Natural Language Processing

1. Classification and Simple Language Models

Preprocessing

Preprocessing Steps

1. Remove unwanted formatting (e.g. HTML)
2. Sentence segmentation: break documents into sentences
3. Word tokenisation: break sentences into words
4. Word normalisation: transform words into canonical forms
5. Stopword removal: delete unwanted words

Sentence Segmentation

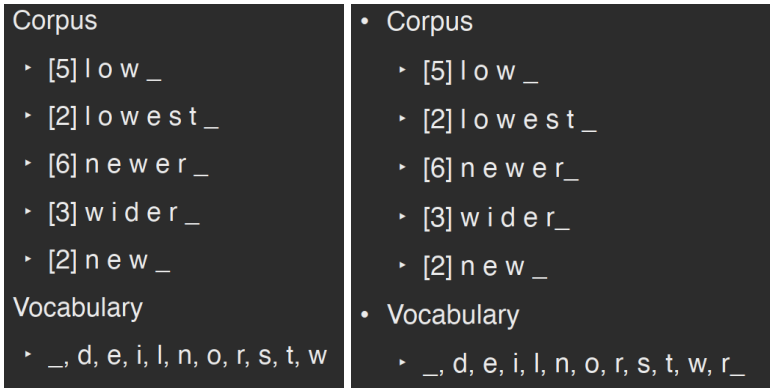
- Naïve approach: break on sentence punctuation. But periods are used for abbreviations, and hyphens occur within words (U.S. dollar, …, Yahoo! as a word)
- Regex: use regex to require capital. But abbreviations often followed by names (Mr. Brown).
- Lexicon: compile a list of exceptions to the regex method to account for such abbreviations and other cases. The problem is this will always be incomplete and contain errors.
- Machine learning: state-of-the-art approach uses statistical techniques rather than rules. The features used are the words and punctuation appearing before and after every period.

Word Tokenisation

- Need to deal with all the punctuation that occurs within a word, including periods, hyphens, commas (in numbers), apostrophes, URLs, and multi-word units.
- For other languages, words are not separated by spaces, including Chinese (which uses logograms) and German (which uses many multiword compounds).
- Modern approaches also use machine learning methods.

Byte-Pair Encoding

- An approach to word tokenisation which builds words up from individual characters, rather than trying to break them down from the full sentence.
- The algorithm works by iteratively merging frequent pairs of characters in the corpus.
- For example, in the simple corpus below (consisting of 18 documents of a few characters each), the pairing ‘r _’ occurs most often (9 times in total), so we merge those two characters and add the result to our vocabulary.



- This process is repeated typically thousands of times, producing a large vocabulary.
- After training on the corpus, the most frequent words will be represented as full words, while rarer words will be broken into subwords. In the worst case, unknown words in test data will be broken into individual letters.

Word Normalisation

- The goal is to reduce the vocabulary size and clean the corpus by converting different forms of the same word to a uniform format.
- Lemmatisation means removing any inflection to reach the uninflected form, the lemma.
- Stemming involves removing all suffixes to leave a stem, which is not always an actual word.
- The porter stemmer is a popular algorithm stemmer for English, which consists of a series of rewrite rules applied in stages.

Stopword removal

- A list of words to be removed from the document.
- Typical in bag-of-word (BOW) representations.
- Not appropriate when sequence is important.
- Typically, these are high-frequency closed-class or function words.

Text classification

Text classification tasks

- Topic classification: classify a text into one or more topics.
- Sentiment analysis: classify text based on emotional sentiment.
- Native-language identification: classify text by first language of author.
- Textual entailment: identify logical relation of two sentences.

Possible features

- Word n-grams
- Word polarity lexicons
- POS parse tree
- Semantic embeddings
- Removal of stop words

Classification algorithms

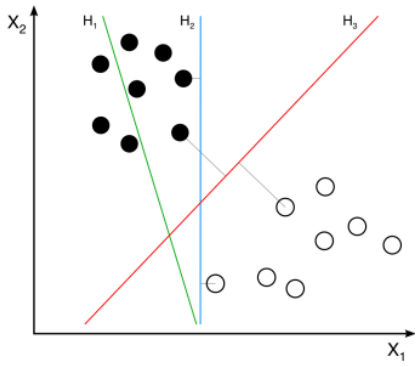
- Naïve Bayes: Finds the class with the highest likelihood under Bayes rule. Naively assumes that features are independent of each other.

$$p(C_n|f_1, f_2, \dots, f_n) = \prod_{i=1}^{N} p(f_1|C_n)p(C_n)$$

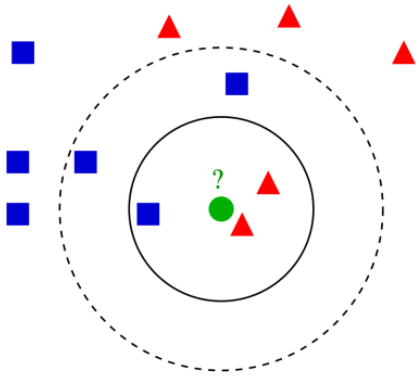
- Logistic regression: a linear combination of features to which softmax is applied to get a probability for each class. Training to maximise output prob subject to regularization.

$$p(C_n|f_1, f_2, \dots f_n) = \frac{1}{Z}\exp\left(\sum_{i=0}^{N} w_i f_i\right)$$

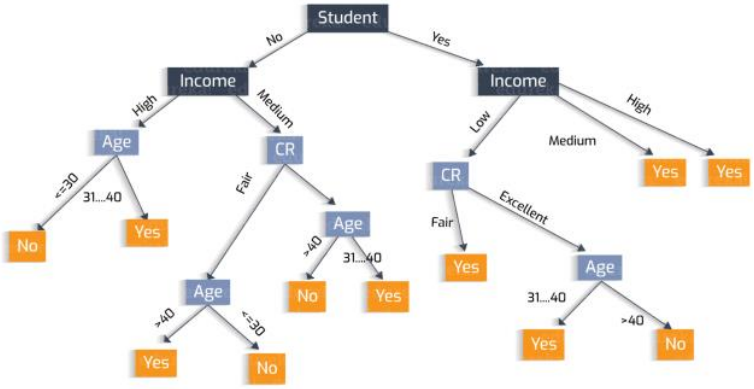
- Support vector machine: Finds hyperplane which separates the training data with a maximum margin.



- KNN: Classify based on majority class of k-nearest training examples in feature space.



- Decision tree: Construct a tree where nodes correspond to tests on individual features, and leaves are final class decisions.



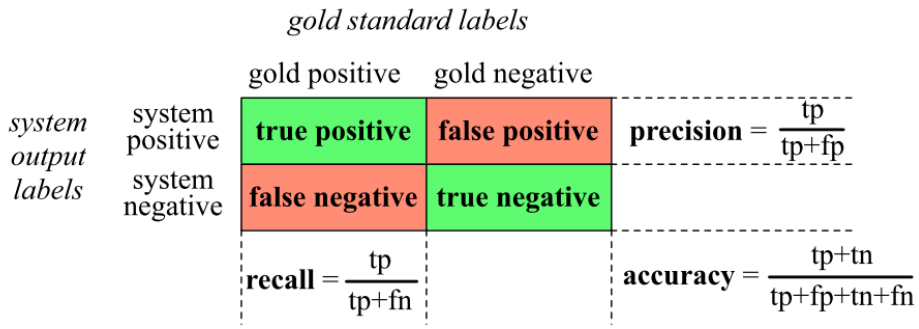
- Random forest: Consists of decision trees trained on different subsets of the training and feature space, where the final class decision is majority vote of sub-classifiers.

Table summary of methods

Classifier	Training speed	Complexity	Variance	Accuracy	Other issues
Naïve Bayes	Fast	Low	Low	Low	Independence rarely holds
Logistic Regression	Slow	Low	High	Moderate	Requires a lot of data to work well in practice
Support Vector Machine	Fast	Moderate	High	High	Multiclass classification awkward; feature scaling is critical
K-Nearest Neighbour	No training needed	Very low	Low	??	Need to select k; problems with imbalanced classes
Decision Tree	Fast	Low	??	Low	No need for feature scaling; hard to interpret in practise
Random Forest	Slower but can do in parallel	High	Moderate	Moderate	Hard to interpret; slow on large feature sets
Neural Network	Slow	High	High	High	Difficult to optimize many hyperparmeters

Evaluation and F1-score

Accuracy is a simple way to evaluate a classification model, but is it not very helpful when there are far more negative than positive labels (either because of many classes or imbalanced data). To give an extreme example, if 99% of the data is negative, a model that always predicts negative will get a 99% accuracy but have no predictive value at all. To instead focus on the predictive value of the model, we instead use recall and precision.



The F-measure is the weighted harmonic mean of precision and recall. The harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of reciprocals. We can write this as:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

Harmonic mean is used because it is a conservative metric; the harmonic mean of two values is closer to the minimum of the two values than the arithmetic mean is. Thus it weighs the lower of the two numbers more heavily.

N-gram models

Introduction to n-grams

Models that assign probabilities to sequences of words are called language model. The simplest class of models are n-gram models, which use sequences of n words using conditional probabilities:

$$P(w_1, w_2, \dots, w_n) = P\left(w_n \middle| w_1, \dots, w_{(m-1)}\right) \dots P(w_3|w_1, w_2)P(w_2|w_1)P(w_1)$$

Because this is intractable, we make a simplifying Markov assumption that only the past n words are relevant to this conditional probability. These probabilities can be estimated using counts (denoted C) of unigrams, bigrams, etc, from a corpus.

n-gram	Name	Joint probability	Estimate from corpus
1	Unigram	$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} P(w_i)$	$\frac{C(w_i)}{\sum_j C(w_j)}$
2	Bigram	$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$	$\frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
3	Trigram	$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1})$	$\frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}$

The start token

Note that we need to add special start and end of sequence tags to factor in the start of the sentence. So predicting the first word of a sentence using a trigram model would be:

$$P(w| < s >< s >)$$

Note that the start token is not included in the vocabulary count.

Smoothing techniques

Since we need the probabilities of all n-grams for an n-gram model, the higher n is, the more likely it becomes that we will encounter n-grams that are not present in our corpus. We thus need a method for estimating their probability.

Let V be the vocabulary set, excluding the start token. So $|V|$ is the number of types of words in the corpus.

Name	Explanation	Formula for $P(w_i|w_{i-1})$
Laplacian (add-one)	Add one count to each n-gram in the corpus.	$\frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + \|V\|}$
Lidstone (add-k)	Add one fractional count to each n-gram in the corpus. Need to choose $k < 1$.	$\frac{C(w_{i-1}w_i) + k}{C(w_{i-1}) + k\|V\|}$
Absolute discounting	Subtracts a fractional count from all seen n-grams, and adds it to all unseen n-grams equally (α).	$\begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})}, & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha, & \text{if } C(w_{i-1}w_i) = 0 \end{cases}$

Katz backoff	Subtracts a fractional count from all seen n-grams, and adds it to all unseen n-grams proportional to their probability relative to all words w_j that don't appear in the corpus after w_{i-1} $(w_j: C(w_{i-1}, w_j) = 0)$.	$\begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})}, \text{if } C(w_{i-1}w_i) > 0 \\ \alpha \frac{P(w_i)}{\sum_j P(w_j)}, \quad \text{if } C(w_{i-1}w_i) = 0 \end{cases}$
Kneser-Ney smoothing	Subtracts a fractional count from all seen n-grams, and adds it to all unseen n-grams proportional to the 'continuation probability', or the proportion of tokens w_{i-1} that co-occur with w_i. More versatile words get higher scores.	$\begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})}, \text{if } C(w_{i-1}w_i) > 0 \\ \alpha P_{cont}(w_i), \quad \text{if } C(w_{i-1}w_i) = 0 \end{cases}$ $P_{cont}(w_i) = \frac{\lvert\{w_{i-1}: C(w_{i-1}w_i) > 0\}\rvert}{\sum_j \lvert\{w_{j-1}: C(w_{i-1}w_j) > 0\}\rvert}$
Interpolation	Weighted average of different orders of n-gram models.	$\lambda_3 P_3(w_i \lvert w_{i-2}, w_{i-1}) + \lambda_2 P_2(w_i \lvert w_{i-1}) + \lambda_1 P_1(w_i)$

Model evaluation

Other things being equal, the best language model is the one which assigns the highest probability to a test set (a series of sentences it was not trained on). In practice we don't use raw probability as our metric for evaluating language models, but a variant called perplexity. The perplexity of a language model is the inverse probability of the test set, normalized by the number of words.

$$PP(W) = \left[\prod_{i=1}^{N} P(w_i|w_{i-1})\right]^{-\frac{1}{N}}$$

A perplexity of 1 would mean perfectly accurate predictions for each word. Higher values indicate worse predictions. If your model always assigns a fixed probability to each word in the vocab, the perplexity will be equal to the vocab size.

The perplexity of two language models is only comparable if they use identical vocabularies.

Part-of-Speech tagging

Part-of-speech classes

A part of speech is a category of words that have similar grammatical properties. Words that are assigned to the same part of speech generally display similar syntactic behavior and similar morphology in that they undergo inflection for similar properties.

Nouns, verbs, adjectives, and adverbs are 'open' classes accepts the addition of new words, while a closed class is one to which new items are very rarely added.

Part of speech identification is difficult because many words belong to multiple classes. Furthermore, there is no single set of classes, and so the tagset used must be selected before classification is possible. The Penn Treebank tags are commonly used.

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Automatic tagging

POS tagging is useful for several applications, including:

- Information retrieval (focus on nouns)
- Sentiment analysis (focus on adjectives)
- Word sense disambiguation
- Semantic or syntactic parsing

Simple tagger models:

- Rule-based: uses a list of manually crafted rules. Tend to be brittle and hard to construct.
- Unigram tagger: just apply the most common tag for each word. Surprisingly good.
- Classifier-based: use a standard approach like logistic regression to classify each token using the surrounding context of tokens. Suffers from error propagation if one mistake is made.

Hidden Markov models

A hidden Markov model works similarly to a classifier-based tagging method. The main difference is that with the correct training, it is possible to consider all possible combinations of tag assignments, thus circumventing the error propagation problem.

Given a sequence of tokens $\tilde{w}$, we want to choose the set of tags $\tilde{t}$ that maximises:

$$P(\tilde{t}|\tilde{w}) = P(\tilde{w}|\tilde{t})P(\tilde{t})$$

Using the Markov assumption that each probability only depends on the previous token or tag, this can be written as:

$$P(\tilde{t}|\tilde{w}) = \prod_{i=1}^{n} P(w_i|t_i) \prod_{i=1}^{n} P(t_i|t_{i-1})$$

The first term are the emission probabilities (word given token), and the latter term are the transition probabilities (state 2 given state 1). These probabilities are computed by counting word frequencies according to their tags, in some labelled corpus. Note that as before we need to introduce a start token <s>, and use smoothing techniques for unseen combinations. Example tables of transition and emission probabilities are given below.

	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

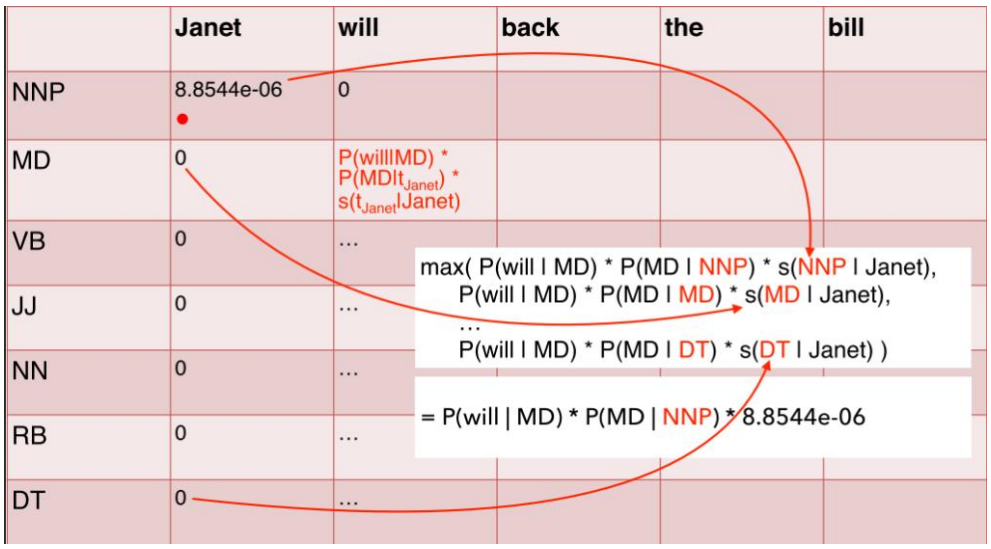
Figure 8.8 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

The difficulty with HMMs is to find the globally optimal prediction/decoding, not just the one that is optimal at each step. The correct way is to consider all possible tag combinations, evaluate them, take the max.

The Viterbi algorithm

This algorithm finds the optimal solution for tagging a sequence of tokens, given the available emission and transition probabilities. This works because at each step (i.e., for each column in the table below) we only need to consider the immediately preceding word. We are thus able to find the globally optimal solution without needing to compute all possible combinations of tags.

The Viterbi algorithm first sets up a probability matrix or lattice, with one column for each observation and one row for each state in the state graph. For example, the 'Janet' column below gives the conditional probabilities of Janet given each tag label.



To write down the value in a target cell in the second column, we first find the highest probability in the preceding column, then write down the conditional probability that links it to the cell of interest.

As an example, consider cell 2,2. The highest value in the preceding column is $L(t_1|w_1) = L(NNP|Janet) = L(Janet|NNP) \times P(NNP|\langle S \rangle)$, so to link this value to the target $L(w_2|t_2)$ we need $P(t_2|t_1)$. Putting together the pieces we have:

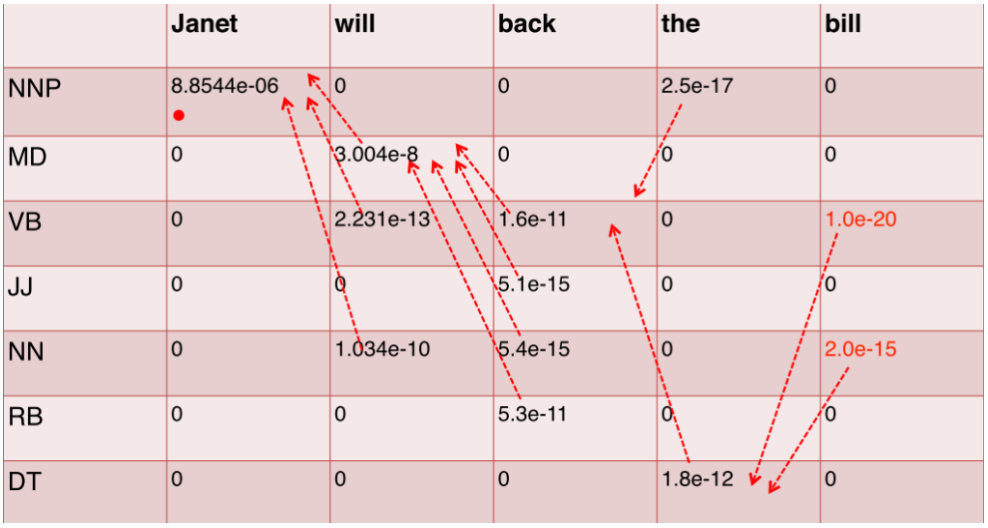
$$L(w_2|t_2) \times P(t_2|t_1) \times L(t_1|w_1) = L(will|MD) \times P(MD|NNP) \times L(NNP|Janet)$$

The first two numbers come from our corpus tables, and the final value comes from the previous column. We thus compute:

$$\begin{aligned} L(MD|will) &= L(will|MD) \times P(MD|NNP) \times L(NNP|Janet) \\ &= 0.3084 \times 0.0110 \times 8.8544 \times 10^{-6} \\ L(MD|will) &= 3.004 \times 10^{-8} \end{aligned}$$

After we finish each likelihood, we draw an arrow from that cell pointing back to the cell in the previous column that we used in the computation. In the above example, the arrow would point back to cell 1,1. Each cell in a column should point back to the same cell in the previous column.

If we repeat this process for each cell in the table above, we arrive at the following:



To find the optimal solution, we pick the highest number in the final column, and read back the arrows across the table. This gives the optimal set of tags (given the Markov assumption). The algorithm complexity is $O(T^2N)$, where T is the size of the tag set and N the length of the sequence.

2. Semantics and Deep Learning

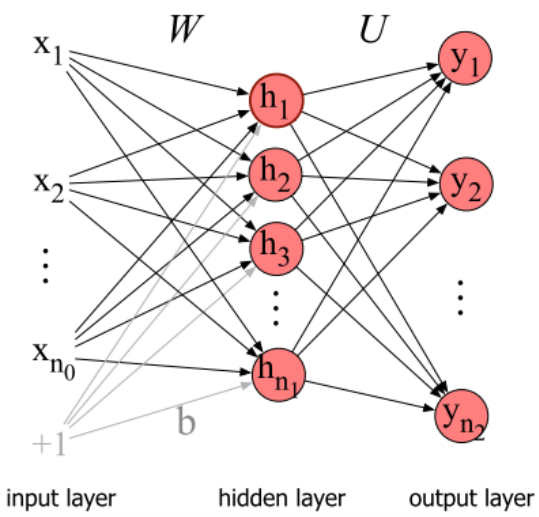
Neural networks

Feedforward neural networks

A neural network consists of a series of nodes connected via weights. Each node computes a scalar function of the activity receives from its input weights.

$$h = f\left(\sum_i w_i x_i + b\right)$$

Here f is the activation function, w_i are the weights, x_i are the input values, and b is the bias.



Hidden layers

As shown in the diagrams below, at least one hidden layer is necessary for computing non-linearly separable functions. In addition, it is essential that the activation function is non-linear, as multiple layers with linear activation functions are equivalent to a single layer. ReLU activation functions are commonly used as they are easily to compute derivatives for and are not susceptible to the diminishing gradient problem of sigmoid activation functions.

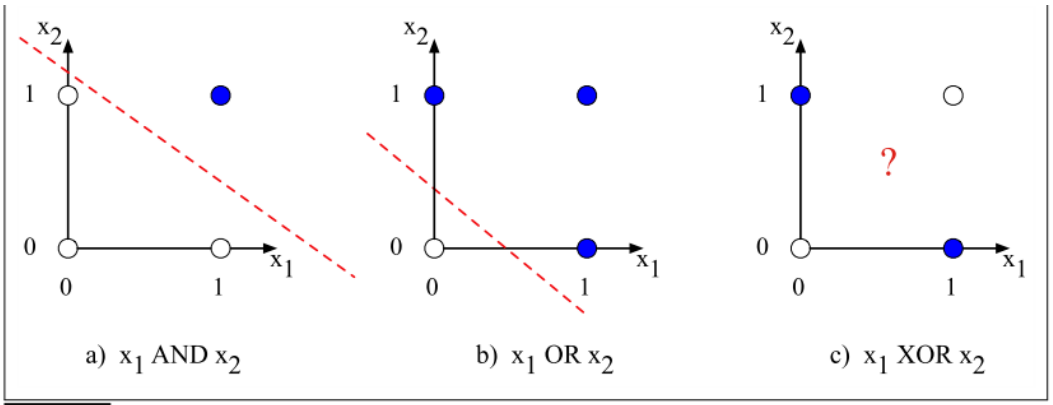
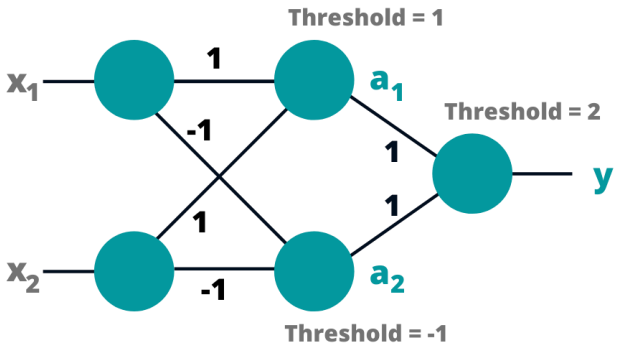
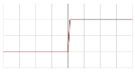
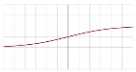
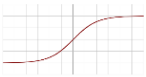
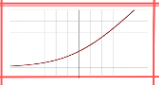
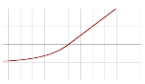


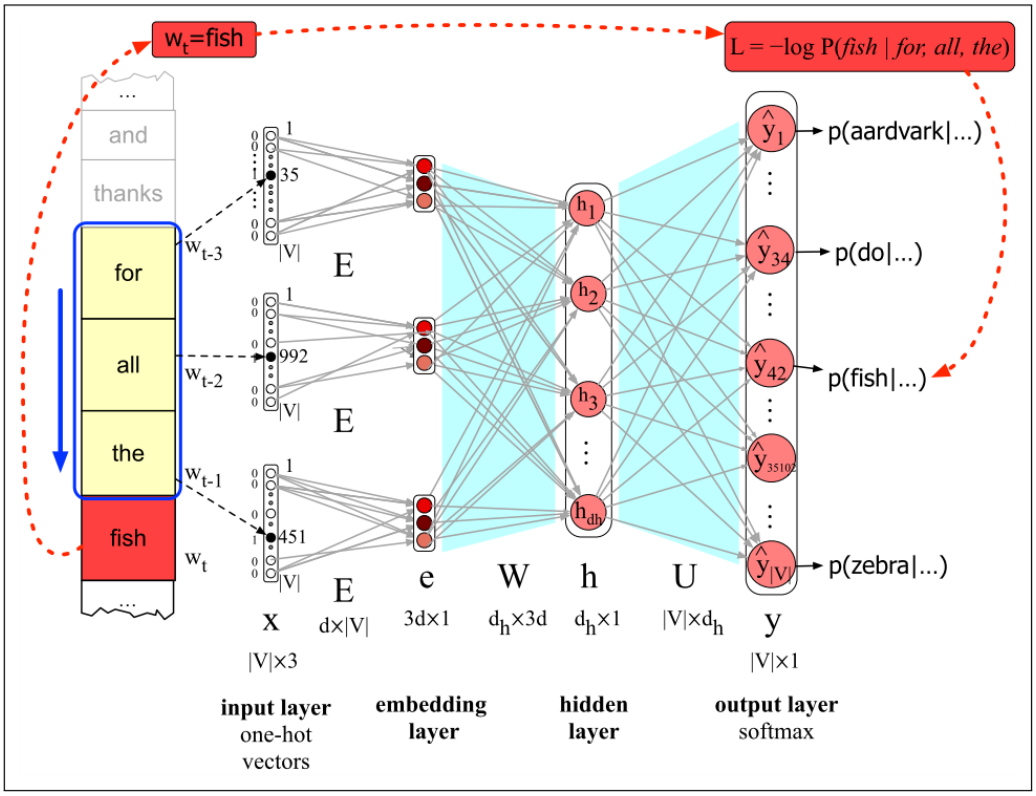
Figure 7.5 The functions AND, OR, and XOR, represented with input x_1 on the x-axis and input x_2 on the y axis. Filled circles represent perceptron outputs of 1, and white circles perceptron outputs of 0. There is no



Some common activation functions are shown below.

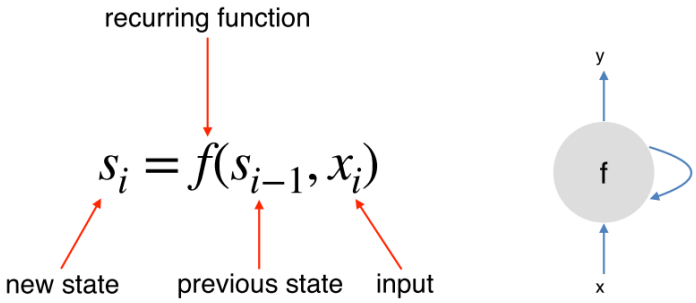
ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	(-∞, ∞)
Binary Step		f(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	{0, 1}
Sigmoid		f(x) = $\sigma(x) = \frac{1}{1+e^{-x}}$	f'(x) = $f(x)(1-f(x))$	(0, 1)
Hyperbolic Tangent(tanh)		f(x) = $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	f'(x) = $1 - f(x)^2$	(-1, 1)
Rectified Linear Unit(ReLU)		f(x) = $\begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	[0, ∞)
Softplus		f(x) = $\ln(1 + e^x)$	f'(x) = $\frac{1}{1+e^{-x}}$	(0, 1)
Leaky ReLU		f(x) = $\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	(-1, 1)
Exponential Linear Unit(ELU)		f(x) = $\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	f'(x)= $\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	[0, ∞)

Neural networks are trained to accurately predict the labels, using gradient descent. Dropout can be used to prevent overfitting. The diagram on the next page shows how this works.

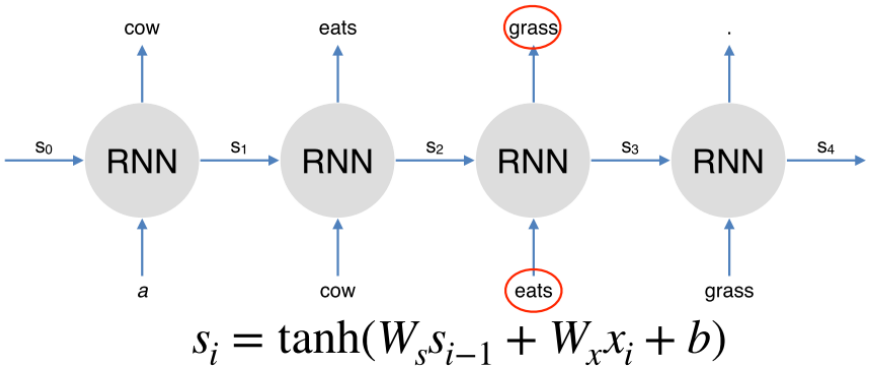


Recurrent neural networks

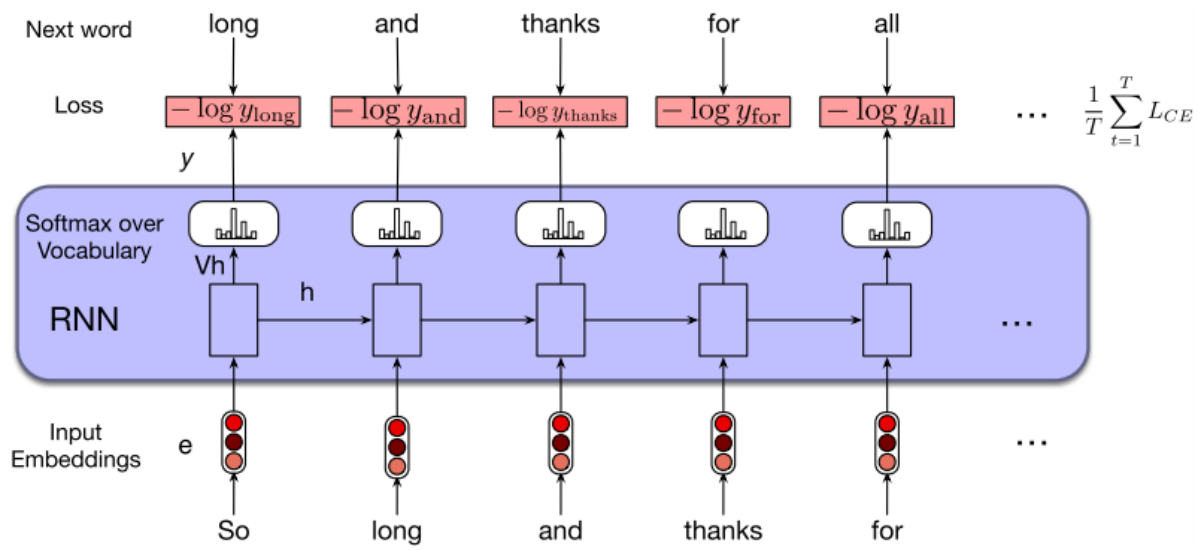
A recurrent neural network (RNN) is any network that contains a cycle within its network connections, meaning that the value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input. These are useful for language processing because they enable incorporating arbitrarily long sequences of context into computing the output.



We can 'unroll' an RNN to represent its output in a linear fashion, representing previous states of the network as if they were additional layers. The example shown below is performing a next-word prediction task.



To train RNN, we just need to create the unrolled computation graph given an input sequence. We use the backpropagation algorithm to compute gradients as usual, with fixed weights for each layer (time step). This procedure is called backpropagation through time. We can use this method to develop language models using RNNs.



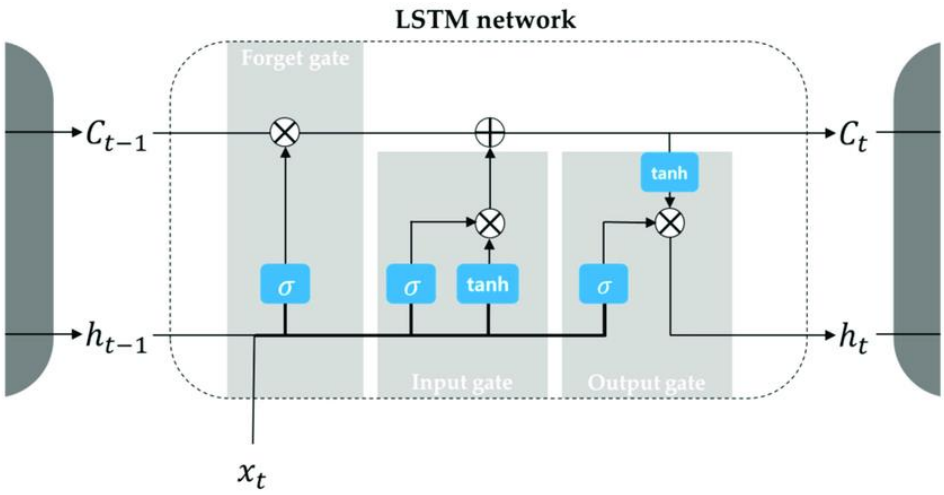
Long short-term memory

RNN has the capability to model infinite context, but in practice it is unable to capture long-range dependencies due to the vanishing gradient problem. Gradients in later steps diminish quickly during backpropagation, and hence earlier inputs do not get much update with training. LSTM is introduced to solve vanishing gradients. The core idea is to have 'memory cells' that preserve gradients across time. Access to the memory cells is controlled by 'gates'.

The LSTM has three gates, each controlling a different flow of information:

- Forget gate: purpose is to delete information that is no longer needed.
- Input gate: select information to add to the current context that is needed.
- Output gate: select which information to use to generate the current output.

The LSTM takes in the context c_{t-1}, the previous output h_{t-1}, and the new input x_t, and outputs the updated context c_t and the new output h_t. The gates control the way in which the two outputs are computed from the three inputs.



Each gate computes its outputs by concatenating its inputs, multiplying them by a weight matrix, and then passing the result through a sigmoid function to convert the result to a number between 0 and 1.

$$f_t = \sigma\big(W_f \cdot [h_{t-1}x_t] + b_f\big)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}x_t] + b_o)$$

Once we have computed all the gates, we need to update the context information, adding the new input and removing the forget data. Note that here we use pointwise multiplication $*$ and a tanh activation function over the new input:

$$C_t = f_t * C_{t-1} + i_t * \tanh(W \cdot [h_{t-1}x_t] + b)$$

Now that we have updated the context, all we need to do is use the output gate to select the parts we want to return as output:

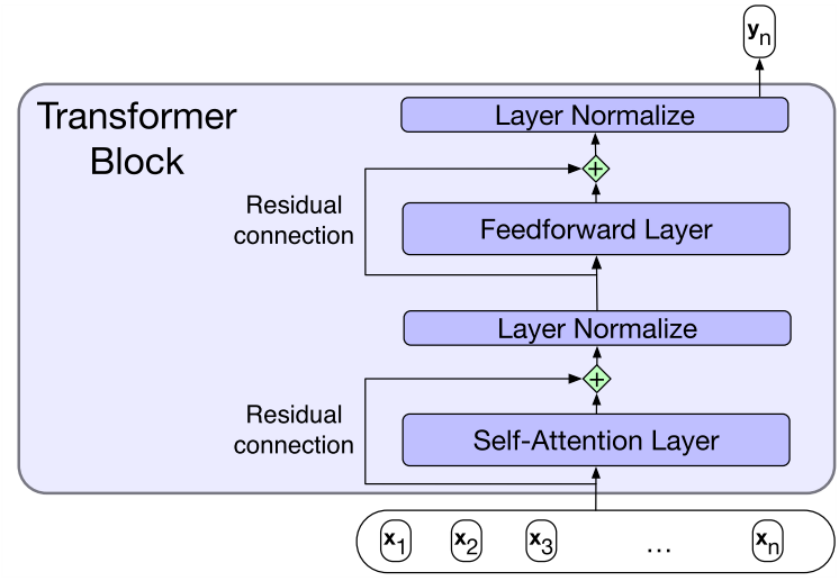
$$h_t = o_t * \tanh(C_t)$$

LSTMs can capture more context than RNNs, but still fails with very long-range dependencies. They are also much slower to train.

Transformer architecture

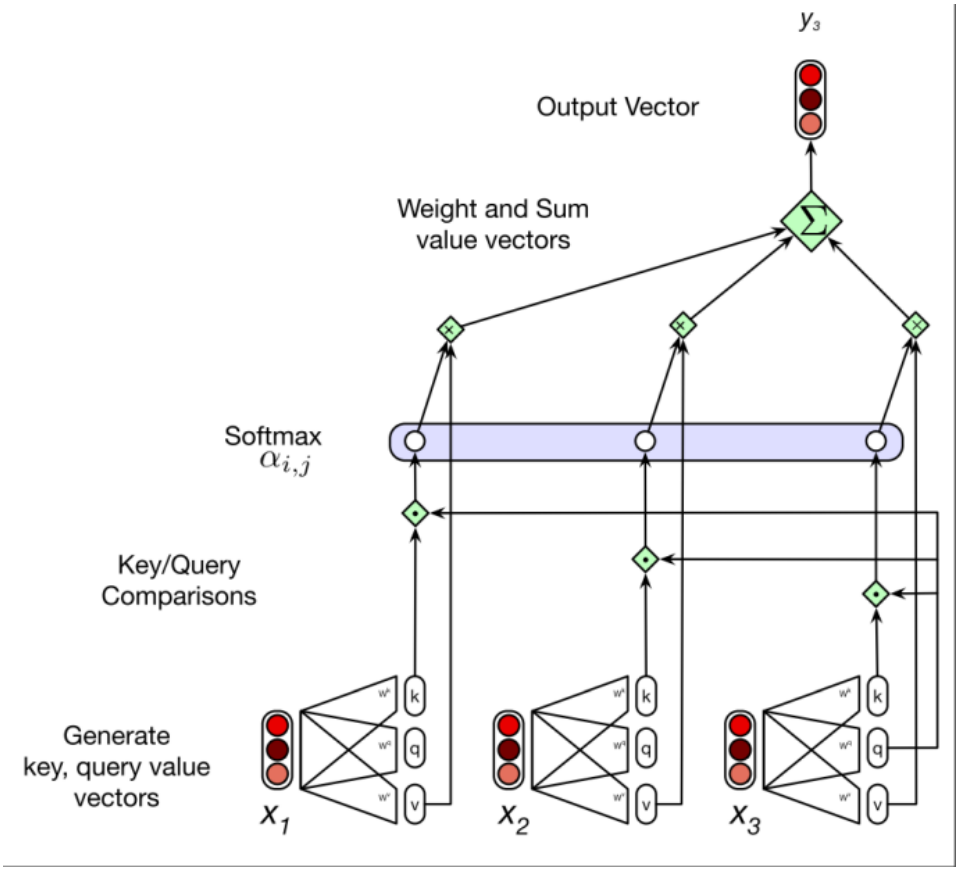
Passing information through an extended series of recurrent connections leads to information loss and difficulties in training. Moreover, the inherently sequential nature of recurrent networks makes it hard to do computation in parallel. These considerations led to the transformers development of transformers – an approach to sequence processing that eliminates recurrent connections.

Transformers are made up of stacks of transformer blocks, which are multilayer networks made by combining simple linear layers, feedforward networks, and self-attention layers, the key innovation of transformers. Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs. The key architecture of a transformer block is shown in the diagram below.



Residual connections allow information to be sent directly up to higher layers, skipping the intermediate layer. This improves learning and allows the model greater flexibility.

The heart of the transformer is the self-attention layer. This is summarized in the diagram below.



To compute the output for input embedding x_3, using a window of size 3 as an example, we first compute three different linear transformations of the input embeddings, to produce key, query, and value embeddings for each input token. These are computed as:

$$\begin{aligned} q_1 &= W^Q x_1, \quad & q_2 &= W^Q x_2, \quad & q_3 &= W^Q x_3 \\ k_1 &= W^K x_1, \quad & k_2 &= W^K x_2, \quad & k_3 &= W^K x_3 \\ v_1 &= W^V x_1, \quad & v_2 &= W^V x_2, \quad & v_3 &= W^V x_3 \end{aligned}$$

Note that the W^Q, $\mathrm{W^K}$, and W^V matrices are learned during training.

The output y_3 is computed as a weighted sum of the value vectors:

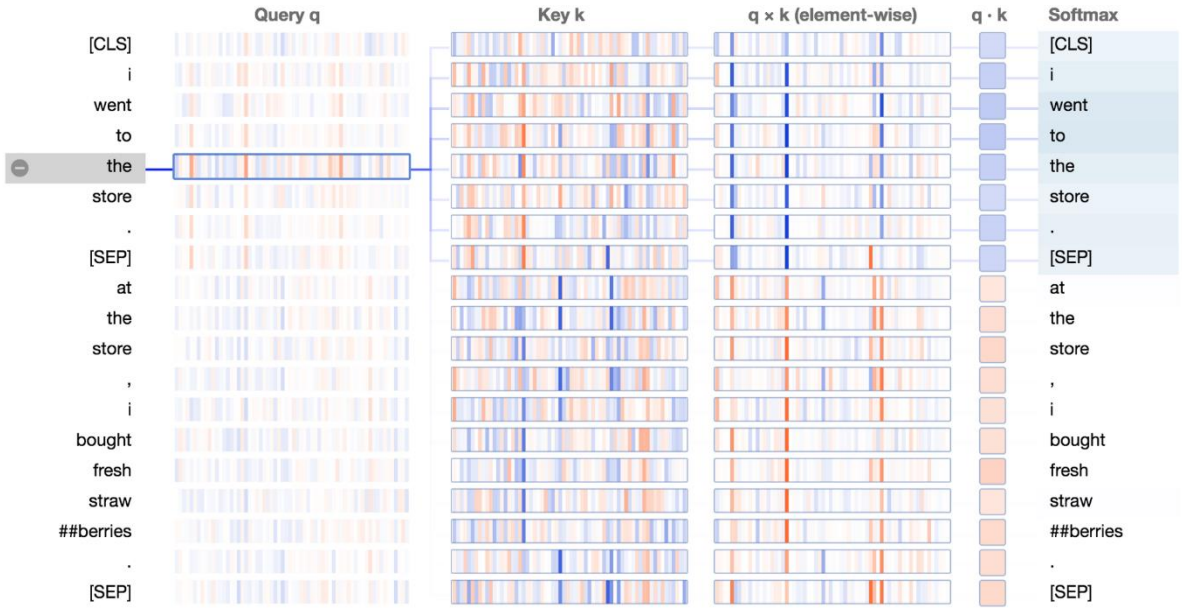
$$y_3 = \sum_{i=1}^{3} \alpha_{3i}\, v_i$$

However, to evaluate this we first need to know the weights α_i. These are calculated by taking the dot product of the relevant query vector with each value vector. This case, we want to compute the output for x_3, so the relevant query vector is q_3, and hence we do not need q_1 or q_2 for the moment. Hence we have:

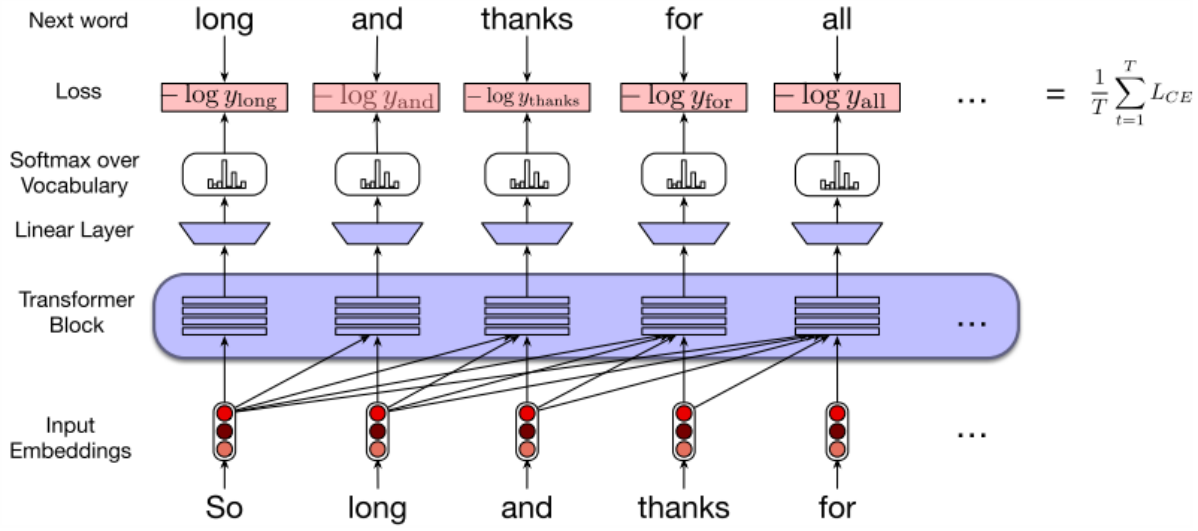
$$\alpha_{31} = k_1 \cdot q_3, \qquad \alpha_{32} = k_2 \cdot q_3, \qquad \alpha_{33} = k_3 \cdot q_3$$

Putting all these steps together and writing the result in terms of the input embeddings we have:

$$y_3 = \alpha_{31}(W^V x_1) + \alpha_{32}(W^V x_2) + \alpha_{33}(W^V x_3)$$
$$y_3 = (k_1 \cdot q_3)(W^V x_1) + (k_2 \cdot q_3)(W^V x_2) + (k_3 \cdot q_3)(W^V x_3)$$
$$y_3 = (W^K x_1 \cdot W^Q x_3)(W^V x_1) + (W^K x_2 \cdot W^Q x_3)(W^V x_2) + (W^K x_3 \cdot W^Q x_3)(W^V x_3)$$



Transformers can be used as a language model by stacking several transformer blocks on top of one another, and then performing a softmax at the end to produce an output token.



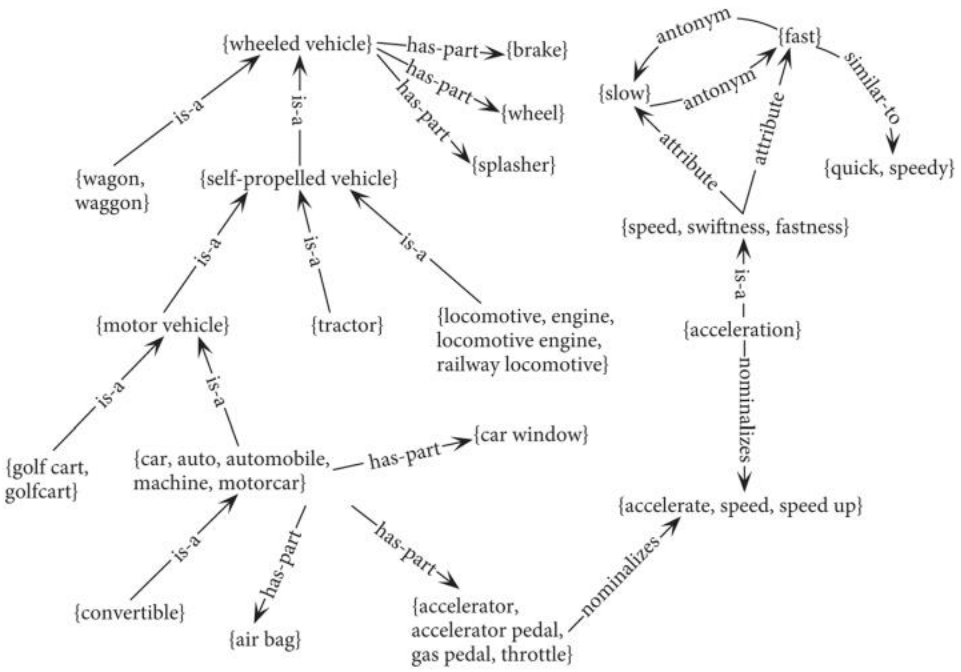
Lexical semantics

Meaning relations

One way to characterize word meaning is to specify the semantic relations between different words.

- Synonymy: similar meaning
- Antonymy: opposite meaning
- Hypernymy: is-a relation
- Meronymy: part-whole relation

Wordnet is a large lexical database of such semantic relations, depicted as a graph.



Word similarity

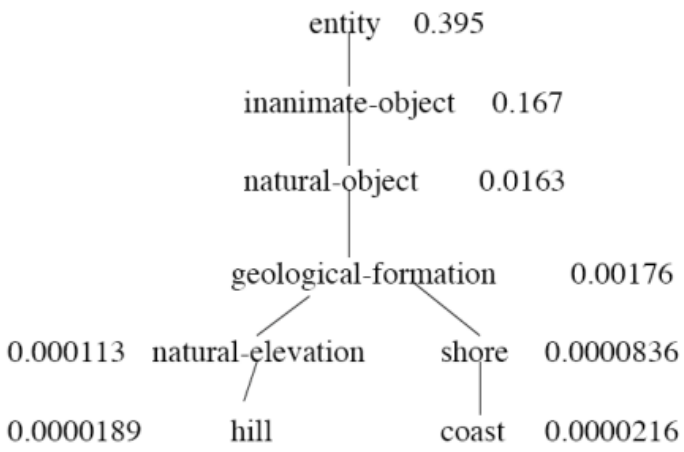
Similarity between words can be computed from wordnet based on the path length between two words. This is computed as the reciprocal of one plus the number of edges between the nodes:

$$\text{simpath}(c_1, c_2) = \frac{1}{1 + \text{edgelen}(c_1, c_2)}$$

However simple path length does not incorporate the fact that there are much bigger 'jumps' in meaning between nodes near the top of the hierarchy than at the bottom. To control for this, we can incorporate the depth in the tree of the lowest common subsume (parent) of the two words:

$$\text{simwup}(c_1, c_2) = \frac{2 \times \text{depth}\big(LCS(c_1, c_2)\big)}{\text{depth}(c_1) \times \text{depth}(c_2)}$$

Directly counting the number of nodes, however, is still not a very useful measure of distance. Instead, we can use the information content (negative log of probability) for each node in the tree, as computed by the sum of the unigram probabilities for all children of the node. Higher nodes thus will have higher probabilities than lower nodes.



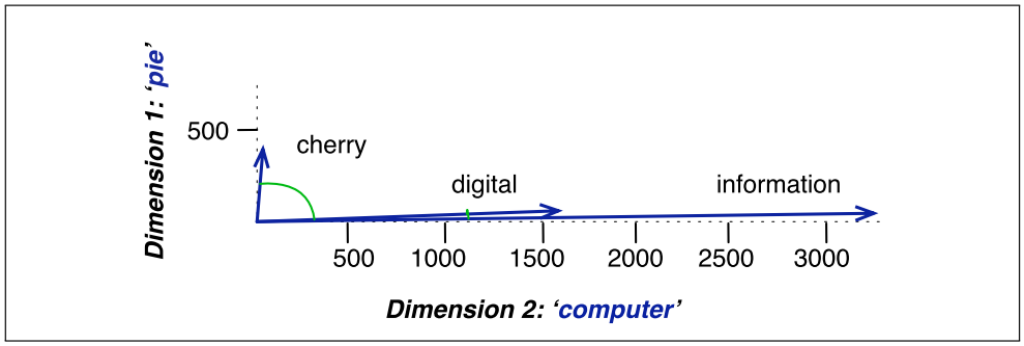
We thus compute the information content similarity (simlin) as:

$$\text{simlin}(c_1, c_2) = \frac{2 \times \text{IC}\big(LCS(c_1, c_2)\big)}{\text{IC}(c_1) \times \text{IC}(c_2)}$$

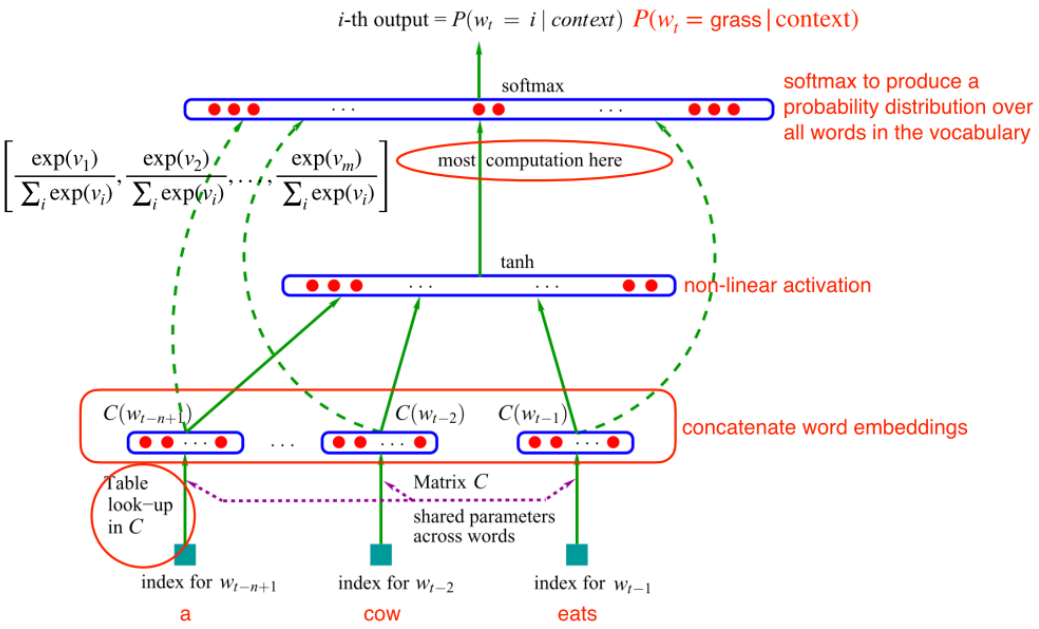
$$IC(c) = -\log\left(\sum_{s \in \text{child}(c)} \frac{\text{count}(s)}{N}\right)$$

Word embeddings

Neural networks can represent words by using word embeddings. Maps discrete word symbols to continuous vectors in a relatively low dimensional space. Word embeddings allow the model to capture similarity between words in a graded manner.



To encode a sentence for input into a neural network, we look up the word embedding for each word in the sentence, concatenate these embeddings, and that becomes the input vector to the network. We can then train the network on a variety of tasks, such as sentiment classification, topic modelling, or next-word prediction. This process is summarized in the diagram below.



Constructing word embeddings

Latent Semantic Analysis

Tf-idf is a method for computing a word-document cooccurrence matrix, which can then be used to construct a set of word embeddings.

A term frequency (tf) matrix counts the frequency of each word in each document. By itself this isn't very useful for constructing word embeddings, since the counts are dominated by very common, uninformative words. We can use another metric, inverse document frequency (idf) to calculate the inverse proportion of documents that contain the word in question. The more documents include the word, the less informative it is. We can represent these terms for term t and document d as:

$$\text{tf}(t,d) = \log(\text{count}(t,d) + 1)$$
$$\text{idf}(t) = \log\left(\frac{D}{df(t)}\right)$$

Where df is the number of documents where term t occurs at least once. We combine tf and idf as:

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t)$$

An alternative method for obtaining cooccurrence matrices is to use pointwise mutual information (PMI). This is a measure of the discrepancy between the joint distribution of words x and y, and their individual distributions. It is calculated as:

$$\text{PMI}(x,y) = \log\left(\frac{P(x,y)}{P(x)P(y)}\right)$$

PMI often captures semantic relations better than tf-idf, however it is very sensitive to rare word combinations.

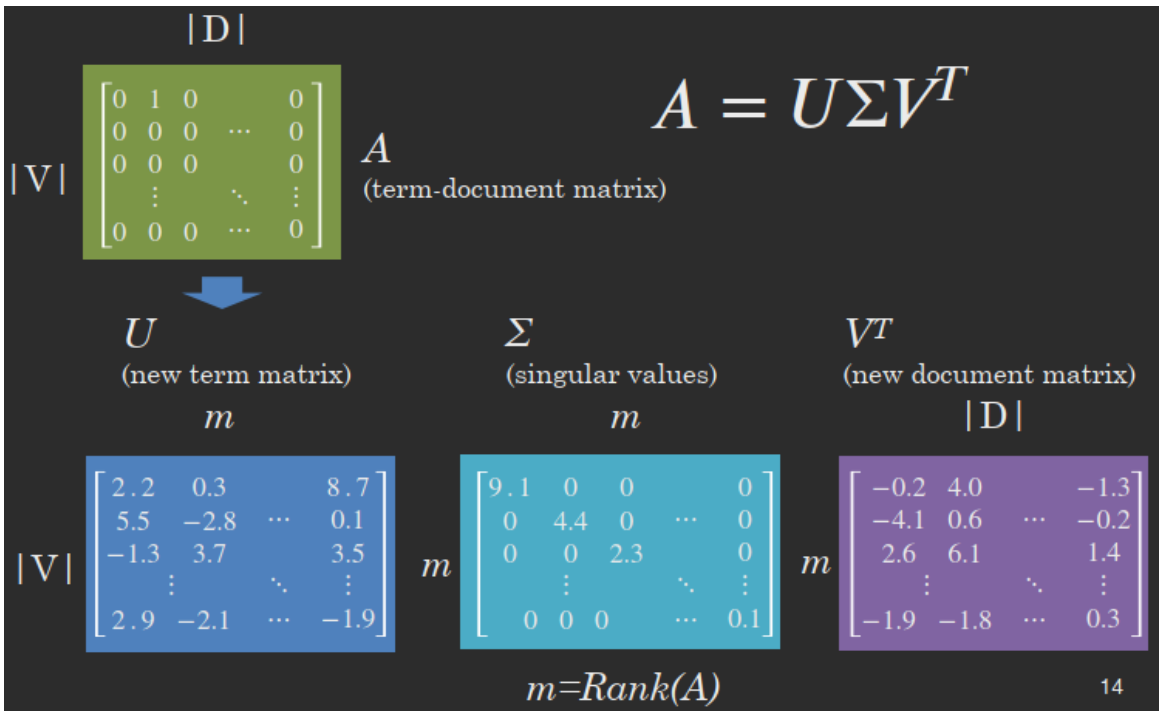
	...	the	country	hell	...
...					
425		0	26.0	6.2	
426		0	5.2	0	
427		0	0	18.6	
...					

tf-idf matrix

	...	the	country	hell	...
...					
state		1.22	1.78	0.63	
fun		0.37	3.79	0	
heaven		0.41	2.80	6.60	
.....					

PPMI matrix

Whether we use tf-idf or PMI, we can convert the matrix into a set of word embeddings using a dimensionality reduction technique known as singular value decomposition (SVD).

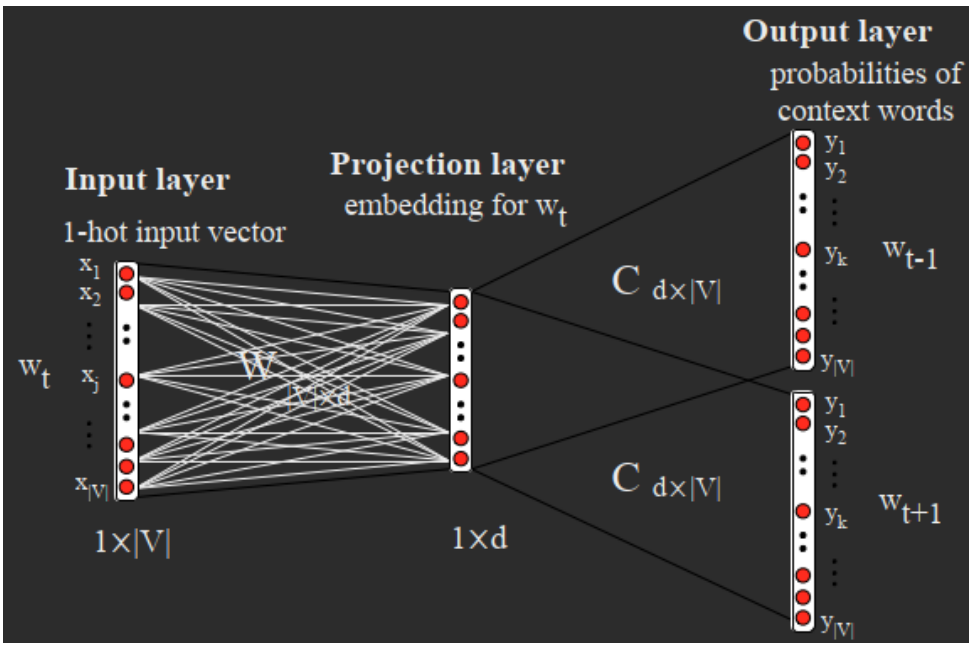


The new term matrix U is truncated down to k columns (usually a few hundred or thousand), yielding the word embeddings as the rows of the truncated U matrix.

Word2Vec

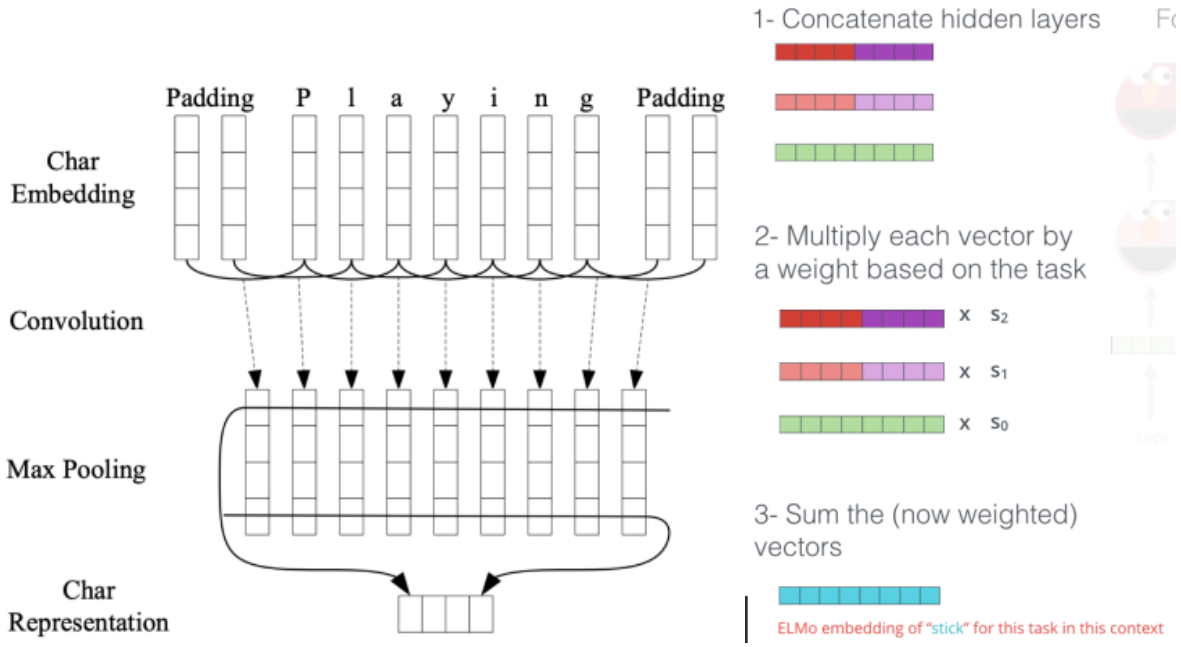
A completely different approach to learning word embeddings is to train a neural network to predict a target word using surrounding words. The weights learned in the hidden layer are used to construct word embeddings.

Skipgram is a very similar model, except that is predicts surrounding words of target word. It is trained as a binary classification task, judging whether a candidate is a true or a false example of a context word for the given target word. Its task is to maximise similarity between target word and real context words, and also minimise similarity between target word and non-context words.



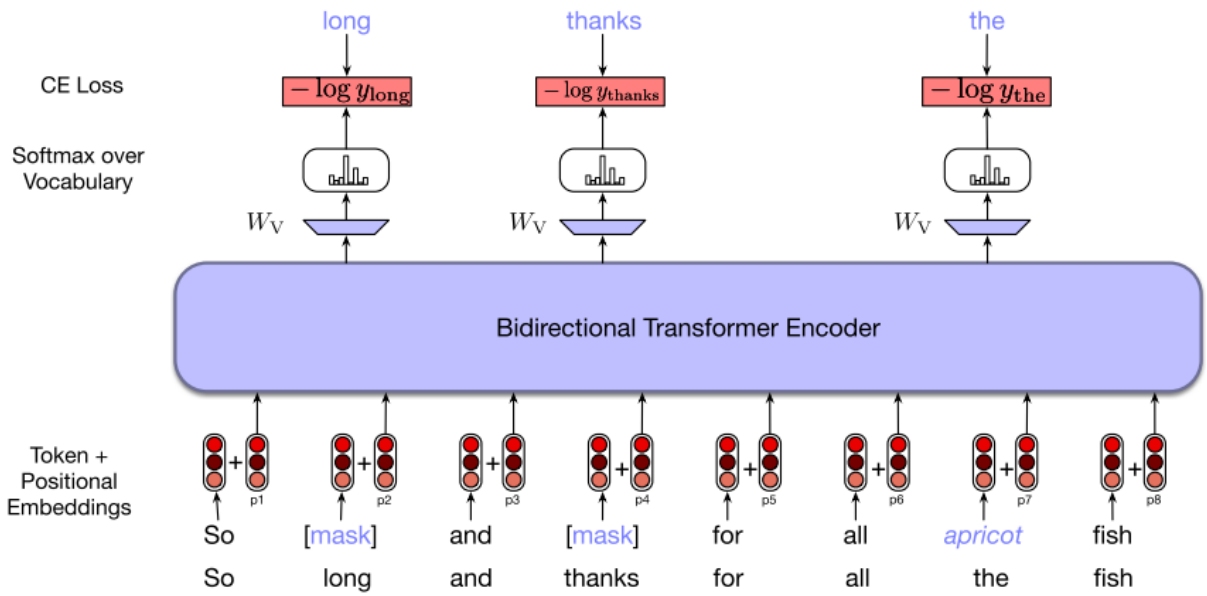
ELMo embeddings

ELMo is a model that trains a bidirectional, two-layer LSTM language model over a 1B word corpus. Word embeddings are constructed using character convolutional networks from individual token embeddings. Multiple hidden states from the LSTM are combined to yield the final embeddings.



BERT architecture

BERT is a transformer model which uses self-attention instead of recurrent connections to capture dependencies between words. Instead of predicting the next word, it is trained to predict randomly masked words from the input.



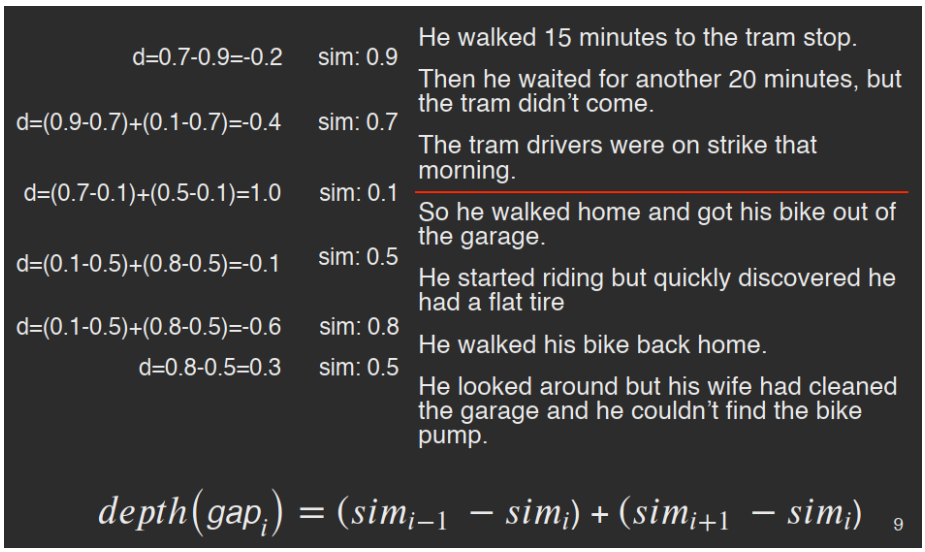
The embeddings obtained from the pre-trained BERT model are typically used for downstream tasks. This is done by adding a classification layer on top of BERT.

Discourse modelling

Discourse segmentation

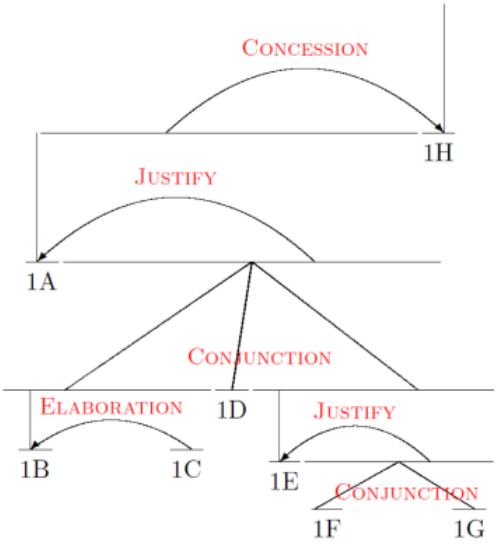
- The purpose of discourse modelling is to understand how sentences relate to each other in a document.
- Often the focus is on segmenting the document into sections, where a segment is a span of cohesive text within a document.
- The TextTiling algorithm is an unsupervised technique that looks for points of low lexical cohesion between sentences, as calculated by cosine similarity of BOW vectors of neighboring sentences.
- A 'depth' score is calculated between every two successive sentences, as a measure of when similarity dips. A boundary is inserted whenever the depth is greater than some threshold.

$$\text{depth}(gap_i) = (sim_{i-1} - sim_i) + (sim_{i+1} - sim_i)$$



Discourse parsing

- The goal of discourse parsing is to identify discourse units, and the relations that hold between them.
- It uses Rhetorical Structure Theory (RST), which is a framework to do hierarchical analysis of discourse structure in documents.
- Each discourse unit is a clause in a sentence, so a single sentence may have one or more discourse units.
- The goal is to classify the relations between discourse units into classes such as: conjuction, justify, concession, elaboration.
- Within a discourse relation, one argument is the nucleus (the primary argument), and the other (supporting) argument is the satellite. Some relations are equal (e.g. conjunction), and so both arguments are nuclei.
- An RST relation combines two or more DUs into composite DUs. As this process is repeated, an RST tree is gradually constructed.
- RST parsing can be done using explicit rule-based methods, using machine learning approaches, and using bottom-up greedy or top-down progressive segmenting approaches.



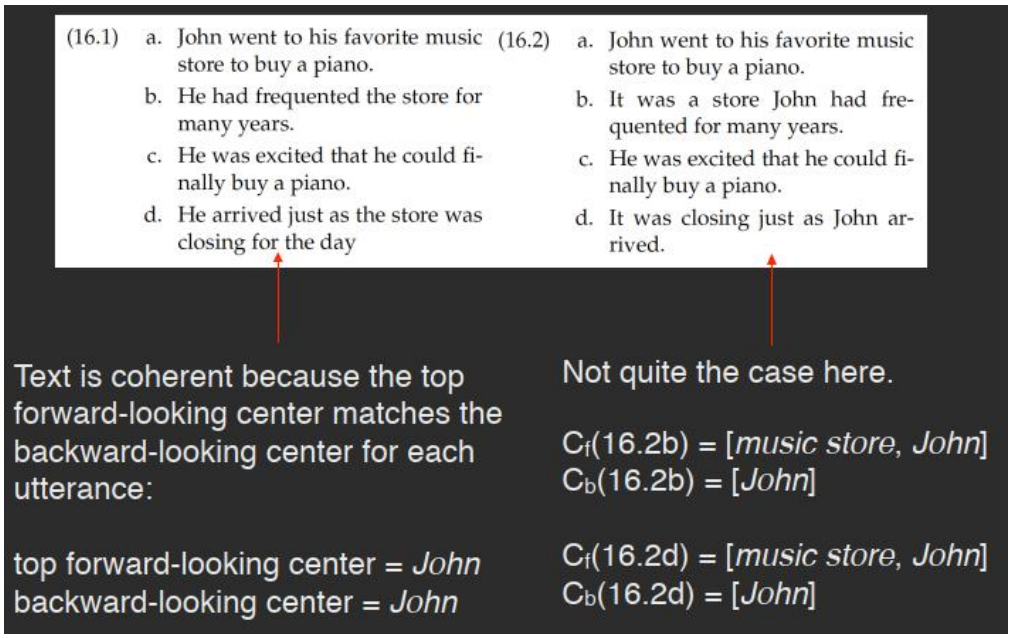
Anaphora resolution

- An anaphor is a linguistic expression that refers back to earlier elements in the text.
- For example, in the passage ‘Yesterday, Ted was late for work. It all started when his car wouldn’t start.’, the word ‘it’ refers back to the event of Ted being late for work.
- Pronouns are the most common form of anaphor, but many other types are possible.
- Various restrictions between an anaphor and its antecedent:
    - Words must agree in number with their antecedent.
    - Pronouns must agree in gender with their antecedents.
    - Pronouns whose antecedents are the subject of the same syntactic clause must be reflexive (e.g. myself).
    - Antecedents of pronouns should be recent in the passage.
    - The antecedent should be salient, as determined by grammatical position: Subject > object > argument of preposition.

Centering theory

Centering theory describes the relationship between discourse structure and entity reference. Every utterance is characterised by a set of entities, known as centers.

- Some centers are forward-looking, while others are backward-looking. When resolving entity for anaphora resolution, choose the entity such that the top forward-looking center matches with the backward-looking center.
- Anaphora resolution can be achieved by training a binary classifier for anaphor/antecedent pairs, converting restrictions and preferences from centering theory into features.

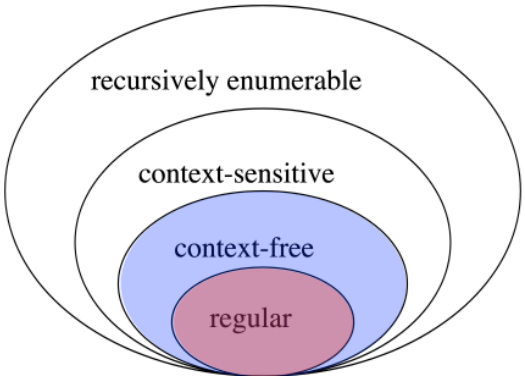


3. Syntax and Context Free Grammars

Finite state automata

Formal language theory

- Formal language theory studies classes of formal languages and their properties.
- A formal language is a set of strings, where a string is a sequence of elements from a finite alphabet of symbols.
- In formal language theory, strings do not have any 'meaning', but must meet certain criteria to be classed as members of a given language.
- The main goal is to solve the membership problem: is a given string a member of the language?



- Some examples of formal languages with two strings assessed for membership status is shown below.

- Binary strings that start with 0 and end with 1
  - { 01, 001, 011, 0001, … } ✓
  - (1, 0, 00, 11, 100, … } ✗
- Even-length sequences from alphabet {*a*, *b*}
  - { *aa*, *ab*, *ba*, *bb*, *aaaa*, … } ✓
  - { aaa, aba, bbb, … } ✗
- English sentences that start with *wh*-word and end in *?*
  - { *what ?*, *where my pants ?*, … } ✓
  - { *hello how are you?*, *why is the dog so cute!* }

Regular languages

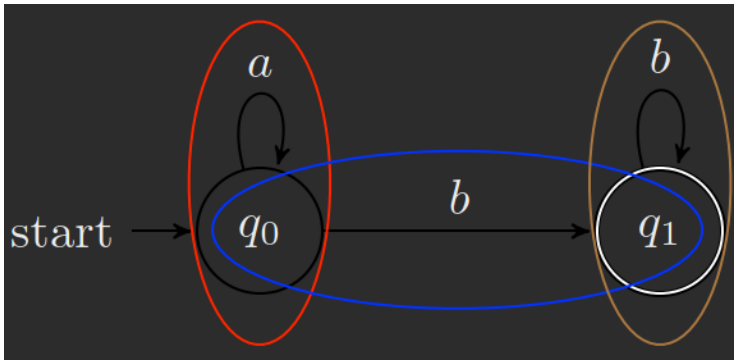
- A regular language is a member of the simplest class of languages, which are those whose strings can be matched by a regular expression.
- Formally, a regular expression includes the following operations/definitions:
  - A symbol drawn from alphabet, Σ
  - The empty string, ε
  - Concatenation of two regular expressions, RS
  - Alternation of two regular expressions, R|S

    - Kleene star operator for 0 or more repeats, R*
    - Parenthesis () to define scope of operations
- A set is 'closed' under an operation when applying the operation to any element of the set always yields another member of the set.
- The set of regular languages is closed under the following operations (i.e. performing these operations produces a new regular language):
    - Concatenation: all possible concatenations of pairs of strings from L1 and L2
    - Union: strings that are valid in L1 or L2
    - Intersection: strings that are valid in both L1 and L2
    - Negation: strings that are not in L

Finite state acceptor

- A finite state acceptor is an algorithm/formalism for determining whether a given string belongs to a particular regular language.
- A finite state acceptor consists of:
    - An alphabet of input symbols, Σ
    - A set of states, Q
    - A start state, q0 ∈ Q
    - One or more final states, F ⊆ Q
    - A transition function: symbol and state → next state
- An FSA accepts a string if there is path from q0 to a final state with transitions matching each symbol in the input string.
- For example, the FSA shown below accepts strings consisting of zero or more 'a's, followed by at least one 'b'. This is denoted: a^*b+

• Input alphabet	{a, b}	aab ✓ b ✓ aaa ✗
• States	{q0, q1}	
• Start, final states	q0, {q1}	
• Transition function	{(q0,a) → q0, (q0, b) → q1, (q1,b) → q1}	

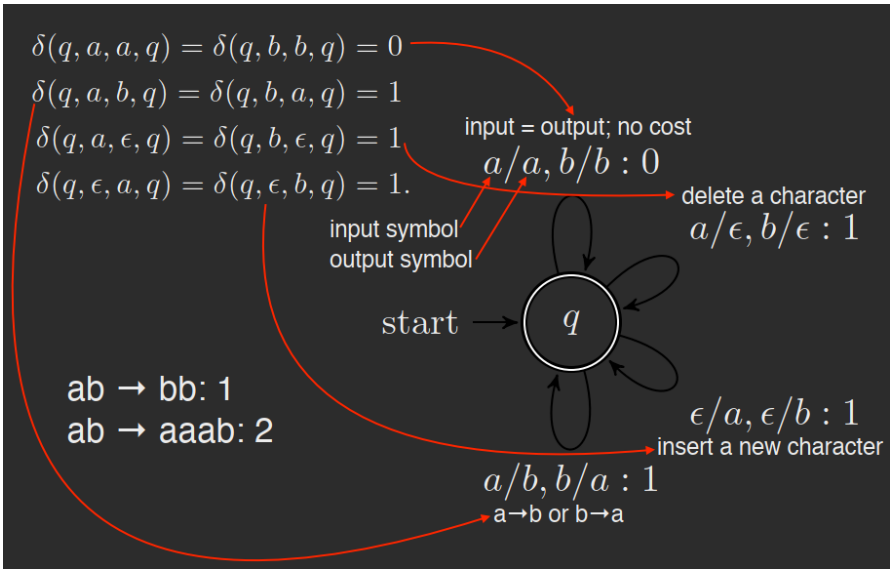


- Weighted FSAs can be constructed which have a probabilistic transition function and a probability distribution over state and end states. They will accept any string, but with varying 'degrees of acceptability'.

- Djisktra's shortest-path algorithm can be used to find the shortest acceptable path (if it exists) for a given string.

Finite state transducer

- A finite state transducer does not simply determine whether a string is accepted or not. Instead, it also produces a new string given each input string.
- It has the same structure as an FSA, but also includes an output alphabet.
- FSTs can be used to calculate the distance to transform one string to another (edit distance).
- An example of a simple finite state transducer is shown below.



Context free grammars

Beyond regular languages

- Some natural language expressions cannot be expressed as sets of strings acceptable by any regular language.
- For example, arithmetic expressions with balanced parentheses can have arbitrarily many opening parentheses. This number needs to be 'remembered' to produce the same number of closed parentheses. This can't be done with a finite number of states.
- Center embedding of relative clauses is an example of another type of structure which requires remembering the number of levels of embedding, which cannot be done in the general case with a finite number of states.
- Such expressions require a more expressive type of language, called a context-free grammar.

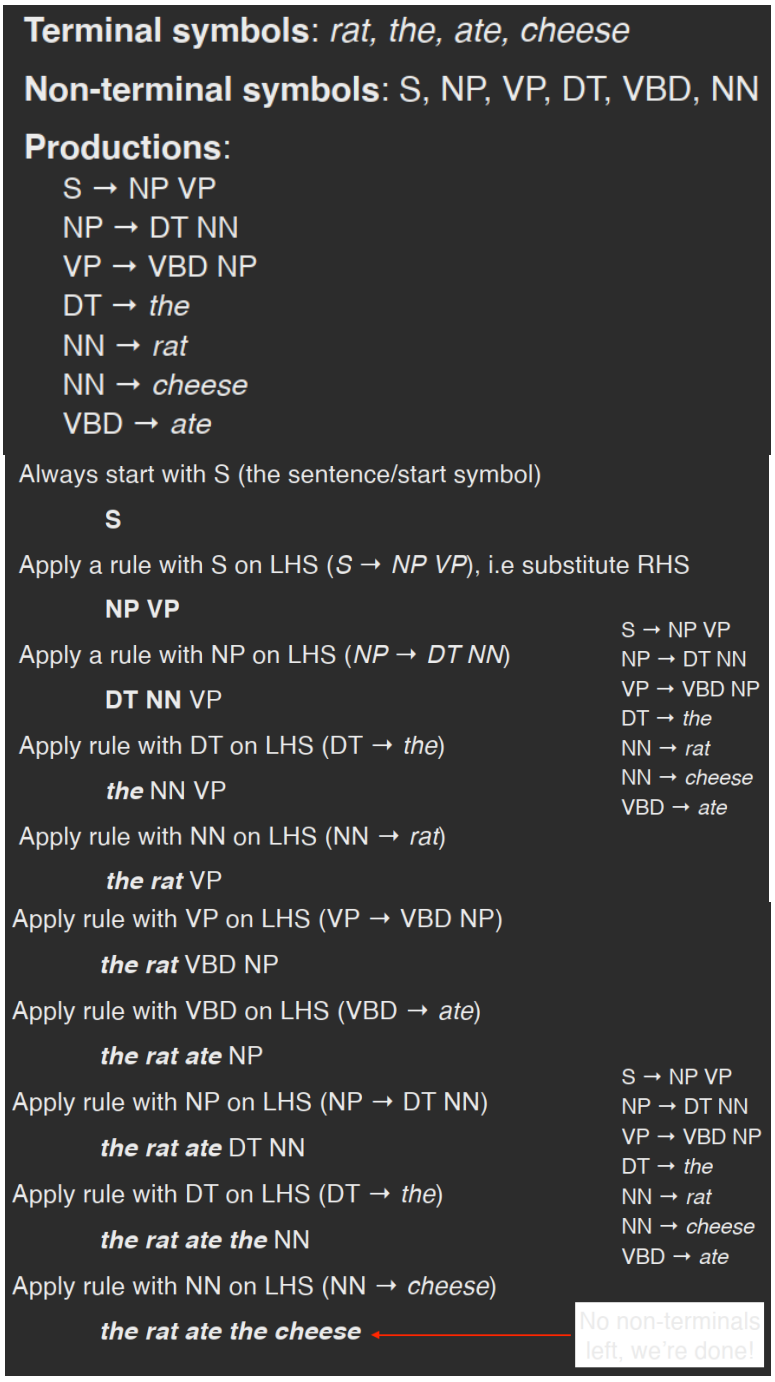
Basics of CFGs

- A context-free grammar consists of a set of rules, each of which expresses the ways that symbols of the language can be grouped together, and a lexicon of words and symbols.
- Terminal: word such as book, written in lowercase.
- Non-terminal: syntactic label such as NP or VP, written in uppercase.
- Rules: contain exactly one nonterminal on the righthand side, and an ordered list of any symbols on the lefthand side.
- Each production rule can only depend upon its lefthand side, and not on ancestors or neighboring rules. This is analogous to a Markov chain.

- Natural language is not quite context free, but the expressions which cannot be captured by CFGs are rare, and the extra complexity required to capture them is seldom worthwhile.

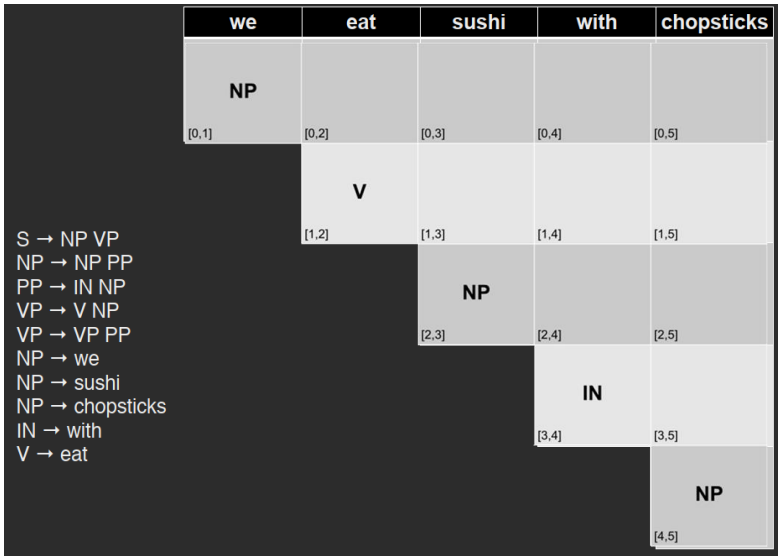
Constituents

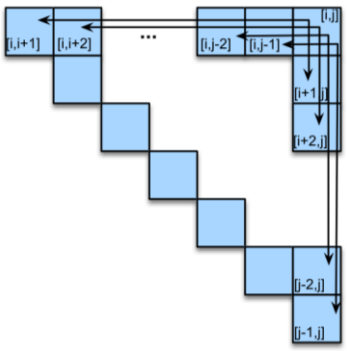
- Sentences are broken into constituents, which is a sequence of words that function as a coherent unit for linguistic analysis.
- Constituents can be identified as they can be moved as a unit from one part of a sentence to another. The meaning of the sentence may chance, but it will often still be grammatical.
- Constituents can also be substituted by other phrases of the same type, again changing the meaning but remaining grammatical.
- Finally, constituents can be coordinated using coordinators (conjunctions) like or and and.
- An example of a sentence being generated from a simple CFG is shown below:

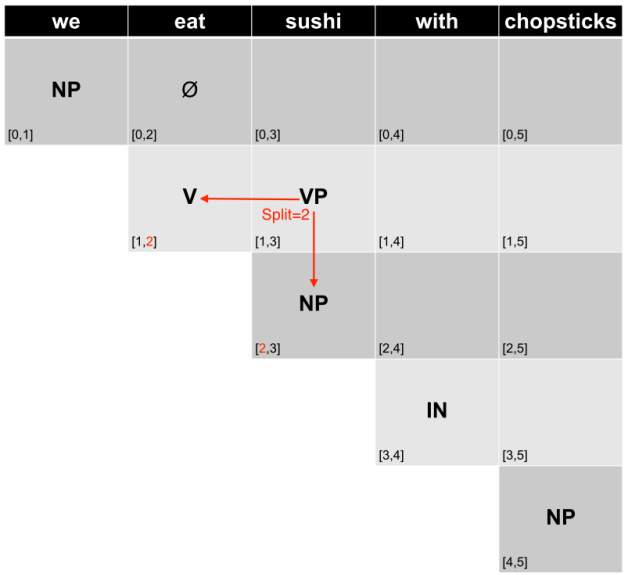


Parsing (CYK algorithm)

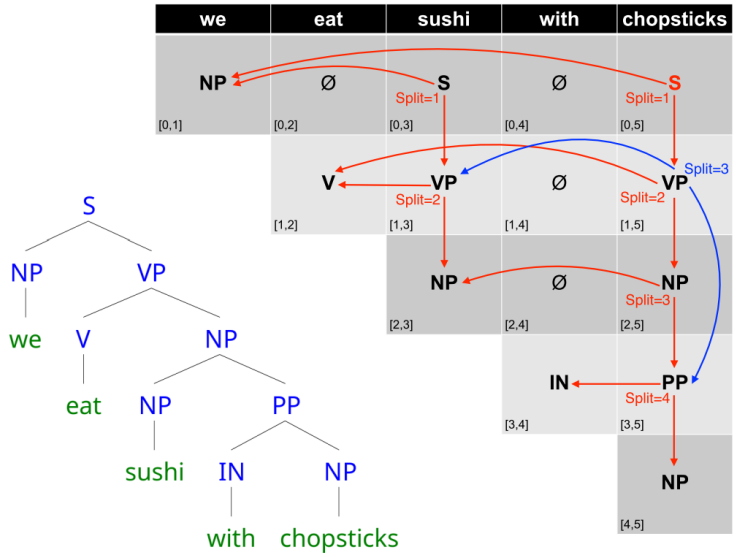
- Parsing is the reverse process from generation – deriving a parse tree from a sentence.
- The CYK algorithm is a bottom-up parsing method which tests whether a string is valid given a CFG, without enumerating all possible parses.
- It is a dynamic programming technique, meaning that it involves simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.
- The core idea is to form small constituents first and merge them into larger constituents.
- Requirement: CFGs must be in Chomsky Normal Form.
- The procedure for CYK is as follows:
    - Construct an upper triangle of an n x n matrix, where n is the sentence length.
    - Write each word along the top of each column.
    - Use terminal rules to determine the POS labels for the bottom diagonal cells.
    - Fill out all the cells in the table by column working left to right, and within each column working bottom to top.
    - To fill each cell, we need to find a RHS rule that generates the outputs of the cell to the LEFT and BELOW the cell in question, in that order.
    - For each cell we need to check each possible combination of left and below cells in which at least one index is matched. Combinations are shown in the diagram below.
    - Arrows are drawn from each filled cell to the left and right cells containing the symbols that it relates to.
    - A successful parse is returned when we reach 'S' in a cell. The parse is retrieved by following all pairs of arrows backwards through the table.
- The algorithm may return multiple parses for a given sentence. Not all are necessarily useful.

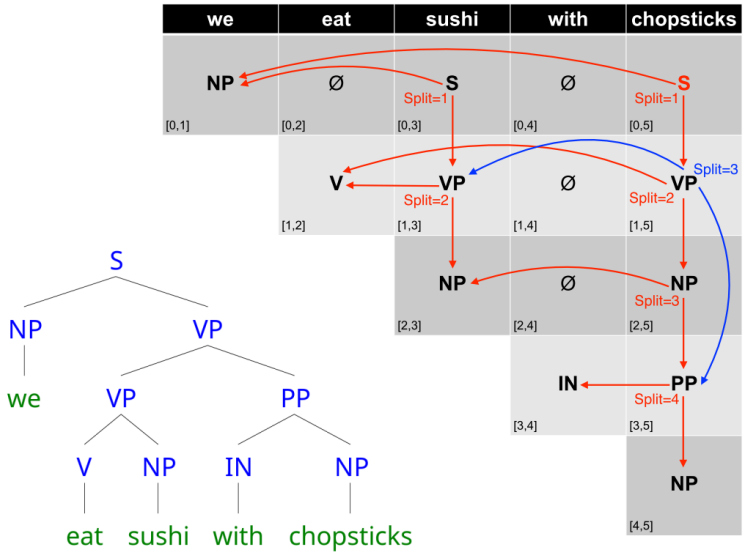






- In this example the first parse doesn't make sense, since you are eating 'sushi' and just using the chopsticks, you aren't eating 'sushi with chopsticks'.





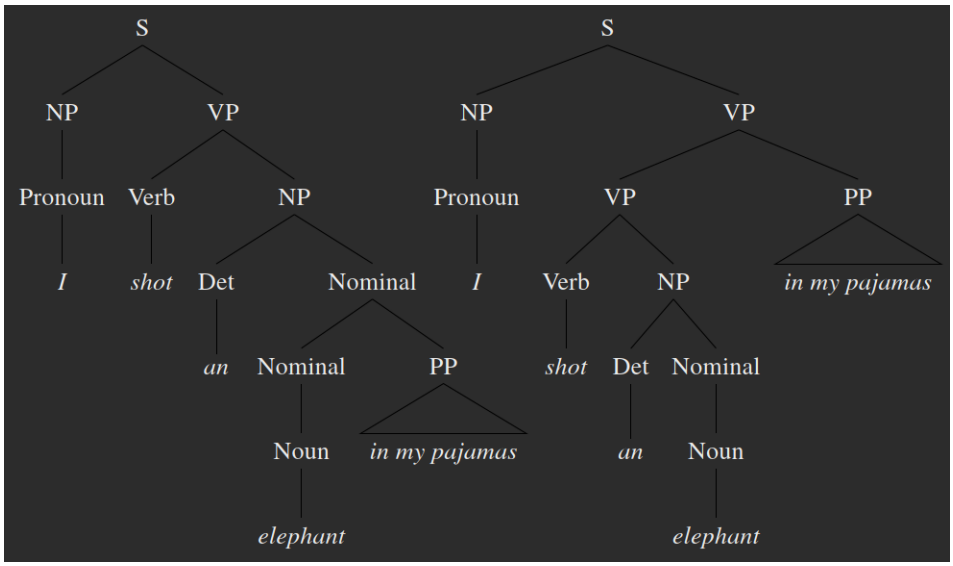
English CFGs

- Real English grammars are very complex with thousands of production rules.
- Some key constituents in the Penn Treebank:
  - Sentence (S)
  - Noun phrase (NP), e.g. NP → DT NN
  - Verb phrase (VP), e.g. VP → VB IN NP
  - Prepositional phrase (PP), e.g. PP → IN NP
  - Adjective phrase (AdjP), e.g. AdjP → AdvP JJ
  - Adverbial phrase (AdvP), e.g. AdvP → AdvP RB
  - Coordination clause (NP → NP CC NP; VP → VP CC VP)
- Some of the major English sentence structures are summarized below.

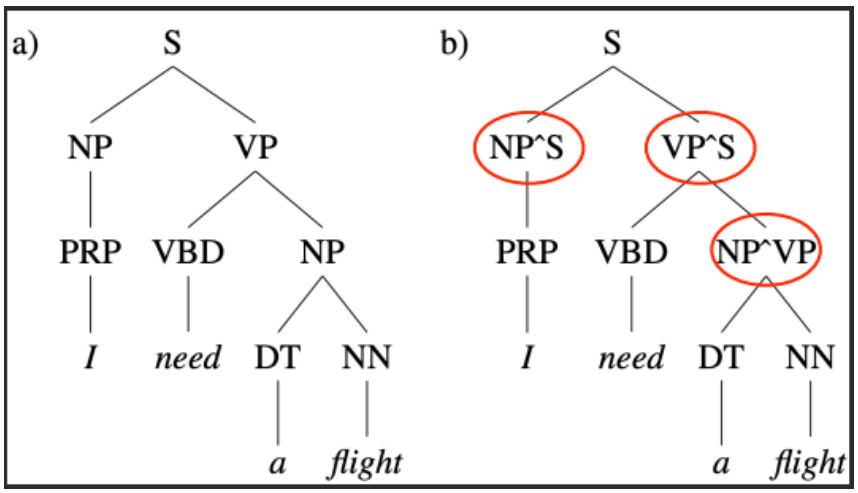
- Declarative sentences (S → NP VP)
  - *The rat ate the cheese*
- Imperative sentences (S → VP)
  - *Eat the cheese!*
- Yes/no questions (S → VB NP VP)
  - *Did the rat eat the cheese?*
- *Wh*-subject-questions (S → WH VP)
  - *Who ate the cheese?*
- *Wh*-object-questions (S → WH VB NP VP)
  - *What did the rat eat?*

Limitations of CFGs

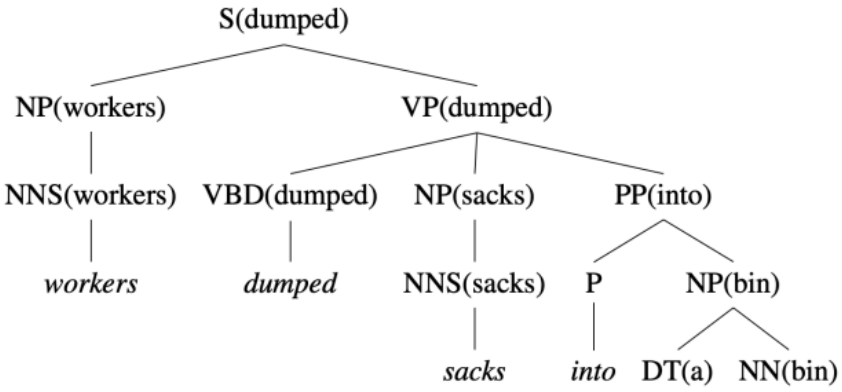
- Parse ambiguity: a major problem with CFGs. An example of this is shown below, where two parses are possible for a single sentence. Because of parse ambiguity, there is a need to determine which is better. Probabilistic CFGs can be used for this purpose.



- No parent conditioning: even with probabilistic CFGs, the strong independence assumption means that there is no way to represent contextual dependencies in production rule probabilities. One solution is parent conditioning, where we non-terminals more explicit by incorporating a parent symbol into each symbol, at the cost of additional complexity.



- No lexical conditioning: lack of dependence on the semantic content of the words in the three often leads to implausible parses. This can be partly resolved by introducing a head word along with each parent symbol, which specifies the context the symbol occurs in. This allows for lexical context at the cost of massively expanding the symbol inventory.

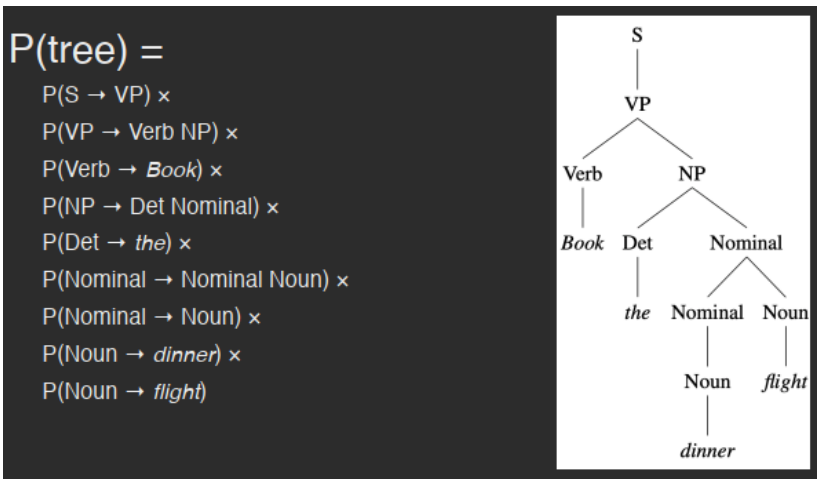


Probabilistic CFGs

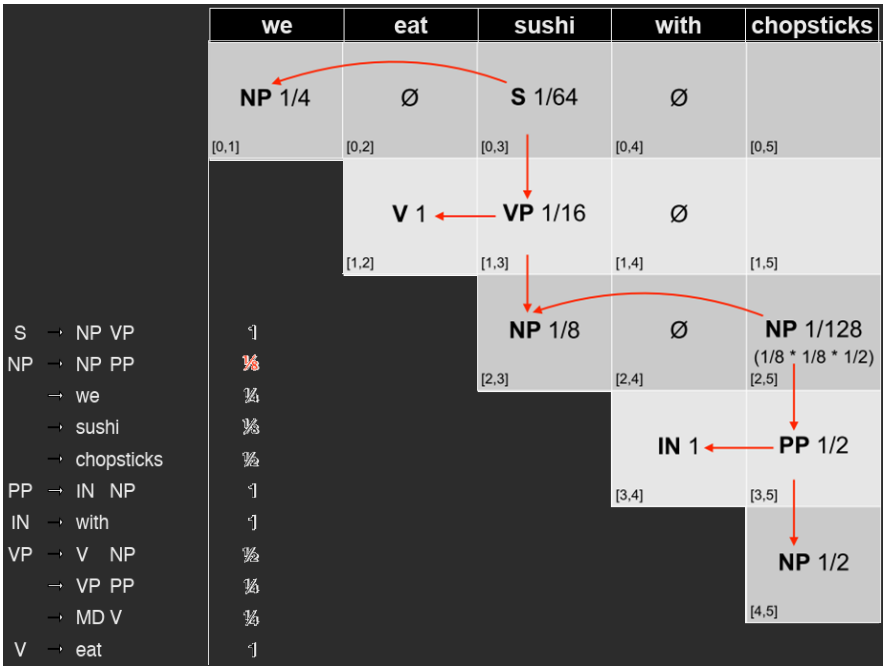
- A probabilistic CFG is exactly the same as a regular one, except that each production rule is also associated with a probability of the corresponding production.
- Each probability is the conditional probability of generating the RHS given the LHS. A simple example is shown below.

NN → aadvark	[p = 0.0003]
NN → cat	[p = 0.02]
NN → leprechaun	[p = 0.0001]
$\sum_x$ P(NN → *x*) = 1	

- We can use these conditional probabilities to compute the probability of any given parse tree. It simply decomposes into the product of each production rule.



- To apply the CYK algorithm in the probabilistic case, we follow the same process as the non-probabilistic case, and also write down the cumulative probability of applying all the production rules required to each cell.



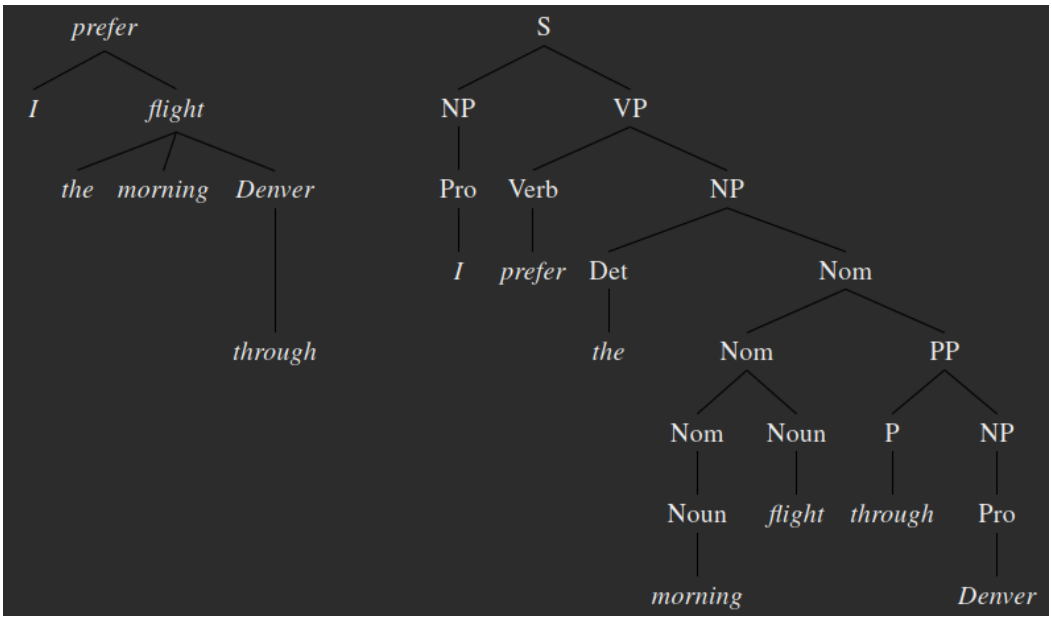
Dependency grammar

Dependency relations

- In dependency formalisms, phrasal constituents and phrase-structure rules do not play a direct role. Instead, the syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words.
- Dependency relations capture the grammatical relation between a head (or central) word, and a dependent (or supporting) word.
- The grammatical relation can be of a subject and its direct object, for example.
- Universal Dependency: a framework to create a set of dependency relations that are computationally useful and cross-lingual.

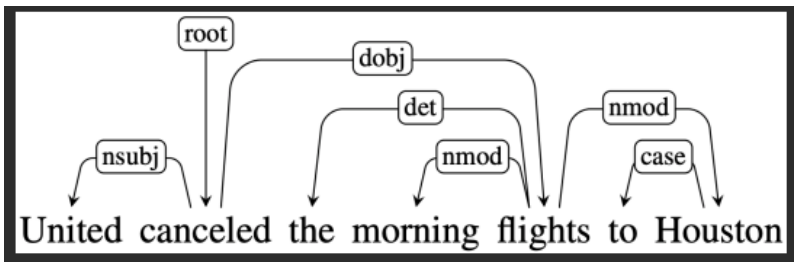
Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

- The comparison below shows the difference between a dependency and constituency parse for the same sentence.

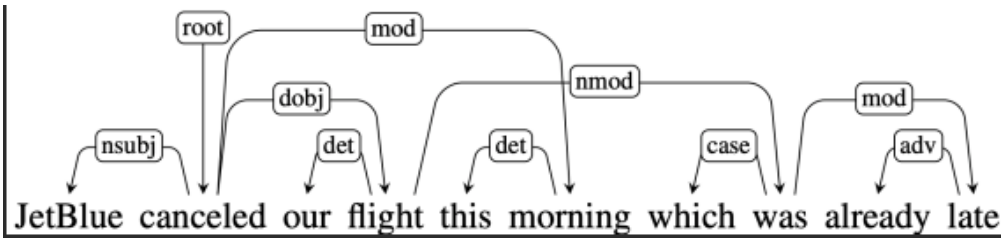


Dependency trees

- Properties of a Dependency Tree
  - Each word has a single head (parent)
  - There is a single root node
  - There is a unique path to each word from the root
  - All arcs should be projective
- Projectivity: An arc is projective if there is a path from the head to every word that lies between the head and the dependent. A dependency tree is projective if all arcs are projective; in other words if it can be drawn with no crossing edges.
- An example dependency parse for a simple sentence is shown below:



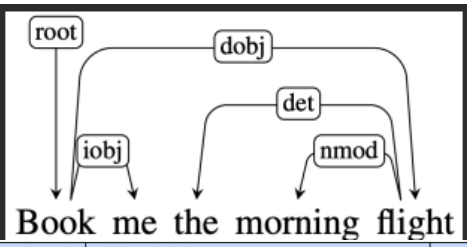
- The example parse below is not projective:



- Head-dependent relations provide an approximation to the semantic relationship between predicates and their arguments. This makes the approach useful for many applications such as coreference resolution, question answering and information extraction.

Transition-based parsing

- Transition-based parsers are a bottom-up method and can only handle projective dependency trees.
- They are hence less applicable for languages where cross-dependencies are common.
- The method works by processing the word from left to right, while maintaining two data structures:
  - Buffer: input words yet to be processed
  - Stack: store words that are being processed
- At each step, perform one of the three actions:
  - Shift: move a word from buffer to stack.
  - Left-Arc: assign current word as the head of the previous word in stack.
  - Right-Arc: assign previous word as the head of current word in stack.



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

- In practice, we parameterise the left-arc and right-arc actions with dependency labels, yielding more than three types of actions.
- We assume an oracle that tells us the right action at every step in the parse. Obviously we don't have an oracle, but we can build a classifier that will provide us with a prediction.
- The idea is to add all dependents of a particular component to the stack, and then process them in reverse order back to the root. Then we repeat with a different dependency branch.
- We can construct the classifier from any standard method. It is trained to take stack and buffer configurations and predict which of the three output classes the action belongs in (shift, rightarc or leftarc).
- Beam search can be used to keep track of the best n actions at any point, to minimize the risk of error propagation.

Graph-based parsing

- Graph-based dependency parsers search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score.
- These methods encode the search space as directed graphs and employ methods drawn from graph theory to search the space for optimal solutions.
- They are more computationally demanding than transition-based methods, as they are not greedy algorithms.
- Major advantage is improved accuracy, and ability to parse non-projective dependency trees.

4. Applications of Natural Language Processing

Machine translation

Challenges of machine translation

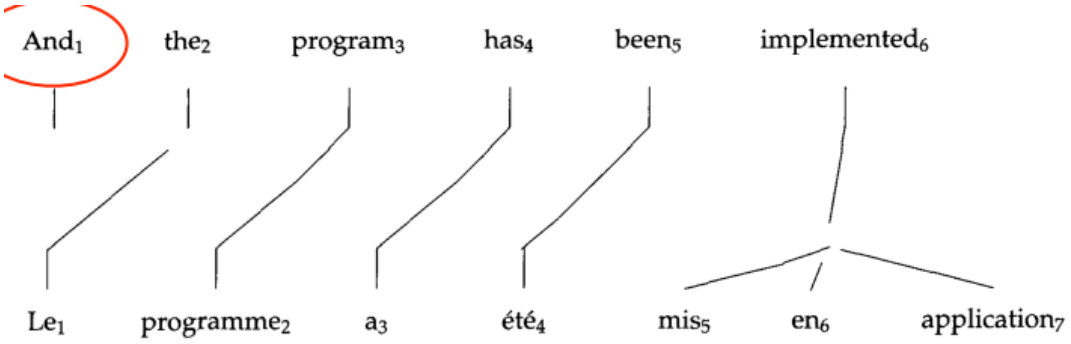
- Machine translation (MT) is the task of translating text from one source language to another target language.
- Difficult because one cannot simply translate word-for-word; need the whole idea.
- Different languages have structural differences in syntax and word order.
- Some languages have complex inflections and word conjugations due to case, gender, tense.

Statistical machine translation

- Earlier methods for MT used statistical methods based on Bayes' Theorem.
- Find the target sentence t that is most probable given the source sentence s:

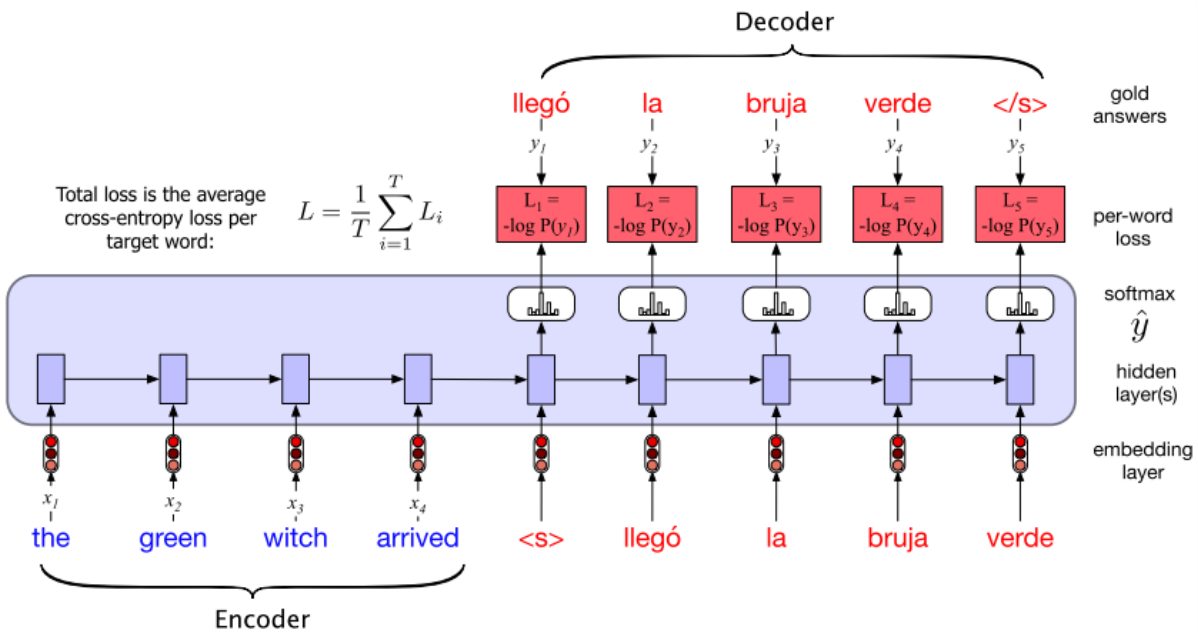
$$P(t|s) = P(s|t)P(t)$$

- This combines a regular language model of the target language to produce $P(t)$ with a translation model $P(s|t)$.
- We can learn the language model using n-gram models discussed previously from a monolinguistic corpus. The translation model requires a bilingual corpus consisting of a parallel corpus of paired source and target sentences.
- To learn the probability $P(s|t)$ from the bilingual corpus, we need to align the words in each of the two sentences. We can introduce the alignment as a latent variable and train using expectation maximization: $P(s|t, a)$.
- Alignment is complex since some words have no alignment, and mappings can be one-to-many or man-to-one.

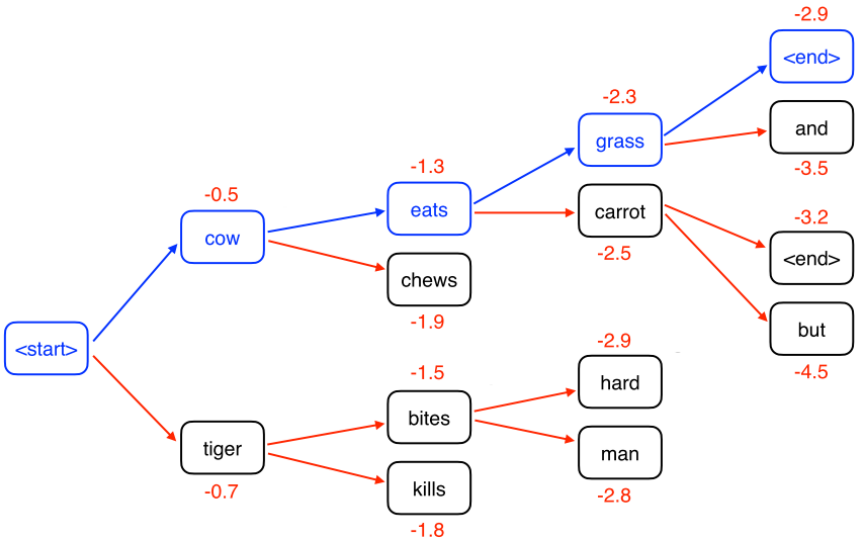


Early neural methods

- Neural network methods dispense with the language model, and instead directly train an RNN to encode the source sentence, and another to decode into the target sentence.
- Requires parallel corpus just like statistical MT. Trains with next word prediction.
- The training proceeds in a supervised fashion, encoding the source sentence and then training the decoder based on the known target sentence, one word at a time.



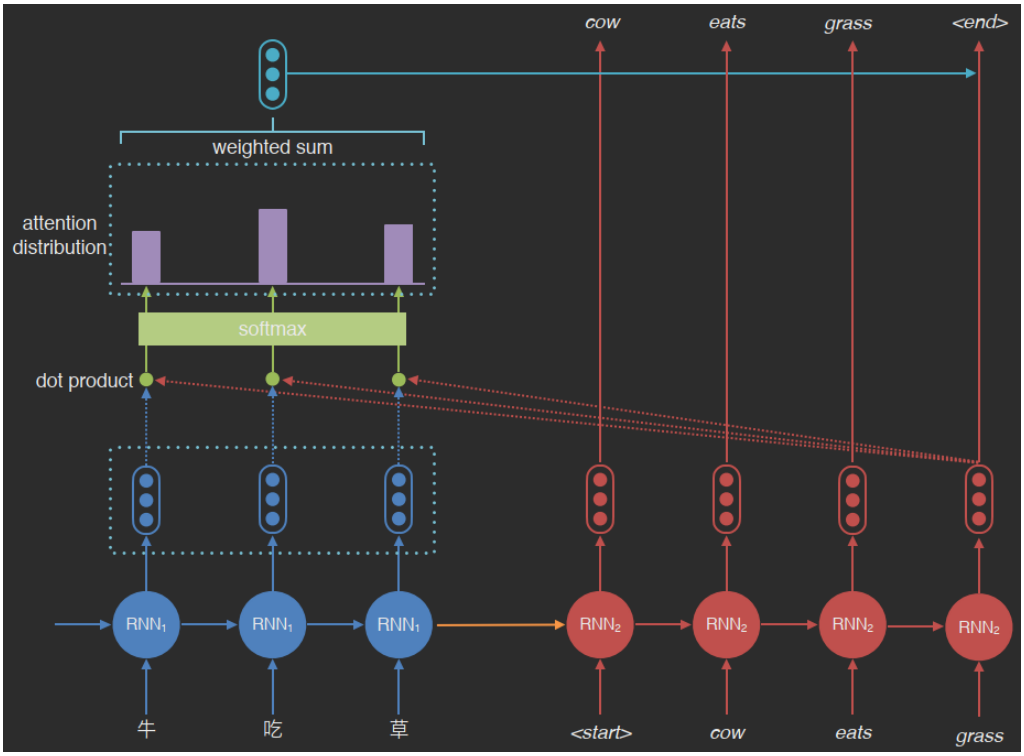
- During decoding in test mode, we obviously don't have the target sentence, so we take the maximum probability word prediction at each step.
- This greedy decoding approach leads to exposure bias, where if the model makes an error early on, it will not be able to recover to yield the optimal global solution.
- An alternative approach is to keep track of the k best words at each step, where k is known as the beam width. This is known as beam search. The diagram below shows k=2.
- Decoding terminates when the model generates a <stop> token. In beam search, we store hypotheses that have terminated, and continue explore those that haven't.



Attention-based methods

- With a long source sentence, the encoded vector is unlikely to capture all the information in the sentence. This creates an information bottleneck.
- To resolve this issue, we can allow the encoder to 'attend' to words in the source sentence.
- This is implemented by learning attention weights from the target back to the source representation. These are dotted with the source sentence to yield an attention-weighted representation of the source sentence.

- This attention-weighted source sentence is than concatenated with the decoder hidden states to predict the next word.
- In addition to resolving the information bottleneck, attention also provides some form of interpretability, as attention weights can be seen as word alignments.



Evaluation

- A common metric for evaluation is called BLEU, which computes the n-gram overlap between "reference" translation and generated translation.
- This is typically computed for 1-grams to 4-grams.

$$BLEU = BP \times \exp\left(\frac{1}{N}\sum_{n=1} \log\left(\frac{corr_n}{pred_n}\right)\right)$$

- Where $corr_n$ is the number of correct n-grams and $pred_n$ the number of predicted n-grams.
- BP is a brevity penalty, which penalizes very short outputs relative to L_{ref}.

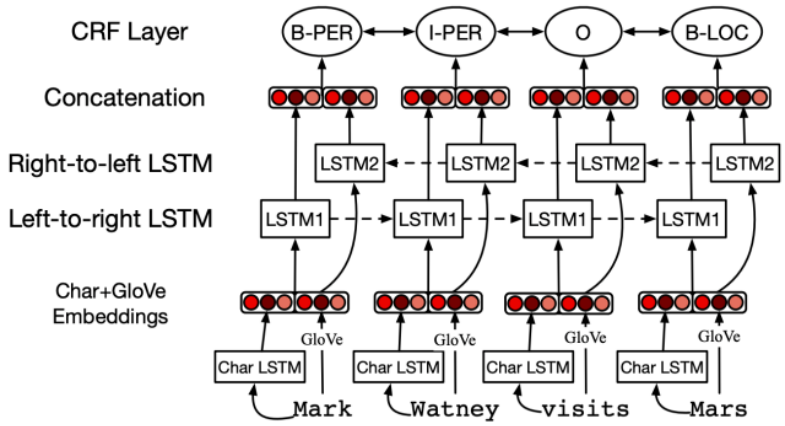
$$BP = \min\left(1, \frac{\text{len}(output)}{L_{ref}}\right)$$

Information extraction

Named entity recognition

- Seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, etc.
- Requires a set of entity tags, which will be determined based on the desired application.
- Since tags can span multiple words, we need tags for the start and continuation of a particular category.
- NER is a sequence labelling task, so we can train hidden Markov models, conditional random fields, RNNs, or LSTMs.

Words	IOB Label	IO Label
American	B-ORG	I-ORG
Airlines	I-ORG	I-ORG
,	O	O
a	O	O
unit	O	O
of	O	O
AMR	B-ORG	I-ORG
Corp.	I-ORG	I-ORG
,	O	O
immediately	O	O
matched	O	O
the	O	O
move	O	O
,	O	O
spokesman	O	O
Tim	B-PER	I-PER
Wagner	I-PER	I-PER
said	O	O
.	O	O



Relation extraction

- Relation extraction is the task of finding and classifying semantic relations among entities mentioned in a text, like child-of (X is the child-of Y), or part-whole or geospatial relations.
- To do this we need to have a set of relation tags to use, which are based on application.
- Most relations are based on a triple: relation(subject, object).
- The relata for the relation may be the named entities recognized as discussed above.

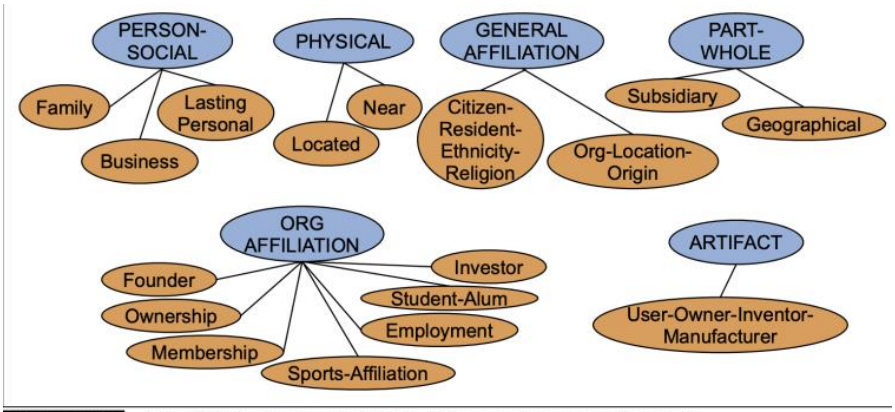


Figure 18.9 The 17 relations used in the ACE relation extraction task.

Methods for relation extraction

- Rule-based: uses hand-written patterns of words called Hearst patterns, which commonly encode relational information. Some examples are shown below.

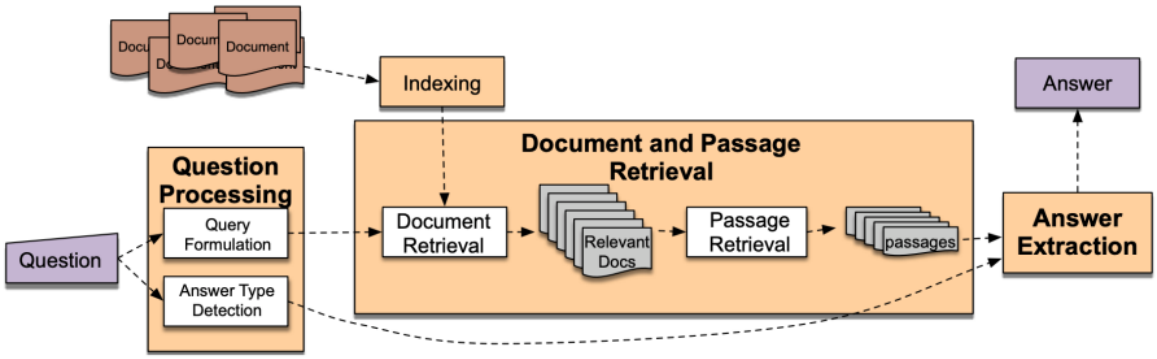
NP {, NP}* {,} (and|or) other NP_H	temples, treasuries, and other important civic buildings
NP_H such as {NP,}* {(or|and)} NP	red algae such as Gelidium
such NP_H as {NP,}* {(or|and)} NP	such authors as Herrick, Goldsmith, and Shakespeare
NP_H {,} including {NP,}* {(or|and)} NP	common-law countries, including Canada and England
NP_H {,} especially {NP}* {(or|and)} NP	European countries, especially France, England, and Spain

- Supervised: obtain a set of hand-labelled data and then train a classifier on this data. Labelled data is very expensive and often does not generalize well.
- Semi-supervised: uses a small set of seed tuples to find more, similar tuples by matching patterns in a dataset. These new tuples are then used as seeds. Allows for a large dataset to be collected without having to manually label all examples. Main problem is semantic drift.
- Distant supervision: instead of just a handful of seeds, distant supervision uses a large database to acquire a huge number of seed examples, creates lots of pattern features from these examples and combines them in a supervised classifier. Effective but results are noisy.
- Unsupervised: does not use any labelled training data or set of pre-defined relations. Instead learns relations by POS tagging, identifying the verbs, and then identifying the arguments of the verbs. Then cluster relations into similar classes.
- Typically, relation extraction with known relation set is evaluated using F1-score.

Question answering

Information retrieval methods

- The approach of IR-based systems is to find one or more documents relative to the query, then extract a short string from the document which answers the query.

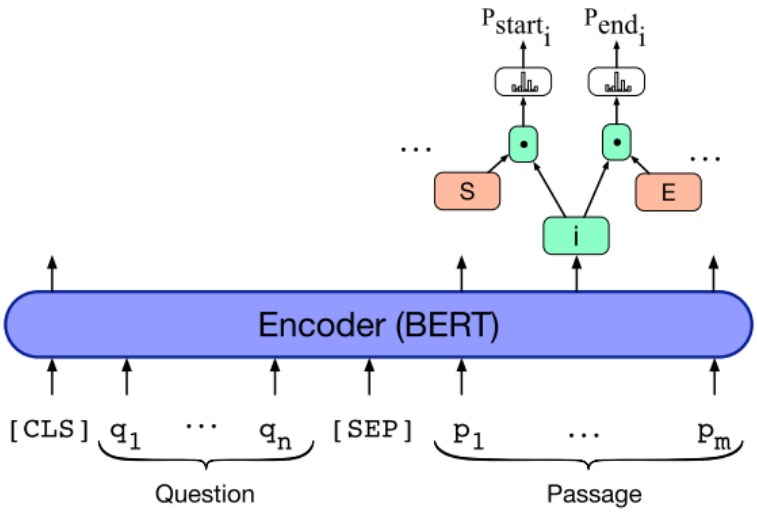


- A document d can be scored for its relevance to query q by examining all terms t in the query, using a simplification of the dot product between document and query embeddings:

$$\text{score}(q,d) = \sum_{t \in q} \frac{\text{tf.idf}(t,d)}{|d|}$$

- Once we have found a candidate document, we repeat the above analysis using each passage or segment of the document to find a candidate answer.
- Knowing the type of answer that we are looking for can be helpful in identifying the passage in the matching document. This can be determined using a simple classifier.

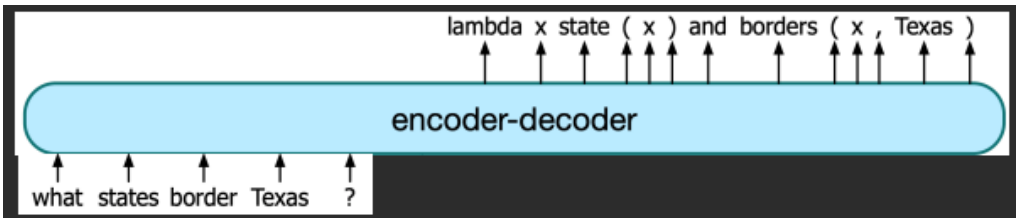
- More recent IR methods use semantic embedding models such as BERT to match query to document, and then predict the span of the answer (called reading comprehension).



- These supervised models are trained on datasets like the Stanford Question Answering Dataset (SQuAD), which consists of passages from Wikipedia and associated questions whose answers are spans from the passage, as selected by humans.

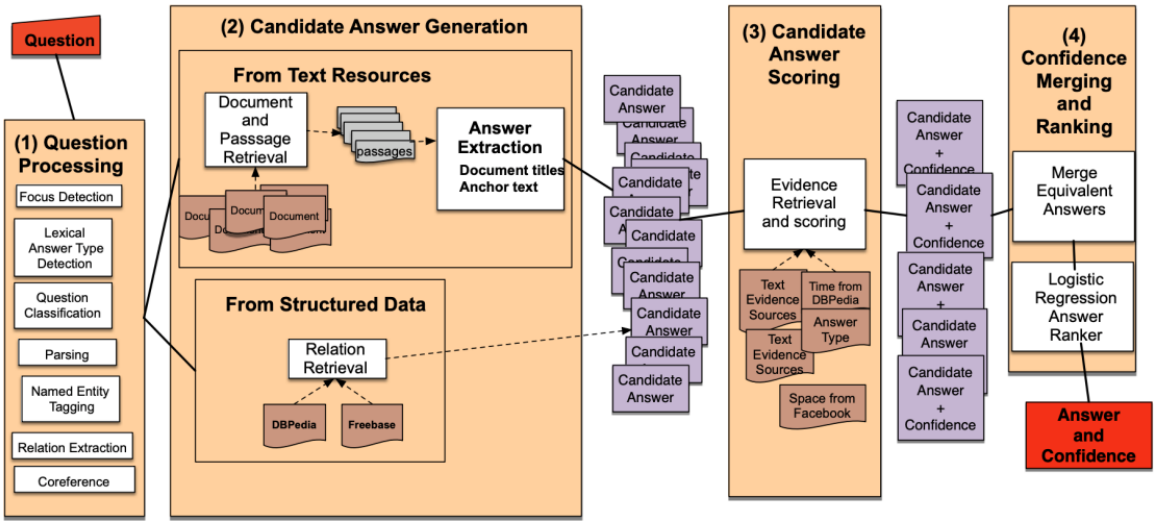
Knowledge-based methods

- The approach of knowledge-based question answering systems is to map a question to a specific query over a structured database.
- The first step is semantic parsing, the process of converting the question into logical form suitable for the relevant database. This can be done using a model like BERT.
- The second step is the run the query in the structured database, returning the result.



Hybrid methods

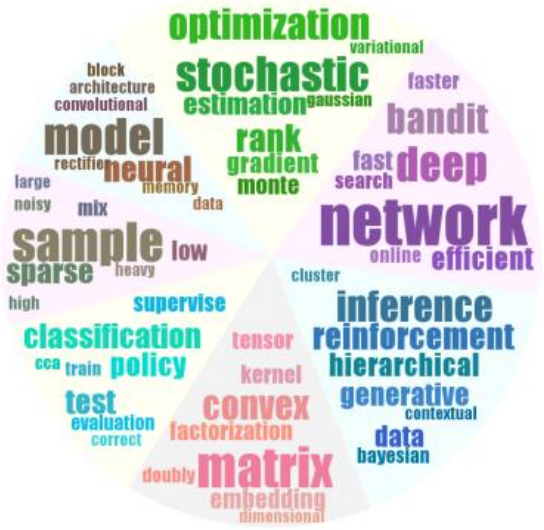
- Hybrid models use a combination of IR and KB methods.
- IBM's Watson won Jeopardy!, using a variety of resources to answer questions.
- First, the question focus is extracted. The focus is the string of words in the question that corefers with the answer. For Jeopardy! It is possible to use hand-written rules to extract the focus, as the questions are written in a consistent style.
- Next, the lexical answer type is extracted. This is a word or words which tell us something about the semantic type of the answer. These are again extracted by rules.
- Finally, the type of question is extracted, again using rules (definition question, multiple-choice, puzzle, fill-in-the-blank).
- Using all this extracted information, a large collection of candidate answers is constructed from structured databases.
- Candidate answers are then scored based on a wide range of features extracted from the question. Equivalent answers are merged using entity linking techniques.



Topic modelling

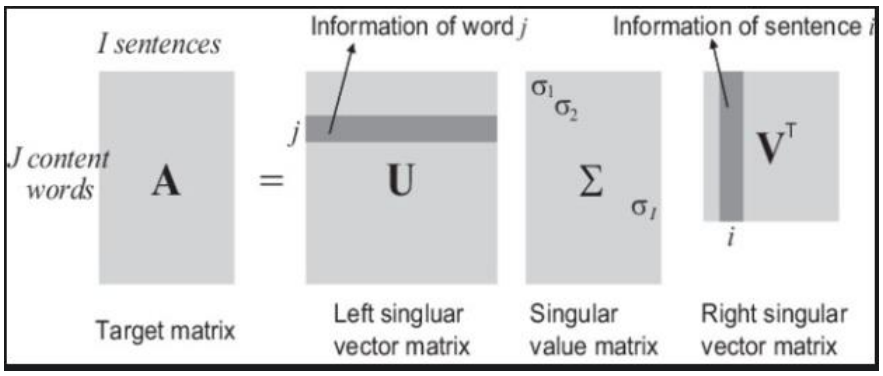
Intro to topic models

- Topic models learn common, overlapping themes in a document collection.
- Each document is assigned a list of topics, each of which is in turn a weighted list of words.
- Words of a topic typically appear in the same set of documents in the corpus.
- A visualization of a topic model for machine learning is given in the diagram below.



Latent semantic analysis

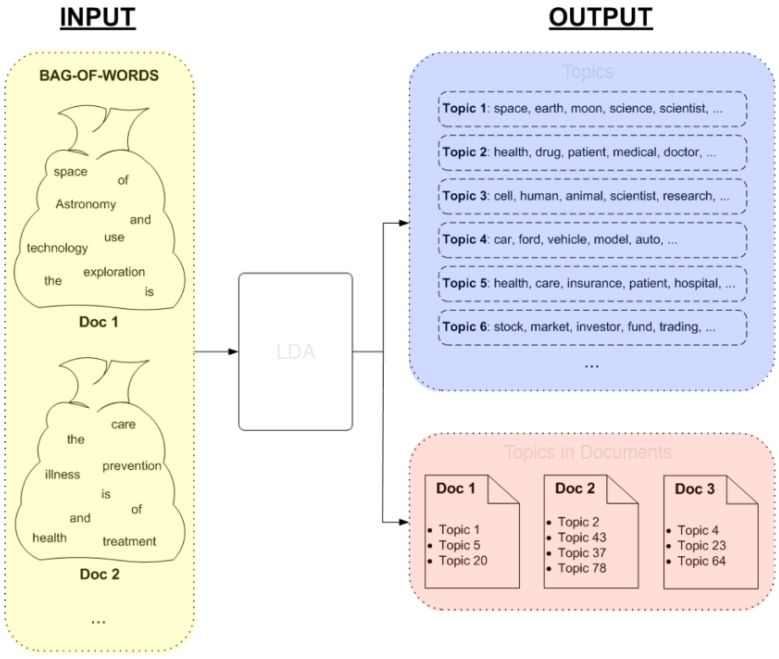
- A set of documents can be represented as a document by word matrix, by simply counting how often each word occurs in each document.
- Using a linear algebra technique called singular value decomposition, we can rewrite this doc by word matrix into the product of three matrices: word by topic (U), a diagonal matrix of eigenvalues (Σ), and topic by document (V).
- While each row of U can be used as a word-embedding for the corresponding word, each column of V can be used as a topic representation for the corresponding document.



- Probabilistic LSA is a variation of regular LSA which makes all the values in V probabilities, making them much easier to interpret as weights for each topic.
- The main limitation of PLSA is that it cannot be used to infer topics for new documents, but must be retrained every time we update our set of documents.

Latent Dirichlet allocation

- Latent Dirichlet allocation is effectively a Bayesian variant of PLSA, which treats the topics as a latent variable. It allows for inferring topics on new (untrained) documents.



- The latent 'topic' variable can be learned using Gibbs sampling techniques:
  - Randomly assign topics to all tokens in documents.
  - Collect topic-word and document-topic co-occurrence statistics based on the assignments.
  - For each word token in each document in the corpus, randomly select a topic using the current topic-word table.
  - If this topic is the same as the current assigned topic for that word token, do nothing. If it is different, reassign the topic and update the topic-word and document-topic co-occurrence statistics appropriately.
  - Train until convergence, which occurs when the model probability of the training set becomes stable. For number of word tokens N and topics T, this is computed as:

$$\log P(w_1, w_2, \dots, w_n) = \sum_{i=1}^{N} \left[\log \sum_{j=0}^{T} P(w_i|t_j) P(t_j|d_{w_i}) \right]$$

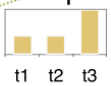
1. Randomly assign topics to all tokens in documents

doc1	mouse: ?	cat: t3	rat: t2	chase: t1	mouse: t3
doc2	scroll: t1	mouse: t3	scroll: t3	scroll: t2	click: t2
doc3	tonight: t2	baseball: t1	tv: t2	exciting: t1	

2. Collect **topic-word and document-topic co-occurrence statistics** based on the assignments

	mouse	cat	scroll	tv	...
t1	1.01-1	0.01	1.01	0.01	
t2	0.01	0.01	1.01	1.01	
t3	2.01	1.01	1.01	0.01	

	t1	t2	t3
d1	2.1-1	1.1	2.1
d2	1.1	2.1	2.1
...			

3. Go through every word token in corpus and sample a new topic:
   - $P(t_i|w,d) \propto P(t_i|w)P(t_i|d)$ 

   Need to de-allocate the current topic assignment and update the co-occurrence matrices before sampling
4. Go to step 2 and repeat until convergence

$$P(t_1|w,d) = P(t_1|\text{mouse}) \times P(t_1|d_1)$$
$$\frac{0.01}{0.01+0.01+2.01} \times \frac{1.1}{1.1+1.1+2.1}$$

- Several hyper-parameters must be chosen:
  - T: the number of topis.
  - β: prior on the topic-word distribution (generally choose low value to ensure each topic has only a few words).
  - α: prior on the document-topic distribution (generally choose larger value to ensure each document has many topics).

Evaluation

- Because topic modelling is an unsupervised method, there are no labels to compare to.
- We can use perplexity, which is the exponential of the negative average log likelihood of the data under the model.
- However, perplexity is not comparable for different corpora or different tokenisation/preprocessing methods. It also does not correlate with human perception of topic quality.
- A better method is to use topic coherence, which involves injecting a random word into each topic and asking humans to guess the intruder word.
- This this is very expensive, an automated alternative is to calculate the sum of the pairwise PMI over the top N words in the topic t:

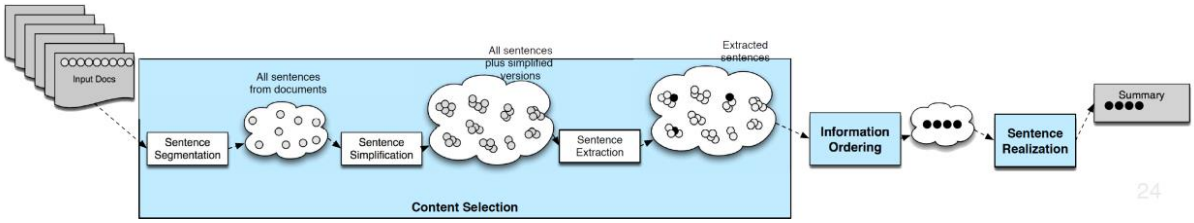
$$PMI(t) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}$$

- If all word pairs in a topic has high PMI, we infer that the topic is coherent.
- We can calculate the probabilities of word cooccurrences using a different corpus.

Text summarisation

Extractive single-doc

- Extractive summarisation summarises by selecting representative sentences from the document or documents.
- Three main steps to extractive summarisation:
    - Content selection: select what sentences to extract from the document
    - Information ordering: decide how to order extracted sentences.
    - Sentence realisation: cleanup to make sure combined sentences are fluent.
- For single document tasks, the most important goal is to select salient sentences from the document. One then just presents them in the original order.
- Methods for content selection:
    - Tf-idf: Weigh each word in document by its inverse document frequency.
    - Log likelihood ratio: a word is salient if its probability in the input corpus is very different to a background corpus.
    - Sentence centrality: measure distance between sentences, and choose sentences that are closer to other sentences. Use tf-idf BOW to represent sentence, and cosine similarity to measure distance.
    - RST parsing: rhetorical structure theory, used to explain how clauses are connected.
    - Define the types of relations between a nucleus (main clause) and a satellite (supporting clause).



Extractive multi-doc

- Must have a mechanism for removing redundant information that has been found in multiple documents.
- Maximum Marginal Relevance: a technique for removing redundancy, in which the best sentence is iteratively added to the summary, but with a penalty for any candidates that are similar to existing sentences.
- Sentences can be ordered chronologically, or such that adjacent sentences are more similar to each other. This can be determined using sentence centrality.

Abstractive summarisation

- Abstractive summarisation creates a paraphrase of the document, rather than simply extracting sentences from it.
- This can be treated as a sentence encoding and decoding task, where first the document is encoded, and then the summary is decoded. This requires labelled examples for training.
- One dataset that can be used for this are news article headlines and the first sentence or paragraph of their corresponding article.
- One problem with such methods is that they have the potential to ‘hallucinate’ material not found in the original article.

- Evaluation is doing mainly using ROUGE (Recall Oriented Understudy for Gisting Evaluation). This is similar to BLEU, in that it evaluates the degree of word overlap between generated summary and reference/human summary using n-grams.