstract letter pattern of a word by mapping lower-case letters to x, upper-case to X, numbers to d, and retaining punctuation

12.2 Neural Methods

Two basic intuitions

- Names often consist of multiple tokens, so each tagging decision requires joint reasoning over multiple tokens
- Token-level evidence that a token represents a name includes orthographic (use character-level embeddings) and distributional evidence (use word embeddings)

Feeding the output layer of a RNN directly to a softmax does not allow to impose any constraints on the tag sequence. To impose strong constraints on the output, a Conditional Random Field layer on top of the Bi-LSTM output is used. Bidirectional RNNs are used to capture the left and right context of each word.



For a word sequence $\mathbf{X}$ and a sequence of predictions $\mathbf{y}$, a score is computed

$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

$A_{y_i,y_{i+1}}$ is the transition score (going from tag y_i to tag y_{i+1}), and P_{i,y_i} is the emission score (labelling word i as y_i)

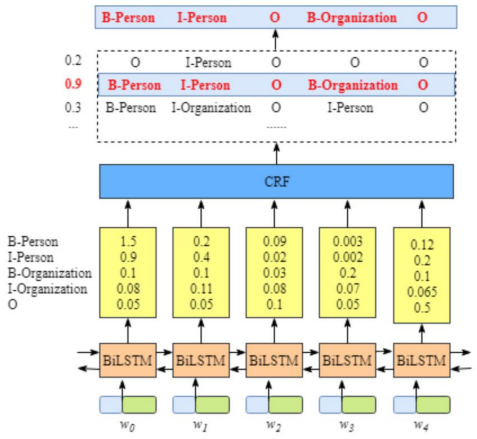


Figure 14: The CRF scores are not a probability but the raw output of the RNNs

A softmax layer over all possible tag sequences $\mathbf{Y_X}$ gives a probability for the sequence of predictions $\mathbf{y}$

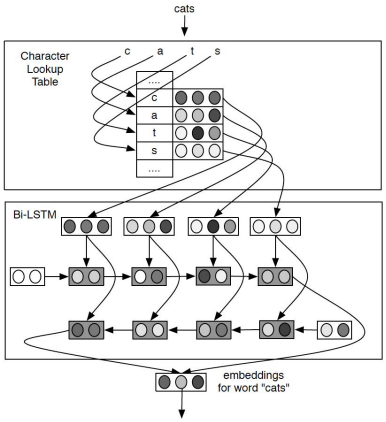
$$p(\mathbf{y}|\mathbf{X}) = \frac{e^{s(\mathbf{X},\mathbf{y})}}{\sum_{\tilde{\mathbf{y}}\in\mathbf{Y_X}} e^{s(\mathbf{X},\tilde{\mathbf{y}})}}$$

The global normalisation prevents the label bias problem of Maximum Entropy Markov Models.

12.2.1 Character-Level Embeddings

Such embeddings are very useful to represent unknown words based on their characters.

- Input: sequence of characters
- Projection Layer (PL): lookup table to capture similarities across characters
- Forward LSTM: returns a state sequence corresponding to the sequence of character representations from the projection layer
- Reverse LSTM: returns a state sequence corresponding to the inverse sequence
- Combine the two sequences into an embedding vector



13 Contextual Embeddings and Transformer Architecture

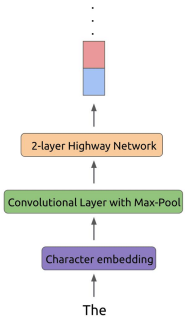
Word embeddings have revolutionized the field of Natural Language Processing

- 2013: Word2Vec
- 2014: GloVe
- 2015: FastText

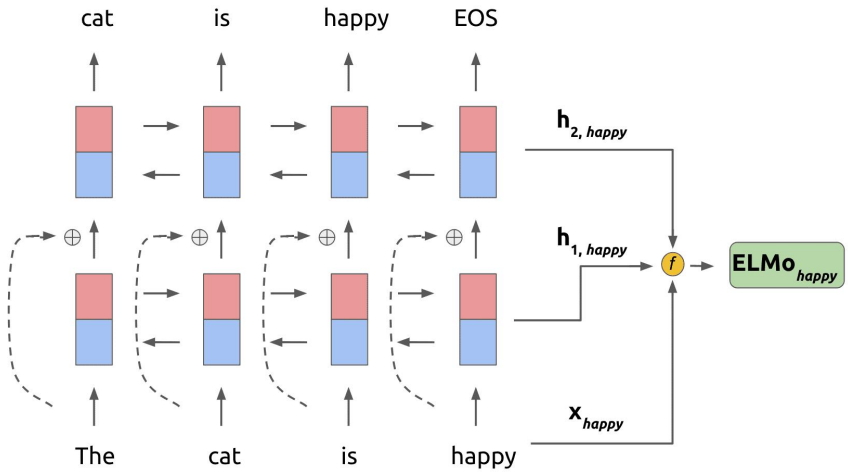
But the problem with word representations is that a **word type has the same word embedding regardless of context**. Words also have different aspects depending on semantics, syntactic behavior, register or connotations. The solution to this is contextual word embeddings.

13.1 ELMo

ELMo is based on a bidirectional language model and a Bi-LSTM. Their goal was to make the language model reasonably compact so people can use it even with limited hardware. ELMo has two BiLSTM layers with 4096 units and an output projected to 512 dimensions, with residual connections from the first to second layer.



Character embeddings allow to capture morphological features and handle out-of-vocabulary words, while convolutional filters capture local patterns. The key idea behind ELMo was to not just use the output, but use all the available hidden layers and combine them in a task-specific way with task-dependent trainable weights that you learn for each task.



13.2 Transformer Architecture

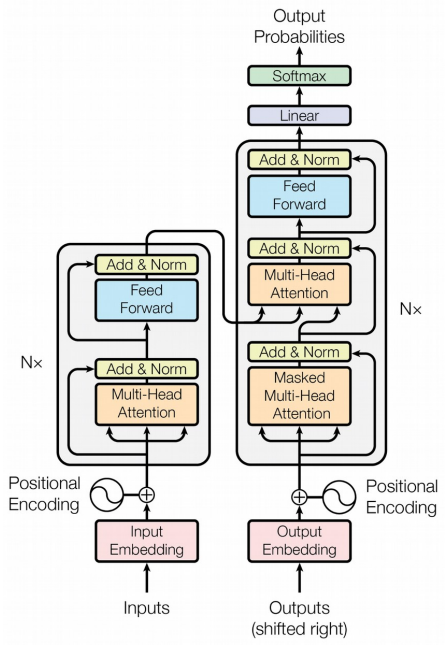
Quick Overview:

- Positional Embeddings: Models relative positioning and ordering of words
- Self-Attention: Encodes the context (contextualized word embeddings)
  Long-distance context has equal opportunities (without positional encoding it cannot even tell the difference since the path length of all words is one)
- Multi-Head Attention: Independently and simultaneously focus on different positions of the input in parallel
- Feed-Forward layers: Computes non-linear hierarchical features
- Residual Connections: Flow of positional information across layers
- Layer-Normalization: Makes it possible to train the network

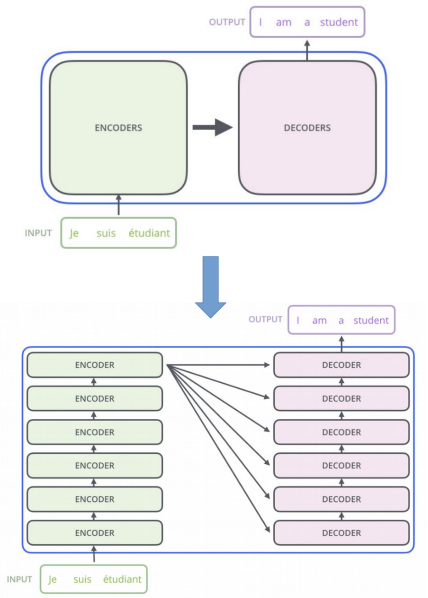
- Transformer Architecture is designed to allow for calculations in matrix form which can be efficiently implemented on GPUs/TPUs

Transformers were introduced in the paper "Attention is all you need" by Vaswani et al. to address the problem that RNNs require sequential calculations by design, which prohibits parallelisation and thus slows down training. The transformer solves this problem by employing an Encoder/Decoder structure.

- **Encoder**:
  N stacked **multi-head attention layers**, followed by a fully connected feed forward layers
- **Decoder**:
  N stacked **masked** multi-head attention layers, followed by multi-head attention layers (with input from encoder and decoder layers), followed by a fully connected feed forward layers



(a) Simplified Encoder and Decoder architecture of a Transformer



(b) Exploded view of stacked encoders and decoders

In the first layer the input words are mapped to their embeddings, the words in each position flow through their own paths, which can be executed in parallel. Self-attention looks at other positions in the input sequence for clues that lead to a better encoding for the current word, which results in contextualised word embeddings.

The attention capacity is increased by splitting attention heads using different projections or subspaces. This improves the performance of the attention layer by providing multiple representation subspaces, with multiple Query, Key and Value weight matrices projecting the inputs (word embeddings or vectors from encoders/decoders) into different subspaces. Thus expanding the model's ability to independently and simultaneously focus on different positions of the input sequence in parallel.

13.2.1 Positional Encoding

Position and order of words are essential parts of any language and are inherently taken into account by RNNs. A transformer however does not have a sense of position or order. This is addressed by adding a vector to each input embedding. These vectors follow a specific pattern

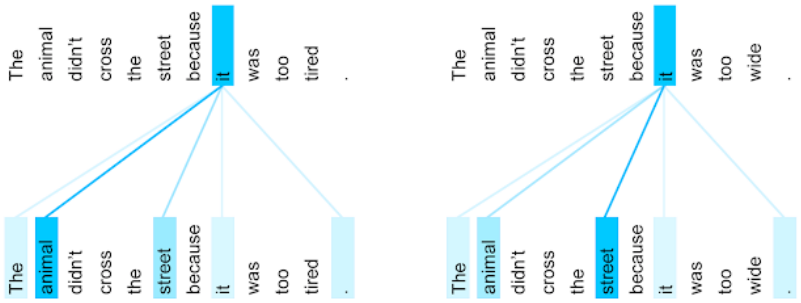


Figure 16: Different self-attention embeddings for 'it' depending on position and context

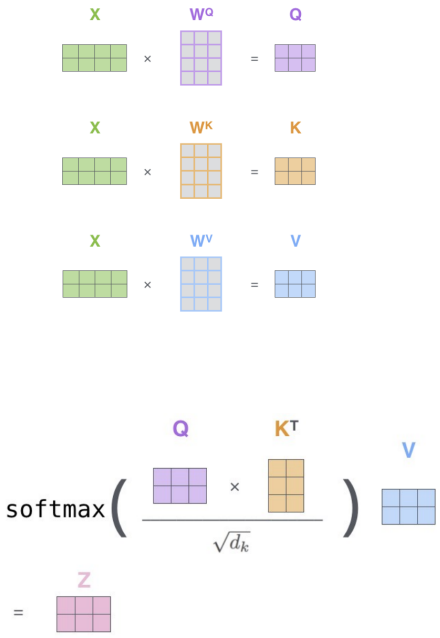


Figure 17: Self-attention calculations done in matrix form for faster processing

that allows to determine the position of each word and the distance between words. This encoding satisfies the following criteria:

- Output a unique encoding for each time-step or word position in the sentence
- Distance between any two time-steps is consistent across sentences with different lengths
- Model generalises to longer sentences without any efforts and with bounded values
- Model is deterministic

The proposed method by Vaswani et al. is to use a d dimensional vector which contains information about a specific position in a sentence. This encoding is not integrated in the model itself but enhances the word with information about its position in a sentence [3].

Let t be the desired position in an input sentence, $\vec{p_t} \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension (where $d \equiv_2 0$) Then $f : \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector $\vec{p_t}$ and it is defined as follows:

$$\vec{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases}$$

[3] Transformer Architecture: The Positional Encoding by Amirhossein Kazemnejad

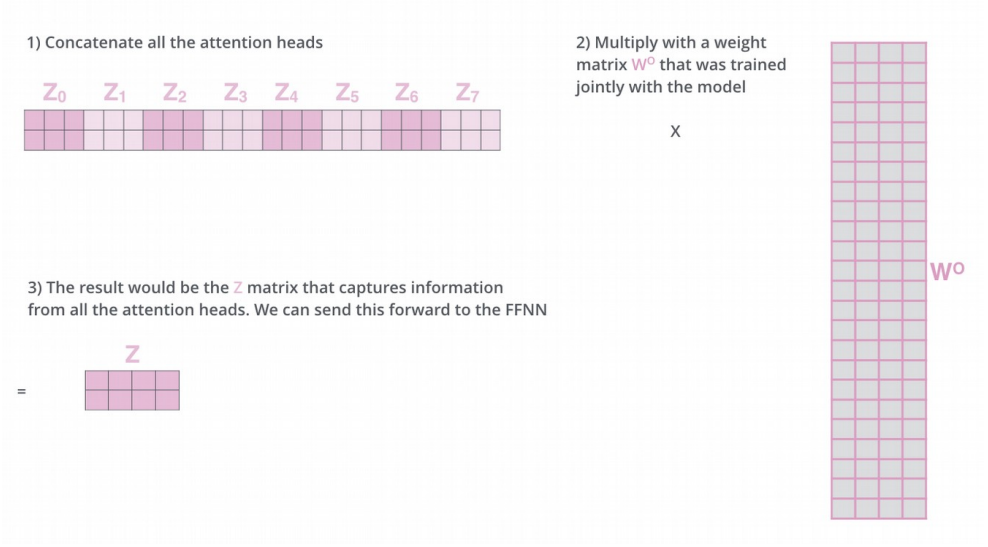


Figure 18: Three attention heads with their results being concatenated and multiplied by an additional weight matrix

where $\omega_k = \frac{1}{10000^{2k/d}}$. Calculating the correspondent embedding which is fed to the model is as follows

$$\psi'(w_t) = \psi(w_t) + \vec{p_t}$$

To make this summation possible, the positional embedding's dimension has to be equal to the word embeddings' dimension, thus

$$d_{\text{word embedding}} = d_{\text{positional embedding}}$$

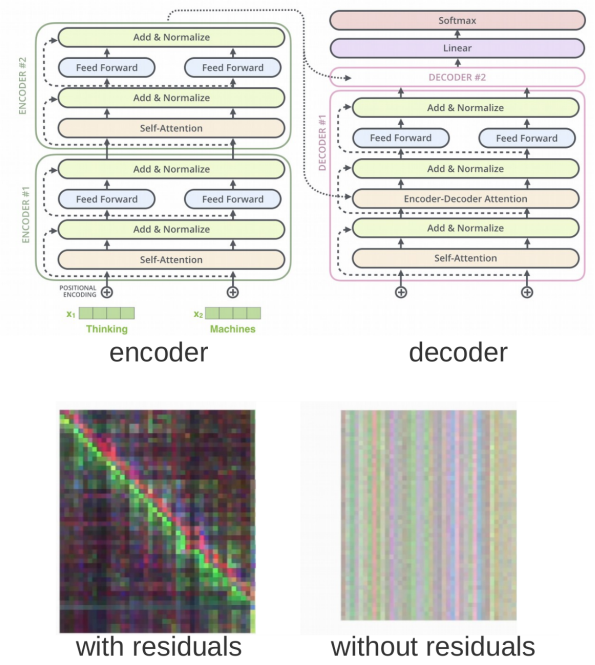


Figure 19: Each sub-layer in each encoder has a residual connection followed by a layer-normalisation step

The decoder has a final linear layer which is a fully connected neural network that projects the vector produced by the decoder stack into a vector with the size of the output vocabulary.

The linear layer is followed by a softmax layer turning the score for each individual word into a probability. The training is done by comparing the ground truth with predicted output for each position and using teacher forcing. This is done by supplying the expected translation instead of predicted output from previous time step, but self-attention layers are only allowed to attend to earlier positions in the output sequence by using masks (see figure 20). Otherwise, the decoder could just attend to the next word in the provided sequence.

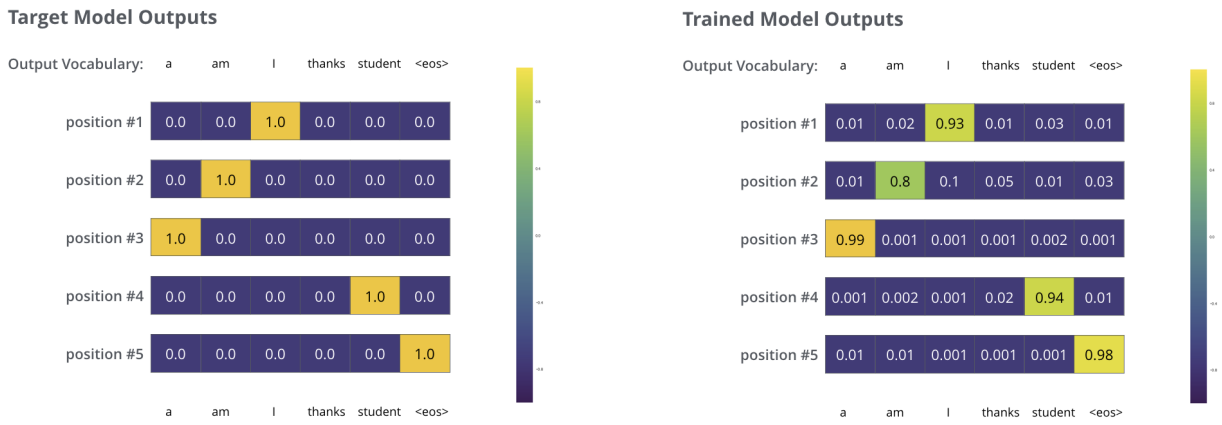


Figure 20: Masked target output and trained model output

13.3 Transformer Models

13.3.1 BERT: Bidirectional Encoder Representation from Transformers

Pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. Pre-trained BERT representations can be fine-tuned with just one additional output layer to create models for a wide range of tasks without substantial task-specific architecture modifications.

Initial training is done in two steps

- Perform an unsupervised pre-training on a large amount of unlabelled data to train a general purpose language understanding model by learning a latent representation of the input text
- Reuse pre-trained model and perform an individual supervised fine-tuning on a small amount of labelled data for the downstream tasks

This provides a Transfer Learning platform for NLP tasks. BERT uses WordPiece tokenisation for input representation. For Pre-Training Masked Language Modelling (MLM), where $k\%$ of the input words are randomly masked and the model has to predict the original words based on the context, and Next Sentence Prediction (NSP) is used. The goal of NSP is to learn relationships between sentences by predicting if a sentence B is an actual sentence that proceeds sentence A, or just a random sentence.

This pre-trained BERT can be used to create contextualised word embeddings for existing models.

13.3.2 Reducing the Size of Trained Models

- **Distillation**
  Train a smaller model to replicate the behaviour of the original model
- **Pruning**
  Make model smaller by removing not- or less-important Weights or Neurons, Layers, or Heads

- **Quantisation**
  Approximate the weights of a NN with lower precision (int8 instead of float32 often works quite well)
- **TensorRT**
  Platform for high-performance DL inference

13.3.3 Summary

- Pre-training and fine-tuning Transformer models works very well
- Models are extremely expensive
- Improvements come from even more expensive models and more data
- One option to address the inference or serving problem is through distillation

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International conference on machine learning*, 1310–1318.

QuantStart. (2020). *Hidden Markov Models - An Introduction*. Retrieved April 12, 2020, from https://www.quantstart.com/articles/hidden-markov-models-an-introduction/

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. In *Philosophy, language, and artificial intelligence* (pp. 79–89). Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998–6008.