Natural Language Processing

Last update: 10 December 2024

Academic Year 2024 – 2025

Alma Mater Studiorum · University of Bologna

Contents

1 Basic text processing

Text normalization Operations such as:

- **Tokenization** Split a sentence in tokens. — Tokenization

  Remark. Depending on the approach, a token is not always a word.

- **Lemmatization/stemming** Convert words to their canonical form. — Lemmatization/stemming

  Example. {`sang`, `sung`, `sings`} $\mapsto$ `sing`

- **Sentence segmentation** Split a text in sentences. — Sentence segmentation

  Remark. A period does not always signal the end of a sentence.

1.1 Regular expressions

Regular expression (regex) Formal language to describe string patterns. — Regular expression (regex)

1.1.1 Basic operators

Disjunction (brackets) Match a single character between square brackets `[]`.

Example. `/[wW]oodchuck/` matches `Woodchuck` and `woodchuck`.

Range Match a single character from a range of characters or digits.

Example.

- `/[A-Z]/` matches a single upper case letter.
- `/[a-z]/` matches a single lower case letter.
- `/[0-9]/` matches a single digit.

Negation Match the negation of a pattern.

Example. `/[^A-Z]/` matches a single character that is not an upper case letter.

Disjunction (pipe) Disjunction of regular expressions separated by `|`.

Example. `/groundhog|woodchuck/` matches `groundhog` and `woodchuck`.

Wildcards

- **Optional** A character followed by `?` can be matched optionally.

  Example. `/woodchucks?/` matches `woodchuck` and `woodchucks`.

- **Any** `.` matches any character.
- **Kleene *** A character followed by `*` can be matched zero or more times.
- **Kleene +** A character followed by `+` must be matched at least once.
- **Counting** A character followed by `{n,m}` must be matched from n to m times.

Example.

- `{n}` matches exactly n instances of the previous character.
- `{n,m}` matches from n to m instances of the previous character.
- `{n,}` matches at least n instances of the previous character.
- `{,m}` matches at most m instances of the previous character.

Anchors

Start of line `^` matches only at the start of line.

Example. `/^a/` matches a but not `ba`.

End of line `$` matches only at the end of line.

Example. `/a$/` matches a but not `ab`.

Word boundary `\b` matches a word boundary character.

Word non-boundary `\B` matches a word non-boundary character.

Aliases

- `\d` matches a single digit (same as `[0-9]`).
- `\D` matches a single non-digit (same as `[^\d]`).
- `\w` matches a single alphanumeric or underscore character (same as `[a-zA-Z0-9_]`).
- `\w` matches a single non-alphanumeric and non-underscore character (same as `[^\w]`).
- `\s` matches a single whitespace (space or tab).
- `\S` matches a single non-whitespace.

Capture group Operator to refer to previously matched substrings.

Example. In the regex `/the (.*)er they were, the \1er they will be/`, `\1` should match the same content matched by `(.*)`.

1.2 Tokenization

Lemma Words with the same stem and roughly the same semantic meaning. Lemma

Example. `cat` and `cats` are the same lemma.

Wordform Orthographic appearance of a word. Wordform

Example. `cat` and `cats` do not have the same wordform.

Vocabulary Collection of text elements, each indexed by an integer. Vocabulary

Remark. To reduce the size of a vocabulary, words can be reduced to lemmas.

Type / Wordtype Element of a vocabulary (i.e., wordforms in the vocabulary). Type / Wordtype

Token Instance of a type in a text. Token

Genre Topic of a text corpus (e.g., short social media comments, books, Wikipedia pages, ...). Genre

Remark (Herdan's law)**.** Given a corpus with N tokens, a vocabulary V over that corpus roughly have size:

$$|V| = kN^{\beta}$$

where the typical values are $10 \leq k \leq 100$ and $0.4 \leq \beta \leq 0.6$.

Stopwords Frequent words that can be dropped. Stopwords

Remark. If semantics is important, stopwords should be kept. LLMs keep stopwords.

Rule-based tokenization Hand-defined rules for tokenization. Rule-based tokenization

Remark. For speed, simple tokenizers use regex.

Data-driven tokenization Determine frequent tokens from a large text corpus. Data-driven tokenization

1.2.1 Data-driven tokenization

Tokenization is done by two components:

Token learner Learns a vocabulary from a given corpus (i.e., training). Token learner

Token segmenter Segments a given input into tokens based on a vocabulary (i.e., inference). Token segmenter

Byte-pair encoding (BPE) Based on the most frequent n-grams. Byte-pair encoding (BPE)

Token learner Given a training corpus C, BPE determines the vocabulary as follows:

1. Start with a vocabulary V containing all the 1-grams of C and an empty set of merge rules M.
2. While the desired size of the vocabulary has not been reached:
   a) Determine the pair of tokens $t_1 \in V$ and $t_2 \in V$ such that, among all the possible pairs, the n-gram $t_1 + t_2 = t_1t_2$ obtained by merging them is the most frequent in the corpus C.
   b) Add t_1t_2 to V and the merge rule $t_1 + t_2$ to M.

Example. Given the following corpus:

Occurrences	Tokens
5	`l o w $`
2	`l o w e r $`
6	`n e w e s t $`
6	`w i d e s t $`

The initial vocabulary is: $V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\}$.
At the first iteration, $\texttt{e} + \texttt{s} = \texttt{es}$ is the most frequent n-gram. Corpus and vocabulary are updated as:

Occurrences	Tokens
5	`l o w $`
2	`l o w e r $`
6	`n e w es t $`
6	`w i d es t $`

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\} \cup \{\texttt{es}\}$$

At the second iteration, $\texttt{es} + \texttt{t} = \texttt{est}$ is the most frequent n-gram:

Occurrences	**Tokens**
5	`l o w $`
2	`l o w e r $`
6	`n e w est $`
6	`w i d est $`

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}, \texttt{es}\} \cup \{\texttt{est}\}$$

And so on...

Token segmenter Given the vocabulary V and the merge rules M, the BPE segmenter does the following:

1. Split the input into 1-grams.
2. Iteratively scan the input and do the following:
   a) Apply a merge rule if possible.
   b) If no merge rules can be applied, lookup the (sub)word in the vocabulary. Tokens out-of-vocabulary are marked with a special unknown token `[UNK]`.

WordPiece Similar to BPE with the addition of merge rules ranking and a special leading/tailing set of characters (usually `##`) to identify subwords (e.g., `new##`, `##est` are possible tokens). WordPiece

Unigram Starts with a big vocabulary and remove tokens following a loss function. Unigram

1.3 Normalization

Normalization Convert tokens into a standard form. Normalization

Example. `U.S.A.` and `USA` should be encoded using the same index.

Case folding Map every token to upper/lower case. Case folding

Remark. Depending on the task, casing might be important (e.g., `US` vs `us`).

Lemmatization Reduce inflections and variant forms to their base form. Lemmatization

Example. $\{\texttt{am}, \texttt{are}, \texttt{is}\} \mapsto \texttt{be}$

Remark. Accurate lemmatization requires complete morphological parsing.

Stemming Reduce terms to their stem. Stemming

Remark. Stemming is a simpler approach to lemmatization.

Porter stemmer Simple stemmer based on cascading rewrite rules.

Example. $\texttt{ational} \mapsto \texttt{ate}$, $\texttt{ing} \mapsto \varepsilon$, $\texttt{sses} \mapsto \texttt{ss}$.

1.4 Edit distance

Minimum edit distance Minimum number of edit operations (insertions, deletions, and substitutions) needed to transform a string into another one. Minimum edit distance

Remark. Dynamic programming can be used to efficiently determine the minimum edit distance.

Levenshtein distance Edit distance where:

Levenshtein distance

- Insertions cost 1;
- Deletions cost 1;
- Substitutions cost 2.

Example. The Levenshtein distance between `intention` and `execution` is 8.

I	N	T	E	*	N	T	I	O	N
|	|	|	|	|	|	|	|	|	|
*	E	X	E	C	U	T	I	O	N
$-$	$\pm$	$\pm$		$+$	$\pm$				
1	2	2		1	2				

2 Language models

2.1 Spelling correction

Spelling correction Spelling errors can be of two types: Spelling correction

Non-word spelling Typos that result in non-existing words. Possible candidates can be determined though a dictionary lookup.

Real-word spelling Can be:

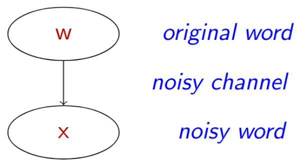
Typographical error Typos that result in existing words.

Cognitive error Due to words similarity (e.g., `piece` vs `peace`).

Noisy channel model Assumes that the observable input is a distorted form of the original word. A decoder tests word hypotheses and selects the best match. Noisy channel model

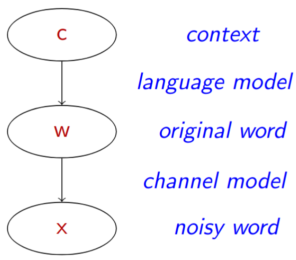
More formally, we want a model of the channel that, similarly to Bayesian inference, determines the likelihood that a word $w \in V$ is the original word for a noisy one x. From there, we can estimate the correct word $\hat{w}$:

$$\hat{w} = \arg\max_{w \in V} \mathcal{P}(w|x)$$



By applying (i) Bayes' rule, (ii) the fact that $\hat{w}$ is independent of $\mathcal{P}(x)$, and (iii) that a subset $C \subseteq V$ of the vocabulary can be used, the estimate becomes:

$$\hat{w} = \arg\max_{w \in C} \underbrace{\mathcal{P}(x|w)}_{\text{channel model}} \underbrace{\mathcal{P}(w)}_{\text{prior}}$$



Moreover, it is reasonable to include a context c when computing the prior:

$$\hat{w} = \arg\max_{w \in C} \underbrace{\mathcal{P}(x|w)}_{\text{channel model}} \underbrace{\mathcal{P}(w|c)}_{\text{language model}}$$

Noisy channel spelling method Spelling correction in a noisy channel model can be done as follows:

1. Find candidate words with similar spelling to the input based on a distance metric (e.g., Damerau-Levenshtein which is the Levenshtein distance with the addition of adjacent transpositions).
2. Score each candidate based on the language and channel model:
   - Use typing features of the user.
   - Use local context.
   - Use a confusion matrix with common mistakes.

Example. Consider the sentence:

> [...] *was called a "stellar and versatile* `acress` *whose combination of sass and glamour has defined her* [...]*"*

By using the Corpus of Contemporary English (COCA), we can determine the following words as candidates:

`actress` · `cress` · `caress` · `access` · `across` · `acres`

Language model By considering a language model without context, the priors are computed as $\mathcal{P}(w) = \frac{\texttt{count}(w)}{|\texttt{COCA}|}$ (where $|\texttt{COCA}| = 404\,253\,213$):

w	$\texttt{count}(w)$	$\mathcal{P}(w)$
`actress`	9321	0.0000231
`cress`	220	0.000000544
`caress`	686	0.00000170
`access`	37 038	0.0000916
`across`	120 844	0.000299
`acres`	12 874	0.0000318

Channel model By using a confusion matrix of common typos, the channel model is:

w	$x\|w$	$\mathcal{P}(x\|w)$
`actress`	`c|ct`	0.000117
`cress`	`a|#`	0.00000144
`caress`	`ac|ca`	0.00000164
`access`	`r|c`	0.000000209
`across`	`e|o`	0.0000093
`acres`	`es|e`	0.0000321
`acres`	`ss|s`	0.0000342

The ranking is the obtained as:

w	$\mathcal{P}(x\|w)\mathcal{P}(w)$
`actress`	$2.7 \cdot 10^9$
`cress`	$0.00078 \cdot 10^9$
`caress`	$0.0028 \cdot 10^9$
`access`	$0.019 \cdot 10^9$
`across`	$2.8 \cdot 10^9$
`acres`	$1.02 \cdot 10^9$
`acres`	$1.09 \cdot 10^9$

Therefore, the most likely correction of `acress` for this model is `across`.
If the previous word is considered in the context, the relevant tokens of the new language model are:

w_{i-1}	w_i	$\mathcal{P}(w_i\|w_{i-1})$
`versatile`	`actress`	0.000021
`versatile`	`across`	0.000021
`actress`	`whose`	0.001
`across`	`whose`	0.000006

This allows to measure the likelihood of a sentence as:

$$\mathcal{P}(\texttt{versatile } \underline{\texttt{actress}} \texttt{ whose}) = \mathcal{P}(\texttt{actress}|\texttt{versatile})\mathcal{P}(\texttt{whose}|\texttt{actress}) = 210 \cdot 10^{-10}$$
$$\mathcal{P}(\texttt{versatile } \underline{\texttt{across}} \texttt{ whose}) = \mathcal{P}(\texttt{across}|\texttt{versatile})\mathcal{P}(\texttt{whose}|\texttt{across}) = 1 \cdot 10^{-10}$$

Finally, we have that:

$$\mathcal{P}(\texttt{versatile } \underline{\texttt{actress}} \texttt{ whose}|\texttt{versatile acress whose}) = 2.7 \cdot 210 \cdot 10^{-19}$$
$$\mathcal{P}(\texttt{versatile } \underline{\texttt{across}} \texttt{ whose}|\texttt{versatile acress whose}) = 2.8 \cdot 10^{-19}$$

So `actress` is the most likely correction for `acress` in this model.

Remark. In practice, log-probabilities are used to avoid underflows and to make computation faster (i.e., sums instead of products).

2.2 Language models

(Probabilistic) language model Model to determine the probability of a word w in a given context c: Language model

$$\mathcal{P}(w|c)$$

Usually, it is based on counting statistics and uses as context the sequence of previous tokens:

$$\mathcal{P}(w_i|w_1, \dots, w_{i-1})$$

This is equivalent to computing the probability of the whole sentence, which expanded using the chain rule becomes:

$$\begin{aligned}\mathcal{P}(w_1, \dots, w_{i-1}w_i) &= \mathcal{P}(w_1)\mathcal{P}(w_2|w_1)\mathcal{P}(w_3|w_{1..2}) \dots \mathcal{P}(w_n|w_{1..n-1}) \\ &= \prod_{i=1}^{n} \mathcal{P}(w_i|w_{1..i-1})\end{aligned}$$

Remark. Simply counting the number of occurrences of a sentence as $\mathcal{P}(w_i|w_{1..i-1}) = w_{1..i}/w_{1..i-1}$ is not ideal as there are too many possible sentences.

Markov assumption Limit the length of the context to a window of k previous tokens: Markov assumption in language models

$$\mathcal{P}(w_i|w_{1..i-1}) \approx \mathcal{P}(w_i|w_{i-k..i-1})$$

$$\mathcal{P}(w_{1..n}) \approx \prod_{i=1}^{n} \mathcal{P}(w_i|w_{i-k..i-1})$$

Unigram model Model without context ($k = 0$):

$$\mathcal{P}(w_{1..n}) \approx \prod_{i} \mathcal{P}(w_i)$$

Bigram model Model with a single token context ($k = 1$):

$$\mathcal{P}(w_{1..n}) \approx \prod_{i} \mathcal{P}(w_i|w_{i-1})$$

N-gram model Model with a context of $k = N - 1$ tokens: N-gram model

$$\mathcal{P}(w_{1..n}) \approx \prod_{i} \mathcal{P}(w_i|w_{i-N+1..i-1})$$

Remark. N-gram models cannot capture long-range dependencies.

Estimating N-gram probabilities Consider the bigram case, the probability that a token w_i follows w_{i-1} can be determined by counting:

$$\mathcal{P}(w_i|w_{i-1}) = \frac{\texttt{count}(w_{i-1}w_i)}{\texttt{count}(w_{i-1})}$$

Remark. N-gram models cannot handle unknown tokens.

Remark. N-gram models capture knowledge about:

- Grammar and syntax.
- Some information about the dataset (e.g., domain, genre of corpus, cultural aspects, ...).

Generation by sampling Randomly sample tokens from the distribution of a language model.

Generation by sampling

Remark. In N-gram models ($N \geq 2$), the distribution changes depending on the previously sampled tokens.

2.3 Metrics

2.3.1 Extrinsic evaluation

Extrinsic/downstream evaluation Compare the performance of different models on specific tasks.

Extrinsic evaluation

Remark. Extrinsic evaluation is the best approach for comparing different models, but it is often computationally expensive.

2.3.2 Intrinsic evaluation

Intrinsic evaluation Measure the quality of a model independently of the task.

Intrinsic evaluation

Perplexity (PP) Probability-based metric based on the inverse probability of a sequence (usually the test set) normalized by the number of words:

Perplexity

$$\mathcal{P}(w_{1..N}) = \prod_i \mathcal{P}(w_i|w_{1..i-1})$$

$$\texttt{PP}(w_{1..N}) = \mathcal{P}(w_{1..N})^{-\frac{1}{N}} \in [1, +\infty]$$

A lower perplexity represents a generally better model.

Example. For bigram models, perplexity is computed as:

$$\mathcal{P}(w_{1..N}) \approx \prod_i \mathcal{P}(w_i|w_{i-1}) \qquad \texttt{PP}(w_{1..N}) = \sqrt[N]{\prod_i \frac{1}{\mathcal{P}(w_i|w_{i-1})}}$$

Remark (Perplexity intuition)**.** Perplexity can be seen as a measure of surprise of a language model when evaluating a sequence.
Alternatively, it can also be seen as a weighted average branching factor (i.e., average number of possible next words that follow any word, accounting for their probabilities). For instance, consider a vocabulary of digits and a training corpus where every digit appears with uniform probability 0.1. The perplexity of any sequence using a

1-gram model is:

$$\texttt{PP}(w_{1..N}) = \left(0.1^N\right)^{-\frac{1}{N}} = 10$$

Now consider a training corpus where 0 occurs 91% of the time and the other digits 1% of the time. The perplexity of the sequence `0 0 0 0 0 3 0 0 0 0` is:

$$\texttt{PP}(\texttt{0 0 0 0 0 3 0 0 0 0}) = \left(0.91^9 \cdot 0.01\right)^{-\frac{1}{10}} \approx 1.73$$

Remark. Minimizing perplexity is the same as maximizing the probability of the tokens.

Remark. Perplexity can be artificially reduced by using a smaller vocabulary. Therefore, it is only reasonable to compare perplexity of models with the same vocabulary.

Remark. Perplexity is generally a bad approximation of extrinsic metrics and only works well if the test set is representative of the training data. Therefore, it is only useful to guide experiments and the final evaluation should be done through extrinsic evaluation.

2.4 N-gram model problems

2.4.1 Overfitting

N-gram models become better at modeling the training corpus for increasing values of N. This risks overfitting and does not allow to obtain a generalized model. Overfitting

Example. A 4-gram model is able to nearly perfectly generate sentences from Shakespeare's works.

2.4.2 Out-of-vocabulary tokens

There are two types of vocabulary systems:

Closed vocabulary system All words that can occur are known. Closed vocabulary

Open vocabulary system Unknown words are possible. They are usually handled using a dedicated token `<UNK>` which allows to turn an open vocabulary system into a closed one: Open vocabulary

- Use a vocabulary and model all other words as `<UNK>`.
- Model infrequent words as `<UNK>`.

Remark. The training set must contain `<UNK>` tokens to estimate its distribution as it is treated as any other token.

2.4.3 Unseen sequences

Only for n-grams that occur enough times a representative probability can be estimated. For increasing values of n, the sparsity grows causing many unseen n-grams that produce a probability of 0, with the risk of performing divisions by zero (e.g., perplexity) or zeroing probabilities (e.g., when applying the chain rule).

Laplace smoothing Adds 1 to all counts and renormalizes them. Given a vocabulary V and an N-gram model, smoothing is done as follows: Laplace smoothing

$$\mathcal{P}_{\text{Laplace}}(w_i | w_{i-N+1..i-1}) = \frac{\texttt{count}(w_{i-N+1..i-1} w_i) + 1}{\texttt{count}(w_{i-N+1..i-1}) + |V|}$$

Alternatively, by only changing the numerator, it can be formulated using an adjusted count as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-N+1..i-1}) = \frac{c^*}{\texttt{count}(w_{i-N+1..i-1})}$$
$$c^* = \big(\texttt{count}(w_{i-N+1..i-1}w_i) + 1\big)\frac{\texttt{count}(w_{i-N+1..i-1})}{\texttt{count}(w_{i-N+1..i-1}) + |V|}$$

where $\frac{\texttt{count}(w_{i-N+1..i-1})}{\texttt{count}(w_{i-N+1..i-1})+|V|}$ is a normalization factor.

Example. For a 2-gram model, Laplace smoothing is computed as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-1}) = \frac{\texttt{count}(w_{i-1}w_i) + 1}{\texttt{count}(w_{i-1}) + |V|}$$

Or by using the adjusted count as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-1}) = \frac{c^*}{\texttt{count}(w_{i-1})} \qquad c^* = \big(\texttt{count}(w_{i-1}w_i) + 1\big)\frac{\texttt{count}(w_{i-1})}{\texttt{count}(w_{i-1}) + |V|}$$

3 Text classification

3.1 Common tasks

Sentiment analysis/Opinion mining Detection of attitudes. It can involve detecting:

Sentiment analysis/Opinion mining

- The holder of attitude (i.e., the source).
- The target of attitude (i.e., the aspect).
- The type of attitude (e.g., positive and negative).
- The text containing the attitude.

Spam detection

Language identification

Authorship attribution

Subject category classification

3.2 Classification

Classification task Given an input x and a set of possible classes $Y = \{y_1, \dots, y_M\}$, a classifier determines the class $\hat{y} \in Y$ associated to x.

Classification task

Classification can be:

Rule-based Based on fixed (possibly handwritten) rules.

Rule-based

Example. Blacklist, whitelist, regex, ...

In-context learning Provide a decoder (i.e., generative) large language model a prompt describing the task and the possible classes.

In-context learning

Example. Zero-shot learning, few-shot learning, ...

Supervised machine learning Use a training set of N labeled data $\{(d_i, c_i)\}$ to fit a classifier.

Supervised machine learning

An ML model can be:

Generative Informally, it learns the distribution of the data.

Discriminative Informally, it learns to exploit the features to determine the class.

3.3 Naive Bayes

Bag-of-words (BoW) Represents a document using the frequencies of its words.

Bag-of-words (BoW)

Given a vocabulary V and a document d, the bag-of-words embedding of d is a vector in $\mathbb{N}^{|V|}$ where the i-th position contains the number of occurrences of the i-th token in d.

Multinomial naive Bayes classifier Generative probabilistic classifier based on the assumption that features are independent given the class. Multinomial naive Bayes classifier

Given a document $d = \{w_1, \dots, w_n\}$, a naive Bayes classifier returns the class $\hat{c}$ with maximum posterior probability:

$$\begin{aligned}
\hat{c} &= \arg\max_{c \in C} \mathcal{P}(c|d) \\
&= \arg\max_{c \in C} \underbrace{\mathcal{P}(d|c)}_{\text{likelihood}} \underbrace{\mathcal{P}(c)}_{\text{prior}} \\
&= \arg\max_{c \in C} \mathcal{P}(w_1, \dots, w_n|c)\mathcal{P}(c) \\
&= \arg\max_{c \in C} \prod_i \mathcal{P}(w_i|c)\mathcal{P}(c) \\
&= \arg\max_{c \in C} \sum_i \log \mathcal{P}(w_i|c) \log \mathcal{P}(c)
\end{aligned}$$

Given a training set D with N_c classes and a vocabulary V, $\mathcal{P}(w_i|c)$ and $\mathcal{P}(c)$ are determined during training by maximum likelihood estimation as follows:

$$\mathcal{P}(c) = \frac{N_c}{|D|} \qquad \mathcal{P}(w_i|c) = \frac{\texttt{count}(w_i, c)}{\sum_{v \in V} \texttt{count}(v, c)}$$

where $\texttt{count}(w, c)$ counts the occurrences of the word w in the training samples with class c.

Remark. Laplace smoothing is used to avoid zero probabilities.

Remark. Stop words can be removed from the training set as they are usually not relevant.

Remark. The likelihood part of the equation ($\sum_i \log \mathcal{P}(w_i|c)$) can be seen as a set of class-specific 1-gram language models.

Example. Given the following training set for sentiment analysis with two classes:

Class	Document
-	`just plain boring`
-	`entirely predictable and lacks energy`
-	`no surprises and very few laughs`
+	`very powerful`
+	`the most fun film of the summer`

We want to classify the sentence "`predictable with no fun`". Excluding stop words (i.e., `with`), we need to compute:

$$\begin{aligned}
&\mathcal{P}(\texttt{+}|\texttt{predictable with no fun}) = \mathcal{P}(\texttt{+})\mathcal{P}(\texttt{predictable}|\texttt{+})\mathcal{P}(\texttt{no}|\texttt{+})\mathcal{P}(\texttt{fun}|\texttt{+}) \\
&\mathcal{P}(\texttt{-}|\texttt{predictable with no fun}) = \mathcal{P}(\texttt{-})\mathcal{P}(\texttt{predictable}|\texttt{-})\mathcal{P}(\texttt{no}|\texttt{-})\mathcal{P}(\texttt{fun}|\texttt{-})
\end{aligned}$$

A vocabulary of 20 tokens can be used to represent the training samples. The required likelihoods and priors with Laplace smoothing are computed as:

$$\mathcal{P}(\texttt{+}) = \frac{2}{5} \qquad \mathcal{P}(\texttt{predictable}|\texttt{+}) = \frac{0+1}{9+20} \quad \mathcal{P}(\texttt{no}|\texttt{+}) = \frac{0+1}{9+20} \quad \mathcal{P}(\texttt{fun}|\texttt{+}) = \frac{1+1}{9+20}$$
$$\mathcal{P}(\texttt{-}) = \frac{3}{5} \qquad \mathcal{P}(\texttt{predictable}|\texttt{-}) = \frac{1+1}{14+20} \quad \mathcal{P}(\texttt{no}|\texttt{-}) = \frac{1+1}{14+20} \quad \mathcal{P}(\texttt{fun}|\texttt{-}) = \frac{0+1}{14+20}$$

3.3.1 Optimizations

Possible optimizations for naive Bayes applied to sentiment analysis are the following:

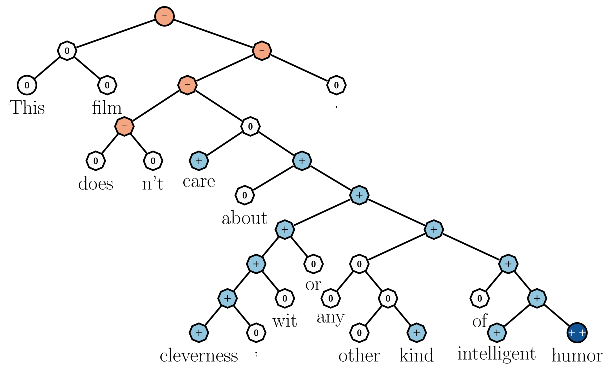
Binarization Generally, the information regarding the occurrence of a word is more important than its frequency. Therefore, instead of applying bag-of-words by counting, it is possible to produce a one-hot encoded vector to indicate which words are in the document. Binarization

Negation encoding To encode negations, two approaches can be taken: Negation encoding

Negation annotation Add to negated words an annotation so that they are treated as a new word.

> **Example.** Prepend `NOT_` to each word between a negation and the next punctuation:
>
> $$\text{didn't like this movie.} \mapsto \text{didn't } \texttt{NOT\_}\text{like } \texttt{NOT\_}\text{this } \texttt{NOT\_}\text{movie.}$$

Parse tree Build a tree to encode the sentiment and interactions of the words. By propagating the sentiments bottom-up it is possible to determine the overall sentiment of the sequence.

> **Example.** The parse tree for the sentence "`This film doesn't care about cleverness, wit or any other kind of intelligent humor.`" is the following:
>
> 
>
>
> Due to the negation (`doesn't`), the whole positive sequence is negated.

Sentiment lexicon If training data is insufficient, external domain knowledge, such as sentiment lexicon, can be used. Sentiment lexicon

> **Example.** A possible way to use a lexicon is to count the number of positive and negative words according to that corpus.

> **Remark.** Possible ways to create a lexicon are through:
>
> - Expert annotators.
> - Crowdsourcing in a two-step procedure:
>   1. Ask questions related to synonyms (e.g., which word is closest in meaning to *startle*?).
>   2. Rate the association of words with emotions (e.g., how does *startle* associate with *joy*, *fear*, *anger*, . . . ?).
> - Semi-supervised induction of labels from a small set of annotated data (i.e.,

seed labels). It works by looking for words that appear together with the ones with a known sentiment.

- Supervised learning using annotated data.

3.3.2 Properties

Naive Bayes has the following properties:

- It is generally effective with short sequences and fewer data samples.
- It is robust to irrelevant features (i.e., words that appear in both negative and positive sentences) as they cancel out each other.
- It has good performance in domains with many equally important features (contrarily to decision trees).
- The independence assumption might produce overestimated predictions.

Remark. Naive Bayes is a good baseline when experimenting with text classifications.

3.4 Logistic regression

Features engineering Determine features by hand from the data (e.g., number of positive and negative lexicon).

Features engineering

Binary logistic regression Discriminative probabilistic model that computes the joint distribution $\mathcal{P}(c|d)$ of a class c and a document d.

Binary logistic regression

Given the input features $\mathbf{x} = [x_1, \dots, x_n]$, logistic regression computes the following:

$$\sigma\left(\sum_{i=1}^{n} w_i x_i\right) + b = \sigma(\mathbf{wx} + b)$$

where σ is the sigmoid function.

Loss The loss function should aim to maximize the probability of predicting the correct label $\hat{y}$ given the observation $\mathbf{x}$. This can be expressed as a Bernoulli distribution:

$$\mathcal{P}(y|x) = \hat{y}^y(1-\hat{y})^{1-y} = \begin{cases} 1-\hat{y} & \text{if } y = 0 \\ \hat{y} & \text{if } y = 1 \end{cases}$$

By applying a log-transformation and inverting the sign, this corresponds to the cross-entropy loss in the binary case:

$$\mathcal{L}_{\text{BCE}}(\hat{y}, y) = -\log \mathcal{P}(y|x) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

Optimization As cross-entropy is convex, SGD is well suited to find the parameters $\boldsymbol{\theta}$ of a logistic regressor f over batches of m examples by solving:

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathcal{L}_{\text{BCE}}(\hat{y}^{(i)}, f(x^{(i)}; \boldsymbol{\theta})) + \alpha\mathcal{R}(\boldsymbol{\theta})$$

where α is the regularization factor and $\mathcal{R}(\boldsymbol{\theta})$ is the regularization term. Typical regularization approaches are:

Ridge regression (L1) $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum_{j=1}^{n} \boldsymbol{\theta}_j^2$.

Lasso regression (L2) $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_{j=1}^{n} |\boldsymbol{\theta}_j|$.

Multinomial logistic regression Extension of logistic regression to the multi-class case. The joint probability becomes $\mathcal{P}(y = c|x)$ and softmax is used in place of the sigmoid.

Multinomial logistic regression

Cross-entropy is extended over the classes C:

$$\mathcal{L}_{\text{CE}}(\hat{y}, y) = -\sum_{c \in C} \mathbb{1}\{y = c\} \log \mathcal{P}(y = c|x)$$

3.4.1 Properties

Logistic regression has the following properties:

- It is generally effective with large documents or datasets.
- It is robust to correlated features.

Remark. Logistic regression is also a good baseline when experimenting with text classifications.
As they are lightweight to train, it is a good idea to test both naive Bayes and logistic regression to determine the best baseline for other experiments.

3.5 Metrics

3.5.1 Binary classification

Contingency table 2×2 table matching predictions to ground truths. It contains true positives (`TP`), false positives (`FP`), false negatives (`FN`), and true negatives (`TN`).

Contingency table

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

Recall

Precision $\frac{\texttt{TP}}{\texttt{TP}+\texttt{FP}}$.

Precision

Accuracy $\frac{\texttt{TP}+\texttt{TN}}{\texttt{TP}+\texttt{FP}+\texttt{FN}+\texttt{TN}}$.

Accuracy

Remark. Accuracy is a reasonable metric only when classes are balanced.

F1 score $\frac{2 \cdot \texttt{recall} \cdot \texttt{precision}}{\texttt{recall}+\texttt{precision}}$.

F1 score

3.5.2 Multi-class classification

Confusion matrix $c \times c$ table matching predictions to ground truths.

Confusion matrix

Precision/Recall Precision and recall can be defined class-wise (i.e., consider a class as the positive label and the others as the negative).

Precision/Recall

Micro-average precision/recall Compute the contingency table of each class and collapse them into a single table. Compute precision or recall on the pooled contingency table.

Micro-average precision/recall

Remark. This approach is sensitive to the most frequent class.

Macro-average precision/recall Compute precision or recall class-wise and then average over the classes.

Macro-average precision/recall

| **Remark.** This approach is reasonable if the classes are equally important.

| **Remark.** Macro-average is more common in NLP.

Example.

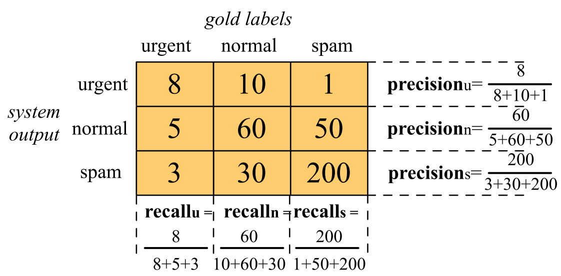


Figure 3.1: Confusion matrix

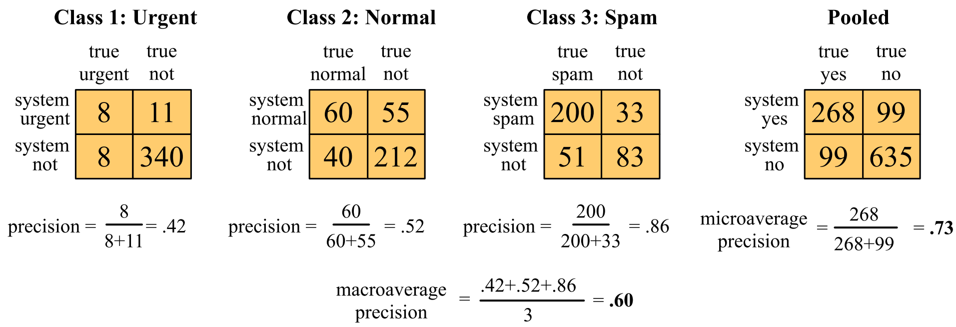


Figure 3.2: Class-wise contingency tables, pooled contingency table, and micro/macro-average precision

3.5.3 Cross-validation

n-fold cross-validation Tune a classifier on different sections of the training data:

n-fold cross-validation

1. Randomly choose a training and validation set.
2. Train the classifier.
3. Evaluate the classifier on a held-out test set.
4. Repeat for n times.

3.5.4 Statistical significance

p-value Measure to determine whether a model A is outperforming a model B on a given test set by chance (i.e., test the null hypothesis H_0 or, in other words, test that there is no relation between A and B).

p-value

Given:

- A test set x,
- A random variable X over the test sets (i.e., another test set),
- Two models A and B, such that A is better than B by $\delta(x)$ on the test set x,

the p-value is defined as:

$$p\text{-value}(x) = \mathcal{P}\left(\delta(X) > \delta(x) | H_0\right)$$

There are two cases:

- p-value(x) is big: the null hypothesis holds (i.e., $\mathcal{P}\left(\delta(X) > \delta(x)\right)$ is high under the assumption that A and B are not related), so A outperforms B by chance.
- p-value(x) is small (i.e., < 0.05 or < 0.01): the null hypothesis is rejected, so A actually outperforms B.

Bootstrapping test Approach to compute p-values. Bootstrapping test

Given a test set x, multiple virtual test sets $\bar{x}^{(i)}$ are created by sampling with replacement (it is assumed that the new sets are representative). The performance difference $\delta(\cdot)$ is computed between two models and the p-value is determined as the frequency of:

$$\delta(\bar{x}^{(i)}) > 2\delta(x)$$

Remark. $\delta(x)$ is doubled due to theoretical reasons.

Example. Consider two models A and B, and a test set x with 10 samples. From x, multiple new sets (in this case of the same size) can be sampled. In the following table, each cell indicates which model correctly predicted the class:

	1	2	3	4	5	6	7	8	9	10	$A\%$	$B\%$	$\delta(\cdot)$
x	AB	A	AB	B	A	B	A	AB	–	A	0.7	0.5	0.2
$\bar{x}^{(1)}$	A	AB	A	B	B	A	B	AB	–	AB	0.6	0.6	0.0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

A possible way to sample $\bar{x}^{(1)}$ is (w.r.t. the indexes of the examples in x) $[2, 3, 3, 2, 4, 6, 2, 4, 1, 9, 1]$.

3.6 Affective meaning

The affective meaning of a text corpus can vary depending on:

Personality traits Stable behavior and personality of a person (e.g., nervous, anxious, reckless, ...).

Attitude Enduring sentiment towards objects or people (e.g., liking, loving, hating, ...).

Interpersonal stance Affective stance taken in a specific interaction (e.g., distant, cold, warm, ...).

Mood Affective state of low intensity and long duration often without an apparent cause (e.g., cheerful, gloomy, irritable, ...).

Emotion Brief response to an external or internal event of major significance (e.g., angry, sad, joyful, ...).

Remark. Emotion is the most common subject in affective computing.

3.6.1 Emotion

Theory of emotion There are two main theories of emotion:

- **Basic emotions** Discrete and fixed range of atomic emotions. Basic emotions

  Remark. Emotions associated to a word might be in contrast. For instance, in the NRC Word-Emotion Association Lexicon the word `thirst` is associated to both *anticipation* and *surprise*.

- **Continuum emotions** Describe emotions in a 2 or 3 dimensional space with the following features: Continuum emotions

  - **Valence** Pleasantness of a stimulus.
  - **Arousal** Intensity of emotion.
  - **Dominance** Degree of control on the stimulus.

  Remark. Valence is often used as a measure of sentiment.

4 Semantics embedding

4.1 Traditional semantic representation

Lemma/citation form Syntactic form of a word. Lemma

> **Example.** The word `pipe`.

Word sense Meaning component of a word. Word sense

Polysemous lemma Lemma with multiple senses.

> **Example.** Possible senses of the word `pipe` are: the music instrument, the conduit to transport material, ...

Supersense Semantic category for senses. Supersense

Word sense disambiguation (WSD) Task of determining the correct sense of a word. Word sense disambiguation (WSD)

4.1.1 Sense relations

Synonym Relation of (near) identity between two senses of two different words (i.e., same propositional meaning). Synonym

> **Remark** (Principle of contrast)**.** A different linguistic form is probably due to some, maybe subtle, difference in meaning.

Antonym Relation of opposition, with respect to one feature of meaning, between two senses. More specifically, antonyms can be: Antonym

- An opposition between two ends of a scale (e.g., `long/short`).
- A reversive (e.g., `up/down`).

Subordination Specificity (i.e., is-a) relation between two senses. Subordination

> **Example.** `car` is a subordinate of `vehicle`.

Superordination Generalization relation between two senses. Superordination

> **Example.** `furniture` is a superordinate of `lamp`.

Meronym Part-of relation between two senses. Meronym

> **Remark.** Relations among word senses can be seen as a graph.

4.1.2 Common ontologies

WordNet Database of semantic relations of English words. WordNet

BabelNet Multilingual database of semantic relations. BabelNet

4.1.3 Word relations

Word similarity Measure the meaning similarity of words (i.e., relation between words and not senses). Word similarity

Remark. Working with words is easier than senses.

Example. Cat and dog are not synonyms but have similar meaning (i.e., pets).

Word relatedness Measure the context relation of words. Word relatedness

Example. `car/bike` are similar while `car/fuel` are related but not similar.

Semantic field Words that cover a particular domain and have structured relations with each other. Semantic field

Example. In the context of a hospital, `surgeon`, `scalpel`, `nurse`, `anesthetic`, and `hospital` belong to the same semantic field.

Topic model Unsupervised method to cluster the topics in a document based on how a word is used in its context. Topic model

Semantic frames Words that describe the perspective or participants of a particular event. Semantic frames

Example. In a commercial transaction, a `buyer` trades `money` with a `seller` in return of some `good or service`.

Semantic role labeling (SRL) Task of determining the frames and their semantic role. Semantic role labeling (SRL)

4.2 Vector semantics

Connotation Affective meaning of a word. Connotation

Remark. As described in Section 3.6, emotions can be represented in a vector space. Therefore, word meanings can also be represented as vectors.

Vector semantics intuitions Vector semantics lay on two intuitions:

Distributionalism intuition The meaning of a word is defined by its environment or distribution (i.e., neighboring words). Words with a similar distribution are likely to have the same meaning. Distributionalism intuition

Vector intuition Define the meaning of a word as a point in an N-dimensional space. Vector intuition

Embedding Vector representation of a word where words with a similar meaning are nearby in the vector space. Embedding

Two common embedding models are:

TF-IDF Sparse embedding based on the counts of nearby words. TF-IDF

Word2vec Dense embedding learned by training a classifier to distinguish nearby and far-away words. Word2vec

4.2.1 Sparse embeddings

Co-occurrence matrix Matrix representing the frequency that words occur with the others. Co-occurrence matrix

Different design choices can be considered:

- Matrix design.
- Reweighing.

- Dimensionality reduction.
- Vector comparison metric.

Matrix design Shape and content of the co-occurrence matrix.

Term-document matrix Given a vocabulary V and a set of documents D, a term-document matrix has shape $|V| \times |D|$ and counts the occurrences of each word in each document.

Term-document matrix

Remark. This representation allows to encode both documents (i.e., by considering the matrix column-wise) and words (i.e., by considering the matrix row-wise).

Example. An excerpt of a possible term-document matrix for Shakespeare is:

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
`battle`	1	0	7	13
`good`	114	80	62	89
`fool`	36	58	1	4
`wit`	20	15	2	3

The representation for the document *As You Like It* is $[1, 114, 36, 20]$, while the representation of the word `battle` is $[1, 0, 7, 13]$.

Word-word matrix Given a vocabulary V, a word-word matrix has shape $|V| \times |V|$. Rows represent target words and columns are context words. Given a training corpus, the word at each row is represented by counting its co-occurrences with the others within a context of N words.

Word-word matrix

Remark. A larger context window captures more semantic information. A smaller window captures more syntactic information.

Example. A possible word-word matrix is:

	`aardvark`	...	`computer`	`data`	`result`	`pie`	`sugar`	...
`cherry`	0	...	2	8	9	442	25	...
`strawberry`	0	...	0	0	1	60	19	...
`digital`	0	...	1670	1683	85	5	4	...
`information`	0	...	3325	3982	378	5	13	...

Reweighing Rescale the vectors to emphasize important features and down-weigh irrelevant words.

Remark (Frequency paradox)**.** Raw frequencies are not an ideal representation for words as they are skewed and not discriminative. Moreover, overly frequent words (e.g., stop words) do not provide context information.

Term frequency-inverse document frequency (TF-IDF) Based on term-document occurrences. Given a word t and a document d, it is computed as:

Term frequency-inverse document frequency (TF-IDF)

$$\texttt{tf-idf}(t, d) = \texttt{tf}(t, d) \cdot \texttt{idf}(t)$$

where:

Term frequency (tf) Log-transformed frequency count of a word t in a document d:

$$\texttt{tf}(t,d) = \begin{cases} 1 + \log_{10}(\texttt{count}(t,d)) & \text{if } \texttt{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Inverse document frequency (idf) Inverse occurrence count of a word t across all documents:

$$\texttt{idf}(t) = \log_{10}\left(\frac{N}{\texttt{df}_t}\right)$$

where $\texttt{df}_t$ is the number of documents in which the term t occurs.

Remark. Words that occur in a few documents have a high `idf`. Therefore, stop words, which appear often, have a low `idf`.

Example. Consider the term-document matrix with `tf` in parentheses:

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
`battle`	1 (1)	0 (0)	7 (1.845)	13 (2.114)
`good`	114 (3.057)	80 (2.903)	62 (2.792)	89 (2.949)
`fool`	36 (2.553)	58 (2.763)	1 (1)	4 (1.602)
`wit`	20 (2.301)	15 (2.176)	2 (1.301)	3 (1.477)

Assume that the `df` and `idf` of the words are:

Word	`df`	`idf`
`battle`	21	0.246
`good`	37	0
`fool`	36	0.012
`wit`	34	0.037

The resulting TF-IDF weighted matrix is:

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
`battle`	0.246	0	0.454	0.520
`good`	0	0	0	0
`fool`	0.030	0.033	0.001	0.002
`wit`	0.085	0.081	0.048	0.054

Positive point-wise mutual information (PPMI) Based on term-term occurrences. Given a word w and a context word c, it determines whether they are correlated or occur by chance as follows:

Positive point-wise mutual information (PPMI)

$$\texttt{PPMI}(w,c) = \max\{\texttt{PMI}(w,c), 0\}$$

where:

Point-wise mutual information (PMI)

$$\texttt{PMI}(w,c) = \log_2\left(\frac{\mathcal{P}(w,c)}{\mathcal{P}(w)\mathcal{P}(c)}\right) \in (-\infty, +\infty)$$

where:

- The numerator is the probability that w and c co-occur by correlation.
- The denominator is the probability that w and c co-occur by chance.

Remark. PMI > 1 indicates correlated co-occurrence. Otherwise, it is by chance.

Remark (Weighting PPMI)**.** PMI is biased towards infrequent events and returns very high values for them. This can be solved by either:

- Using add-k smoothing (typically, $k \in [0.1, 3]$).
- Slightly increasing the probability of rare context words such that $\mathcal{P}_\alpha(c) = \frac{\texttt{count}(c)^\alpha}{\sum_{c'} \texttt{count}(c')^\alpha}$ (typically, $\alpha = 0.75$).

Example. Consider the term-term matrix:

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(c)	4977	5673	473	512	61	11716

The PPMI between `information` and `data` con be computed as:

$$\begin{aligned}
\mathcal{P}(\texttt{information}, \texttt{data}) &= \frac{3982}{11716} = 0.3399 \\
\mathcal{P}(\texttt{information}) &= \frac{7703}{11716} = 0.6575 \\
\mathcal{P}(\texttt{data}) &= \frac{5673}{11716} = 0.4872 \\
\texttt{PPMI}(\texttt{information}, \texttt{data}) &= \max\left\{ \log_2\left(\frac{0.3399}{0.6575 \cdot 0.4872}\right), 0 \right\} = 0.0944
\end{aligned}$$

Remark. Reweighing loses information about the magnitude of the counts.

Dimensionality reduction Reduce the dimensionality of the embeddings.

Vector comparison Metric to determine the distance of two embeddings.

Dot product $\mathbf{w} \cdot \mathbf{v} = \sum_{i=1}^{n} w_i v_i$.

Length Compare the length $|\mathbf{v}| = \sqrt{\sum_{i=1}^{n} v_i^2}$ of the vectors.

Cosine similarity $\frac{\mathbf{w} \cdot \mathbf{v}}{|\mathbf{w}|\,|\mathbf{v}|}$.

4.2.2 Dense non-contextual embeddings

Remark. Dense embeddings are usually:

- Easier to process with machine learning algorithms.
- Able to generalize better than simply counting.
- Handle synonyms better.

Neural language modeling Use a neural network to predict the next word w_{n+1} given an input sequence $w_{1..n}$. The general flow is the following: Neural language modeling

1. Encode the input words into one-hot vectors ($\mathbb{R}^{|V| \times n}$).

2. Project the input vectors with an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{d \times |V|}$ that encodes them into d-dimensional vectors.
3. Pass the embedding into the hidden layers.
4. The final layer is a probability distribution over the vocabulary ($\mathbb{R}^{|V| \times 1}$).

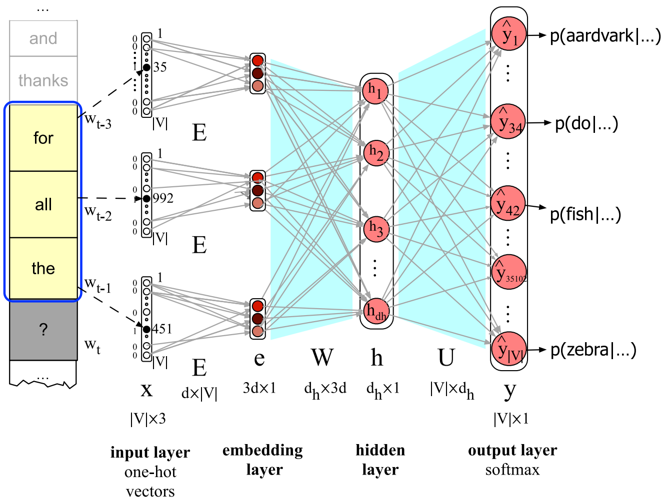


Figure 4.1: Example of neural language model with a context of 3 tokens

Remark. The embedding matrix $\boldsymbol{E}$ can be used independently to embed words. In fact, by construction, the i-th column of $\boldsymbol{E}$ represents the embedding of the i-th token of the vocabulary.

Training Given a text corpus, training is done sequentially in a self-supervised manner by sliding a context window over the sequence. At each iteration, the next word is predicted and cross-entropy is used as loss.

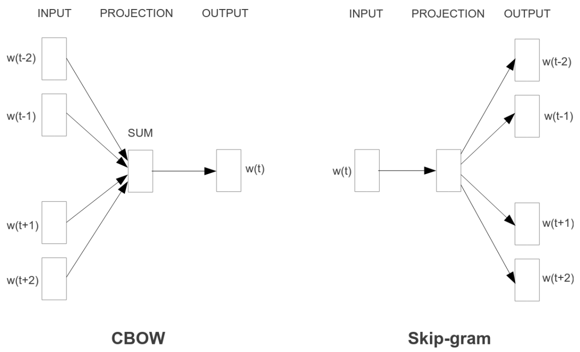
Remark. The initial embedding matrix is usually initialized using statistical methods and not randomly.

Word2vec Word embedding framework that encodes a target word based on the context words near it. Word2vec

Two training variants are available in Word2vec:

Continuous bag-of-words (CBOW) Given the context words, predict the target word.

Skip-gram Given the target word, predict the (position independent) context words.



Skip-gram model Given a context word c and a target word w, a classifier is trained to determine whether c appears in the context of w. After training, the weights of the classifier are used as the skip-gram model to embed words. Skip-gram model

Remark. In practice, for an easier optimization, the skip-gram model learns two sets of embeddings $\boldsymbol{W} \in \mathbb{R}^{|V| \times d}$ and $\boldsymbol{C} \in \mathbb{R}^{|V| \times d}$ for the target and context words, respectively. Therefore, it has two sets of parameters $\boldsymbol{\theta} = \langle \boldsymbol{W}, \boldsymbol{C} \rangle$. At the end, they can either be averaged, concatenated, or one can be dropped.

Training (softmax) Given the target word w and context word c, and their embeddings $\mathbf{w}$ and $\mathbf{c}$, the skip-gram model computes their similarity as the dot product. The probability that c is in the context of w is then computed though a softmax as:

$$\mathcal{P}(c|w;\boldsymbol{\theta}) = \frac{\exp(\mathbf{c} \cdot \mathbf{w})}{\sum_{v \in V} \exp(\mathbf{v} \cdot \mathbf{w})}$$

Given a training sequence $w_1, \dots, w_T$ and a context window of size m, training is done by iterating over each possible target word w_t and considering the conditional probabilities of its neighbors. Then, the loss is defined as the average negative log-likelihood defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log\left(\mathcal{P}(w_{t+j}|w_t;\boldsymbol{\theta})\right)$$

Remark. Due to the normalization factor over the whole vocabulary, using softmax for training is expensive.

Training (negative sampling) Use a binary logistic regressor as classifier. The two classes are: Skip-gram with negative sampling (SGNS)

- Context words within the context window (positive label).
- Words randomly sampled (negative label).

The probabilities can be computed as:

$$\mathcal{P}(+|w,c;\boldsymbol{\theta}) = \sigma(\mathbf{c} \cdot \mathbf{w}) \qquad \mathcal{P}(-|w,c;\boldsymbol{\theta}) = 1 - \mathcal{P}(+|w,c;\boldsymbol{\theta})$$

It is assumed context-independent words, therefore, if the context is a sequence, the probability is computed as follows:

$$\mathcal{P}(+|w,c_{1..L};\boldsymbol{\theta}) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i \cdot \mathbf{w})$$

At each iteration, the batch is composed of a single positive examples and K negative examples randomly sampled according to their weighted unigram probability $\mathcal{P}_\alpha(w) = \frac{\texttt{count}(w)^\alpha}{\sum_{v \in V} \texttt{count}(v)^\alpha}$ (α it used to give rarer words a slightly higher probability).

Given a batch, the loss is defined as:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= -\log\left(\mathcal{P}(+|w,c^{\text{pos}};\boldsymbol{\theta}) \prod_{i=1}^{K} \mathcal{P}(-|w,c_i^{\text{neg}};\boldsymbol{\theta})\right) \\
&= -\left(\log\left(\sigma(\mathbf{c}^{\text{pos}} \cdot \mathbf{w})\right) + \sum_{i=1}^{K} \log\left(\sigma(-\mathbf{c}_i^{\text{neg}} \cdot \mathbf{w})\right)\right)
\end{aligned}$$

fastText Extension of Word2vec based on subwords to deal with out-of-vocabulary words. fastText

A word is represented both as itself and a bag of n-grams. Both whole words and n-grams have an embedding. The overall embedding of a word is represented through the sum of its constituent n-grams.

Example. With $n = 3$, the word `where` is represented both as `<where>` and `<wh, whe, her, ere, re>` (`<` and `>` are boundary characters).

GloVe Based on the term-term co-occurrence (within a window) probability matrix that indicates for each word its probability of co-occuring with the other words. GloVe

Similarly to Word2vec, the objective is to learn two sets of embeddings $\boldsymbol{\theta} = \langle \boldsymbol{W}, \boldsymbol{C} \rangle$ such that their similarity is close to their log-probability of co-occurring. Given the term-term matrix $\boldsymbol{X}$, the loss for a target word w and a context word c is defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = (\mathbf{c} \cdot \mathbf{w} - \log(\boldsymbol{X}[c, w]))^2$$

Remark. Empirically, for GloVe it has been observed that the final embedding matrix obtained as $\boldsymbol{W} + \boldsymbol{C}$ works better.

Example. A possible term-term co-occurrence probability for the words `ice` and `steam` is the following:

	$k =$ `solid`	$k =$ `gas`	$k =$ `water`	$k =$ `fashion`
$\mathcal{P}(k \vert$ `ice`$)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$\mathcal{P}(k \vert$ `steam`$)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}

`ice` is more likely to co-occur with `solid` while `steam` is more likely to co-occur with `gas`. GloVe uses this information when determining the embeddings.

4.3 Embeddings properties

4.3.1 Embeddings similarity

Context size The window size used to collect counts or determine context words can result in different embeddings. Context size

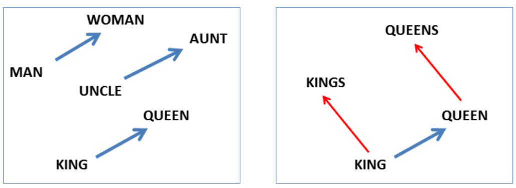
As a general rule, smaller windows tend to capture more syntactic features while a larger window encodes more topically related but not necessarily similar words.

Similarity orders Two words have: Similarity orders

First-order co-occurrence If they are nearby each other.

Second-order co-occurrence If they have similar context words.

Relational similarity Dense embeddings are able to capture relational meanings. Relational similarity



Parallelogram model Given the problem "a is to b as a^* is to b^*" ($a : b :: a^* : b^*$), the parallelogram model solves it as:

$$b^* = \arg\min_x \texttt{distance}(x, b - a + a^*)$$

Example. In Word2vec, the following operation between embeddings can be done:

$$\texttt{Paris} - \texttt{France} + \texttt{Italy} \approx \texttt{Rome}$$

Remark. Even if it sometimes works, parallelogram model is not guaranteed to always produce the expected result.

4.3.2 Embeddings analysis

Word history Trained on different corpora, dense embeddings can provide a semantic evolution of words by analyzing its neighboring embeddings. Word history

Example.

Figure 4.2: Neighboring embeddings of the same words encoded using Word2vec trained on different corpora from different decades

Example.

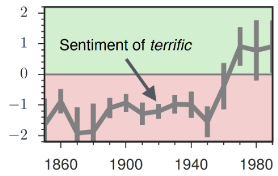


Figure 4.3: Sentiment for the word `terrific` analyzed using the embeddings obtained by training on different corpora

Cultural bias Embeddings reflect implicit biases in the training corpus. Cultural bias

Implicit association test Determine how associated are concepts and attributes.

Example. Using the parallelogram model to solve:

$$\texttt{father} : \texttt{doctor} :: \texttt{mother} : x$$

finds as the closest words $x =$ `homemaker`, `nurse`, `receptionist`, ...

Example. African-American and Chinese names are closer to unpleasant words compared to European-American names.

Example. Using the Google News dataset as training corpus, there is a correlation between the women bias of the jobs embeddings and the percentage of women over men in those jobs.

Woman bias for a word w is computed as:

$$d_{\text{women}}(w) - d_{\text{men}}(w)$$

where $d_{\text{women}}(w)$ is the average embedding distance between words representing women (e.g., **`she`**, **`female`**, ...) and the word w. The same idea is applied to $d_{\text{men}}(w)$.

Figure 4.4: Relationship between the relative percentage of women in an occupation and the women bias.

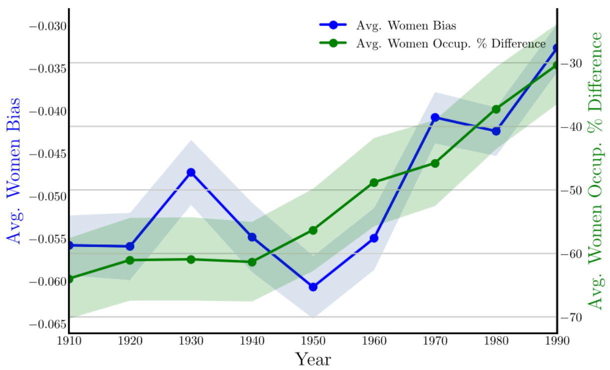


Figure 4.5: Average women bias vs average women occupation difference over time.

5 Recurrent neural networks

5.1 Architectures

5.1.1 (Elman) recurrent neural network

Recurrent neural network (RNN) Neural network that processes a sequential input. At each iteration, an input is fed to the network and the hidden activation is computed considering both the input and the hidden activation of the last iteration.

Recurrent neural network (RNN)

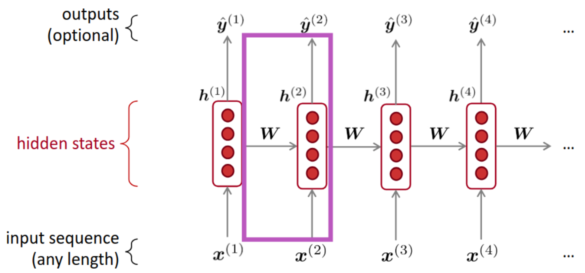
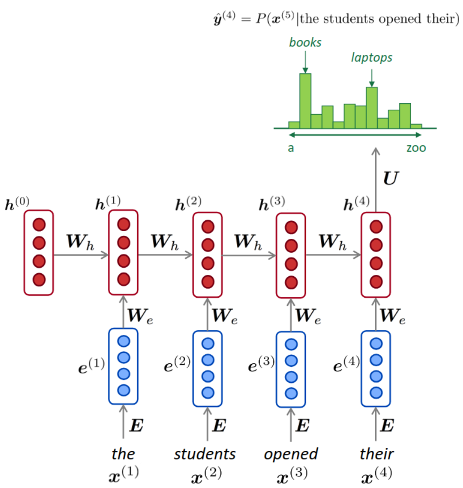


Figure 5.1: RNN unrolled in time

RNN language model (RNN-LM) Given an input word $w^{(t)}$, an RNN-LM does the following:

RNN language model (RNN-LM)

1. Compute the embedding $\mathbf{e}^{(t)}$ of $w^{(t)}$.
2. Compute the hidden state $\mathbf{h}^{(t)}$ considering the hidden state $\mathbf{h}^{(t-1)}$ of the previous step:
$$\mathbf{h}^{(t)} = f(\boldsymbol{W}_e\mathbf{e}^{(t)} + \boldsymbol{W}_h\mathbf{h}^{(t-1)} + b_1)$$
3. Compute the output vocabulary distribution $\hat{\mathbf{y}}^{(t)}$:
$$\hat{\mathbf{y}}^{(t)} = \texttt{softmax}(\boldsymbol{U}\mathbf{h}^{(t)} + b_2)$$
4. Repeat for the next token.



Remark. RNN-LMs allows to generate the output autoregressively.

Training Given the predicted distribution $\hat{\mathbf{y}}^{(t)}$ and ground-truth $\mathbf{y}^{(t)}$ at step t, the loss is computed as the cross-entropy:

$$\mathcal{L}^{(t)}(\boldsymbol{\theta}) = -\sum_{v \in V} \mathbf{y}_v^{(t)} \log \left(\hat{\mathbf{y}}_v^{(t)} \right)$$

Teacher forcing During training, as the ground-truth is known, the input at each step is the correct token even if the previous step outputted the wrong value.

Teacher forcing

Remark. This allows to stay close to the ground-truth and avoid completely wrong training steps.

Remark. RNNs grow in width and not depth, and cannot be parallelized.

5.1.2 Long short-term memory

Remark (Vanishing gradient)**.** In RNNS, the gradient of distant tokens vanishes through time. Therefore, long-term effects are hard to model.

Long short-term memory (LSTM) Architecture where at each step t outputs:

Long short-term memory (LSTM)

Hidden state $\mathbf{h}^{(t)} \in \mathbb{R}^n$ as in RNNs.

Cell state $\mathbf{c}^{(t)} \in \mathbb{R}^n$ with the responsibility of long-term memory.

Gates Non-linear operators to manipulate the cell state (in the following part, $\boldsymbol{W}_*$, $\boldsymbol{U}_*$, and $\mathbf{b}_*$ are parameters).

Forget gate Controls what part of the cell state to keep:

Forget gate

$$\mathbf{f}^{(t)} = \sigma \left(\boldsymbol{W}_f \mathbf{h}^{(t-1)} + \boldsymbol{U}_f \mathbf{x}^{(t)} + \mathbf{b}_f \right)$$

Input gate Controls what part of the input to writer in the cell state:

Input gate

$$\mathbf{i}^{(t)} = \sigma \left(\boldsymbol{W}_i \mathbf{h}^{(t-1)} + \boldsymbol{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i \right)$$

Output gate Controls what part of the cell state to include in the output hidden state:

Output gate

$$\mathbf{o}^{(t)} = \sigma \left(\boldsymbol{W}_o \mathbf{h}^{(t-1)} + \boldsymbol{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o \right)$$

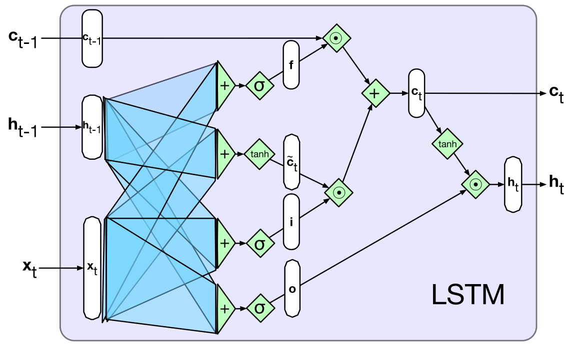
Updates are done as follows:

New cell state content $\tilde{\mathbf{c}}^{(t)} = \texttt{tanh} \left(\boldsymbol{W}_c \mathbf{h}^{(t-1)} + \boldsymbol{U}_c \mathbf{x}^{(t)} + \mathbf{b}_c \right)$.

Cell state $\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \cdot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \cdot \tilde{\mathbf{c}}^{(t)}$.

Hidden state $\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \cdot \texttt{tanh}(\mathbf{c}^{(t)})$.

Remark. LSTM makes it easier to preserve information over time, but it might still be affected by the vanishing gradient problem.



5.1.3 Gated recurrent units

Gated recurrent units (GRU) Architecture simpler than LSTM with fewer gates and without the cell state. Gated recurrent units (GRU)

Gates

Update gate Controls what part of the current hidden state to keep: Update gate

$$\mathbf{u}^{(t)} = \sigma\left(\boldsymbol{W}_u\mathbf{h}^{(t-1)} + \boldsymbol{U}_u\mathbf{x}^{(t)} + \mathbf{b}_u\right)$$

Reset gate Controls what part of the previous hidden state to use: Reset gate

$$\mathbf{r}^{(t)} = \sigma\left(\boldsymbol{W}_r\mathbf{h}^{(t-1)} + \boldsymbol{U}_r\mathbf{x}^{(t)} + \mathbf{b}_r\right)$$

Updates are done as follows:

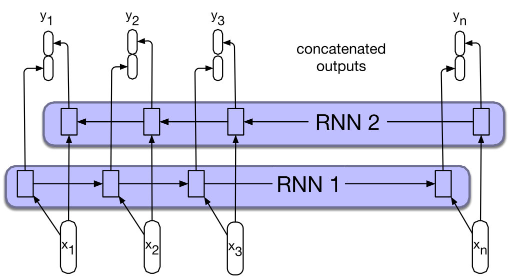
New hidden state content $\tilde{\mathbf{h}}^{(t)} = \texttt{tanh}\left(\boldsymbol{W}_h(\mathbf{r}^{(t)} \cdot \mathbf{h}^{(t-1)}) + \boldsymbol{U}_h\mathbf{x}^{(t)} + \mathbf{b}_h\right)$.

Hidden state $\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \cdot \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \cdot \tilde{\mathbf{h}}^{(t)}$.

Remark. Being faster to train than LSTMs, GRUs are usually a good starting point.

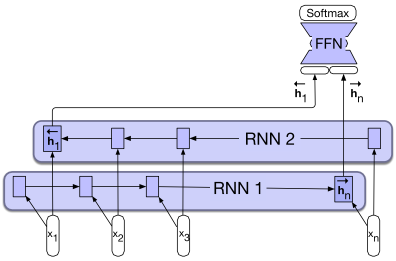
5.1.4 Bidirectional RNN

Bidirectional RNN Two independent RNNs (of any architecture) that processes the input left-to-right (forward) and right-to-left (backward), respectively. Usually, the output hidden state of a token t is obtained as the concatenation of the hidden states $\mathbf{h}^{(t)}_{\text{forward}}$ and $\mathbf{h}^{(t)}_{\text{backward}}$ of both networks. Bidirectional RNN



Remark. This architecture is not for language modelling (i.e., autoregressive models) as it is assumed that the whole input sequence is available at once.

Example (Sequence classification)**.** For sequence classification, the last hidden state of the forward and backward contexts can be used as the representation of the whole sequence to pass to the classifier.

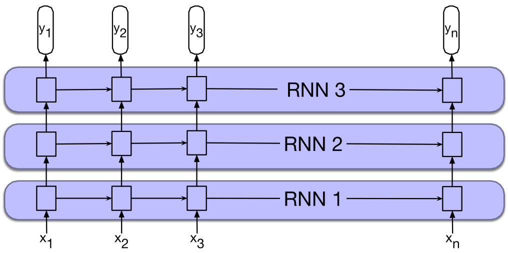


5.1.5 Stacked multi-layer RNN

Stacked RNN Stack of RNNs (of any architecture) where:

Stacked RNN

- The RNN at the first layer $l = 1$ processes the input tokens.
- The input of any following layer $l \geq 2$ is the hidden state $\mathbf{h}_{l-1}^{(t)}$ of the previous layer.



Remark. Skip connections between different layers can help to stabilize the gradient.

5.2 Applications

Autoregressive generation Repeatedly sample a token and feed it back to the network.

Autoregressive generation

Decoding strategy Method to select the output token from the output distribution. Possible approaches are:

Decoding strategy

Greedy Select the token with the highest probability.

Sampling Randomly sample the token following the probabilities of the output distribution.

Conditioned generation Provide an initial hidden state to the RNN (e.g., speech-to-text).

Conditioned generation

Sequence labelling Assign a class to each input token (e.g., POS-tagging, named-entity recognition, structure prediction, . . .).

Sequence labelling

Sequence classification Assign a class to the whole input sequence (e.g., sentiment analysis, document-topic classification, . . .).

Sequence classification

Sentence encoding Produce a vector representation for the whole input sequence. Possible approaches are:

Sentence encoding

- Use the final hidden state of the RNN.

- Aggregate all the hidden states (e.g., mean).

Example (Question answering)**.** The RNN encoder embeds the question that is used alongside the context (i.e., source from which the answer has to be extracted) to solve a labelling task (i.e., classify each token of the context as non-relevant or relevant).

6 Attention-based architectures

Sequence-to-sequence (seq2seq) model

Sequence-to-sequence (seq2seq) model Encoder-decoder architecture where:

Encoder Processes the whole input sequence and outputs a representation of it.

Decoder Processes the output of the encoder and produces the output sequence.

> **Remark.** Training is usually done using teacher forcing and averaging the loss of each output distribution.

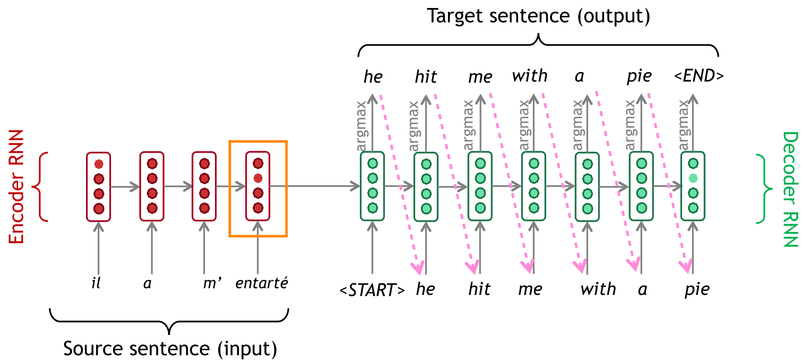


Figure 6.1: Example of seq2seq network with RNNs

6.1 Encoder-decoder RNN with attention

Seq2seq RNN with attention

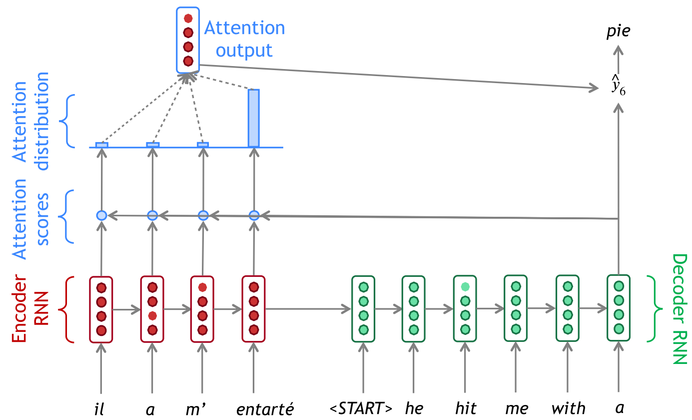
Seq2seq RNN with attention Architecture where the decoder can interact with each token processed by the encoder to determine dot-product attention scores (i.e., based on vector similarity).

The overall flow is the following:

1. The encoder computes its hidden states $\mathbf{h}^{(1)}, \dots, \mathbf{h}^{(N)} \in \mathbb{R}^h$.
2. The decoder processes the input tokens one at the time. Its hidden state is initialized with $\mathbf{h}^{(N)}$. Consider the token in position t, the output is determined as follows:
   a) The decoder outputs the hidden state $\mathbf{s}^{(t)}$.
   b) Attention scores $\mathbf{e}^{(t)}$ are determined as the dot product between $\mathbf{s}^{(t)}$ and $\mathbf{h}^{(i)}$:
   $$\mathbf{e}^{(t)} = \begin{bmatrix} \mathbf{s}^{(t)} \odot \mathbf{h}^{(1)} & \cdots & \mathbf{s}^{(t)} \odot \mathbf{h}^{(N)} \end{bmatrix} \in \mathbb{R}^N$$
   $\mathbf{e}^{(t)}$ is used to determine the attention distribution $\boldsymbol{\alpha}^{(t)}$ that is required to obtain the attention output $\mathbf{a}^{(t)}$ as the weighted encoder hidden states :
   $$\mathbb{R}^N \ni \boldsymbol{\alpha}^{(t)} = \texttt{softmax}(\mathbf{e}^{(t)})$$
   $$\mathbb{R}^h \ni \mathbf{a}^t = \sum_{i=1}^{N} \boldsymbol{\alpha}_i^{(t)} \mathbf{h}^{(i)}$$

c) The overall representation of the t-th token is the concatenation of the attention and decoder output:

$$\left[\mathbf{a}^{(t)} \mid \mathbf{s}^{(t)}\right] \in \mathbb{R}^{2h}$$



6.2 Convolutional neural networks for NLP

1D convolution (NLP) Apply a kernel on a sequence of tokens within a window. A kernel of size k with d-dimensional token embeddings is represented by a $k \times d$ weight matrix. 1D convolution

Remark. As in computer vision, multiple kernels can be stacked to increase the depth of the representation. Padding, stride, and dilation can also be used to change the receptive field. Pooling is also performed before passing to fully-connected layers.

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

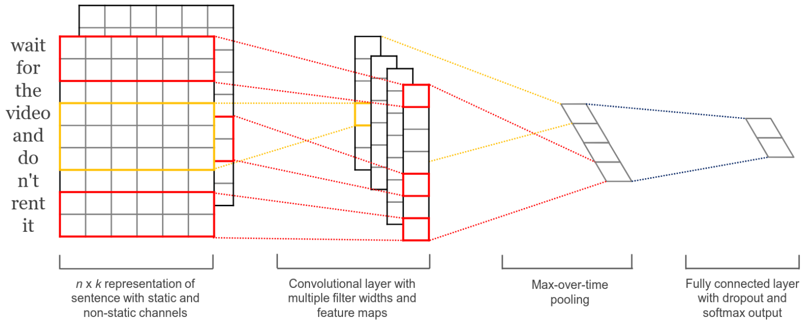
Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

Figure 6.2: Example of three 1D convolutions with padding 1

Remark. Convolutions are easy to parallelize.

Example (CNN for sentence classification)**.** A possible multichannel CNN architecture for sentence classification works as follows:

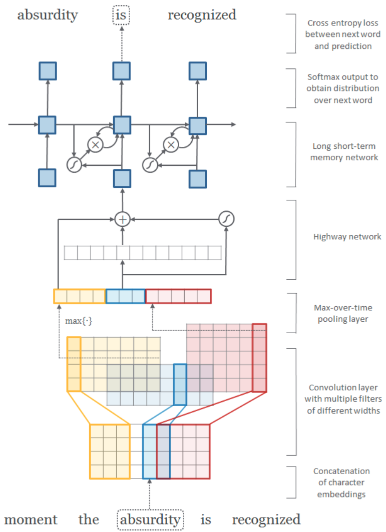
- The input sequence is encoded by stacking both static and learned embeddings (of the same dimensionality).
- Convolutions are applied to each channel. They can work with different widths.
- Pooling is used to flatten the activations and avoid shape mismatch before passing through fully-connected layers.



Example (Character-aware neural LM).

RNN-LM that works on a character level:

- Given a token, each character is embedded and concatenated.
- Convolutions are used to refine the representation.
- Pooling is used before passing the representation to the RNN.



6.3 Transformer decoder (for language modelling)

6.3.1 Self-attention

Self-attention Component that allows to compute the representation of a token considering the other ones in the input sequence. Self-attention

Given an input embedding $\mathbf{x}_i \in \mathbb{R}^{1 \times d_\text{model}}$, self-attention relies on the following values:

Queries Used as the reference point for attention: Queries

$$\mathbb{R}^{1 \times d_k} \ni \mathbf{q}_i = \mathbf{x}_i \boldsymbol{W}_Q$$

Keys Used as values to compare against the query: Keys

$$\mathbb{R}^{1 \times d_k} \ni \mathbf{k}_i = \mathbf{x}_i \boldsymbol{W}_K$$

Values Used to determine the output: Values

$$\mathbb{R}^{1 \times d_v} \ni \mathbf{v}_i = \mathbf{x}_i \boldsymbol{W}_V$$

where $\boldsymbol{W}_Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $\boldsymbol{W}_K \in \mathbb{R}^{d_\text{model} \times d_k}$, and $\boldsymbol{W}_V \in \mathbb{R}^{d_\text{model} \times d_v}$ are parameters.

Then, the attention weights $\alpha_{i,j}$ between two embeddings $\mathbf{x}_i$ and $\mathbf{x}_j$ are computed as:

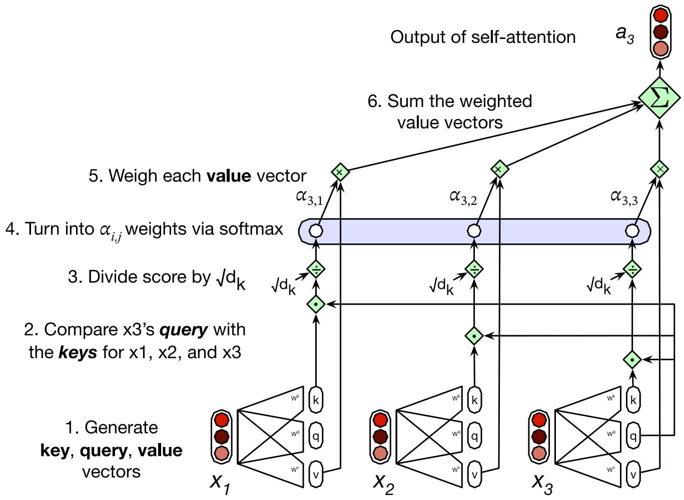
$$\texttt{scores}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{i,j} = \texttt{softmax}_j \left([\texttt{scores}(\mathbf{x}_i, \mathbf{x}_1), \dots, \texttt{scores}(\mathbf{x}_i, \mathbf{x}_T)] \right)$$

The output $\mathbf{a}_i \in \mathbb{R}^{1 \times d_v}$ is a weighted sum of the values of each token:

$$\mathbf{a}_i = \sum_t \alpha_{i,t} \mathbf{v}_t$$

To maintain the input dimension, a final projection $\boldsymbol{W}_O \in \mathbb{R}^{d_v \times d_{\text{model}}}$ is applied.



Causal attention Self-attention mechanism where only past tokens can be used to determine the representation of a token at a specific position. It is computed by modifying the standard self-attention as:

Causal attention

$$\forall j \leq i : \texttt{scores}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \mathbf{k}_j}{\sqrt{d_k}} \qquad \forall j > i : \texttt{scores}(\mathbf{x}_i, \mathbf{x}_j) = -\infty$$

$$\alpha_{i,j} = \texttt{softmax}_j \left([\texttt{scores}(\mathbf{x}_i, \mathbf{x}_1), \dots, \texttt{scores}(\mathbf{x}_i, \mathbf{x}_T)] \right)$$

$$\mathbf{a}_i = \sum_{t:t \leq i} \alpha_{i,t} \mathbf{v}_t$$

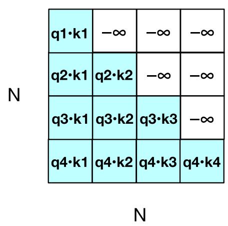
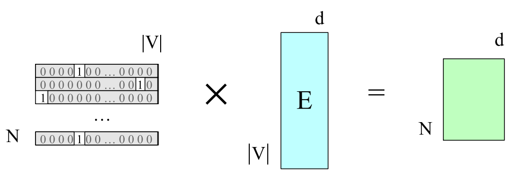


Figure 6.3: Score matrix with causal attention

6.3.2 Components

Input embedding The input is tokenized using standard tokenizers (e.g., BPE, SentencePiece, ...). Each token is encoded using a learned embedding matrix.

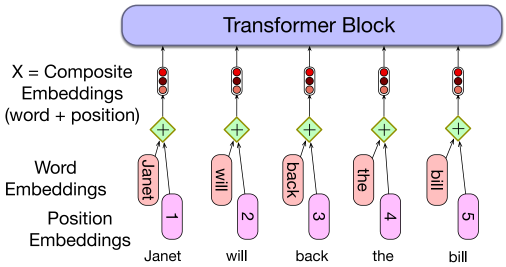
Input embedding



Positional encoding Learned position embeddings to encode positional information are added to the input token embeddings.

Positional encoding

> **Remark.** Without positional encoding, transformers are invariant to permutations.

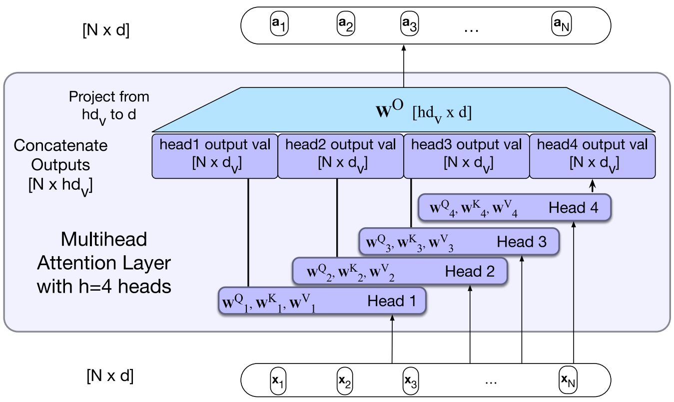


Transformer block Module with the same input and output dimensionality (i.e., allows stacking multiple blocks) composed of:

Transformer block

Multi-head attention Uses h different self-attention blocks with different queries, keys, and values. Value vectors are of size $\frac{d_v}{h}$. The final projection W_O is applied on the concatenation of the outputs of each head.

Multi-head attention



Feedforward layer Fully-connected 2-layer network applied at each position of the attention output:

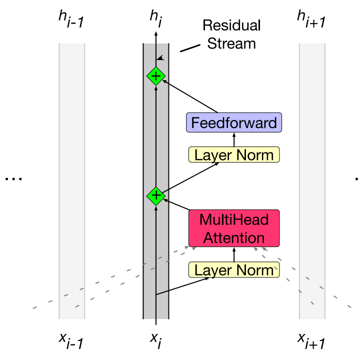
$$\texttt{FFN}(\mathbf{x}_i) = \texttt{ReLU}(\mathbf{x}_i \boldsymbol{W}_1 + \mathbf{b}_1)\boldsymbol{W}_2 + \mathbf{b}_2$$

Where the hidden dimension d_{ff} is usually larger than d_{model}.

Normalization layer Applies token-wise normalization (i.e., layer norm) to help training stability.

Residual connection Helps to propagate information during training.

> **Remark** (Residual stream)**.** An interpretation of residual connections is the residual stream where the input token in enhanced by the output of multi-head attention and the feedforward network.



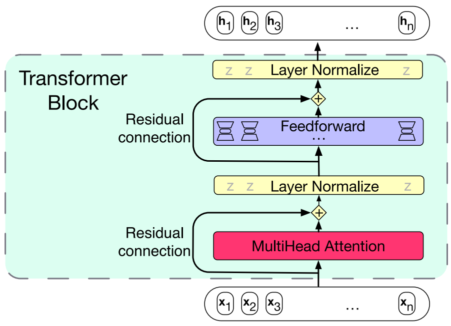
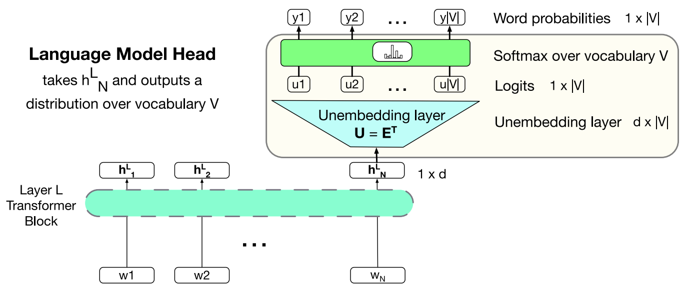


Figure 6.4: Overall attention block

Language modelling head Takes as input the output corresponding to a token of the transformer blocks stack and outputs a distribution over the vocabulary.

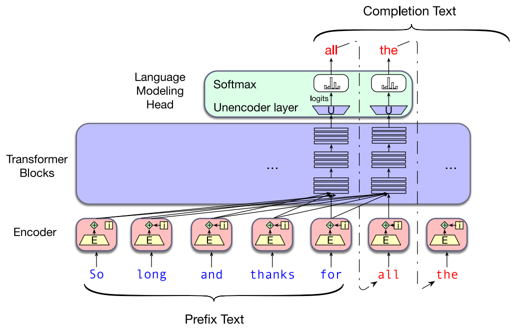
Language modelling head



7 Large language models

7.1 Decoder-only architecture

Conditional generation Generate text conditioned on the input tokens (i.e., prompt). Conditional generation



Example (Sentiment analysis). Given the prompt:

$$p = \texttt{the sentiment of the sentence 'I like Jackie Chan' is}$$

Determine the probability of the tokens `positive` and `negative`:

$$\mathcal{P}(\texttt{positive} \mid p) \qquad \mathcal{P}(\texttt{negative} \mid p)$$

Example (Question answering). Given the prompt:

$$p = \texttt{Q: who wrote the book 'The origin of Species'? A:}$$

Determine the tokens of the answer autoregressively:

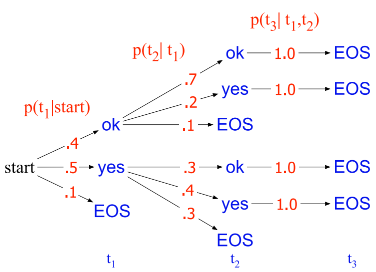
$$\arg\max_{w_1} \mathcal{P}(w_1 \mid p), \arg\max_{w_2} \mathcal{P}(w_2 \mid pw_1), \dots$$

7.1.1 Decoding strategies

Greedy decoding Select the next token as the most probable of the output distribution. Greedy decoding

Remark. Greedy decoding risks getting stuck in a local optimum.

Example. Consider the following search tree of possible generated sequences:

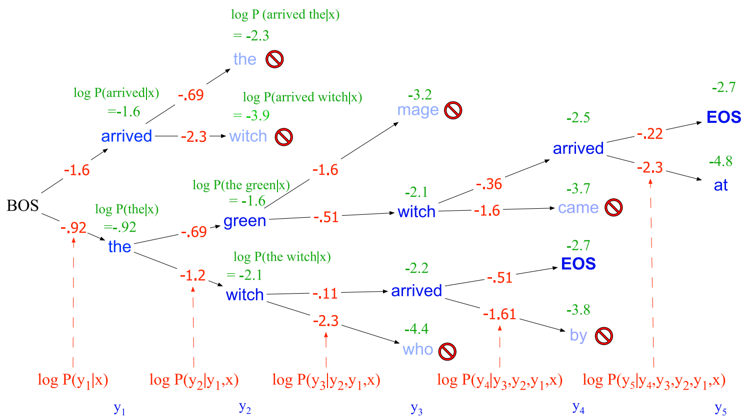


Greedy search would select the sequence `yes yes` which has probability $0.5 \cdot 0.4 = 0.2$. However, the sequence `ok ok` has a higher probability of $0.4 \cdot 0.7 = 0.28$.

Beam search Given a beam width k, perform a breadth-first search keeping at each branching level the top-k tokens based on the probability of that sequence: Beam search

$$\log\left(\mathcal{P}\left(y \mid x\right)\right) = \sum_{i=1}^{t} \log\left(\mathcal{P}\left(y_i \mid x, y_1, \dots, y_{i-1}\right)\right)$$

Example. Consider the following tree with beam width $k = 2$:



The selected sequence is `[BOS] the green witch arrived [EOS]`.

Remark. As each path might generate sequences of different length, the score is usually normalized by the number of tokens as:

$$\log\left(\mathcal{P}\left(y \mid x\right)\right) = \frac{1}{t}\sum_{i=1}^{t} \log\left(\mathcal{P}\left(y_i \mid x, y_1, \dots, y_{i-1}\right)\right)$$

Remark. The likelihood of the sequences generated using beam search is higher than using greedy decoding. However, beam search is still not optimal.

Sampling Sample the next token based on the output distribution. Sampling

Random sampling Sample considering the distribution over the whole vocabulary.

Remark. By adding-up all the low-probability words (which are most likely unreasonable as the next token), their actual chance of getting selected is relatively high.

Temperature sampling Skew the distribution to emphasize the most likely words and decrease the probability of less likely words. Given the logits $\mathbf{u}$ and the temperature τ, the output distribution $\mathbf{y}$ is determined as:

$$\mathbf{y} = \texttt{softmax}\left(\frac{\mathbf{u}}{\tau}\right)$$

where:

- Higher temperatures (i.e., $\tau > 1$) allow for considering low-probability words.
- Lower temperatures (i.e., $\tau \in (0, 1]$) focus on high-probability words.

Remark. A temperature of $\tau = 0$ corresponds to greedy decoding.

Top-k sampling Consider the top-k most probable words and apply random sampling on their normalized distribution.

| **Remark.** $k = 1$ corresponds to greedy decoding.

| **Remark.** k is fixed and does not account for the shape of the distribution.

Top-p/nucleus sampling Consider the most likely words such that their probability mass adds up to p. Then, apply random sampling on their normalized distribution.

7.1.2 Pre-training

Pre-training Use self-supervision and teacher forcing to train the whole context window in parallel on a large text corpus. Pre-training

Remark. Results are highly dependent on the training corpora. Important aspects to consider are:

Language Most of the available data is in English.

Data quality Prefer high-quality sources such as Wikipedia or books. Boilerplate removal and deduplication might be needed.

Safety filtering Toxicity removal might be needed

Ethical and legal issues Use of copyrighted material, permission from data owners, use of private information, . . .

Scaling laws Empirical laws that put in relationship: Scaling laws

- Non-embedding parameters N ($N \approx 2d_{\text{model}} n_{\text{layer}}(2d_{\text{attention}} + d_{\text{ff}})$),
- Training data size D,
- Compute budget C (i.e., training iterations).

By keeping two of the three factors constant, the loss $\mathcal{L}$ of an LLM can be estimated as a function of the third variable:

$$\mathcal{L}(N) = \left(\frac{N_c}{N}\right)^{\alpha_N} \qquad \mathcal{L}(D) = \left(\frac{D_c}{D}\right)^{\alpha_D} \qquad \mathcal{L}(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}$$

where N_c, D_c, C_c, α_N, α_D, and α_C are constants determined empirically based on the model architecture.

7.1.3 Fine-tuning

Fine-tuning Specialize an LLM to a specific domain or task. Fine-tuning

Continued pre-training Continue pre-training with a domain-specific corpus. Continued pre-training

Model adaptation Specialize a model by adding new learnable parameters.

Parameter-efficient fine-tuning (PEFT) Continue training a selected subset of parameters (e.g., LoRA Section 8.1). Parameter-efficient fine-tuning (PEFT)

Task-specific fine-tuning Add a new trainable head on top of the model. Task-specific fine-tuning

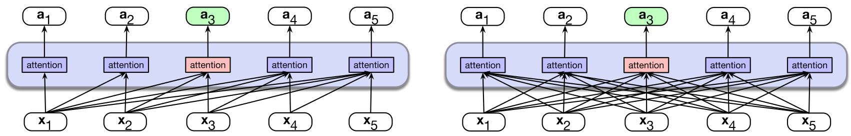
Supervised fine-tuning Continue training using a supervised dataset to align the model to human's expectation. Supervised fine-tuning

7.2 Encoder-only architecture

Transformer encoder Architecture that produces contextual embeddings by considering both left-to-right and right-to-left context. Transformer encoder

Remark. This architecture does feature extraction and is more suited for classification tasks.

Architecture Similar to a transformer decoder, but self-attention is not causal.



a) A causal self-attention layer

b) A bidirectional self-attention layer

Contextual embedding Represent the meaning of word instances (i.e., dynamically depending on the surroundings). Contextual embedding

Remark (Sequence embedding)**.** Encoders usually have a classifier token (e.g., `[CLS]`) to model the whole sentence.

Example (Word sense disambiguation)**.** Task of determining the sense of each word of a sequence. Senses usually come from an existing ontology (e.g., WordNet). An approach to solve the problem is the following:

1. Compute the embedding $\mathbf{v}_i$ of words using a pre-trained encoder (e.g., BERT).
2. Represent the embedding of a sense as the average of the tokens of that sense:

$$\mathbf{v}_s = \frac{1}{n} \sum_i \mathbf{v}_i$$

3. Predict the sense of a word $\mathbf{t}$ as:

$$\arg \max_{s \in \texttt{senses}(\mathbf{t})} \texttt{distance}(\mathbf{t}, \mathbf{v}_s)$$

Tokenizer fertility Average amount of tokens used to represent words. Tokenizer fertility

Remark. Tokenizer fertility is relevant for inference speed.

Curse of multilinguality The performance of each language of a multilingual model tend to be worse than its monolingual counterpart. Curse of multilinguality

7.2.1 Pre-training

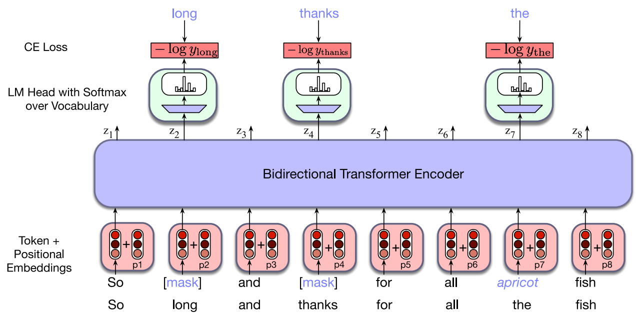
Masked language modelling Task of predicting missing or corrupted tokens in a sequence. Masked language modelling

Remark. Transformer encoders output embeddings. For training purposes, a head to output a distribution over the vocabulary is added.

Example. Given a training corpus, BERT is trained by randomly sampling 15% of the tokens in the training data and either:

- Mask it with a special `[MASK]` token (80% of the time).
- Replace it with a different token (10% of the time).

- Do nothing (10% of the time).



Remark. BERT's training approach is inefficient as masks are determined before training and only 15% of the corpus tokens are actually used for training. Other models (e.g., RoBERTa), dynamically determine the mask at training time, allowing for more variety.

Span masking Mask contiguous spans of words to obtain a harder training objective. Span masking

Remark. This approach generally produces better embeddings.

7.2.2 Fine-tuning

Fine-tuning for classification Add a classification head on top of the classifier token.

Fine-tuning for sequence-pair classification Use a model pre-trained to process pair of sequences. This is usually done by means of a special separator token (e.g., `[SEP]` in BERT).

Fine-tuning for sequence labeling Add a classification head on top of each token. A conditional random field (CRF) layers can also be added to produce globally more coherent tags.

Named entity recognition (NER) Task of assigning to each word of a sequence its entity class. NER taggers usually also capture concepts spanning across multiple tokens. To achieve this, additional information is provided with the entity class: Named entity recognition (NER)

Begin Starting token of a concept.

Inside Token belonging to the same span of the previous one.

End Last token of a span.

Outside Token outside the scope of the tagger.

Metrics

Recall $\frac{\text{Correctly labeled responses}}{\text{Total that should have been labeled}}$

Precision $\frac{\text{Correctly labeled responses}}{\text{Total that has been labeled}}$

Remark. The entity (so, also a span of text) is the atomic unit for NER metrics.

Remark (GLUE)**.** The General Language Understanding Evaluation (GLUE) benchmark is a common set of tasks used to evaluate natural language understanding models. It comprises tasks based on single sentences, multiple sentences, and inference from a sequence.

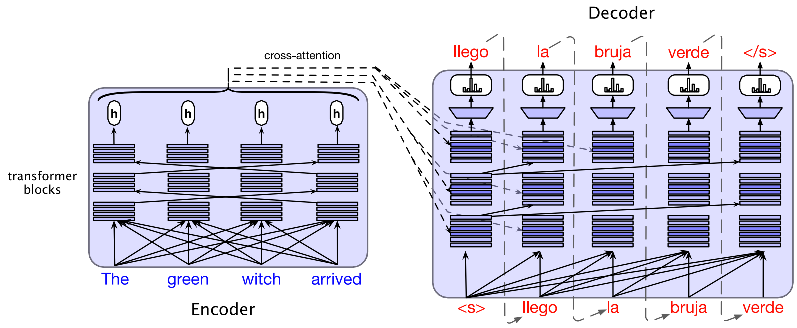
7.3 Encoder-decoder architecture

Encoder-decoder architecture Model with both an encoder and decoder:

Encoder-decoder architecture

- **Encoder** Architecture as presented in Section 7.2. Its result is used to condition the output of the decoder.
- **Decoder** Architecture similar to the one presented in Section 7.1 with an additional cross-attention layer inserted before causal attention.
  - **Cross-attention** Attention layer that uses the output of the encoder as keys and values, while the query is from the decoder.

Cross-attention



7.3.1 Pre-training

Span corruption Given an input sequence, replace different-length spans of text with a unique placeholder. The encoder takes as input the corrupted sequence, while the decoder has to predict the missing words.

Span corruption

> **Remark.** It has been observed that targeted span masking works better compared to random span masking.

> **Example.** Given the sequence:
>
> ```
> <bos> thank you for inviting me to your party last week <eos>
> ```
>
> Some spans of text are masked with placeholder tokens as follows:
>
> ```
> <bos> thank you <X> me to your party <Y> week <eos>
> ```
>
> The masked sequence is passed through the encoder, while the decoder has to predict the masked tokens:
>
> ```
> <bos> <X> for inviting <Y> last <Z> <eos>
> ```

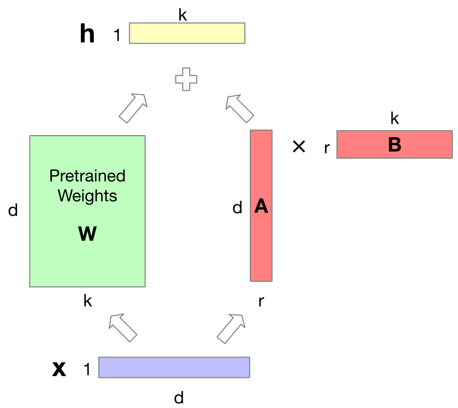
8 Efficient model utilization

8.1 Low-rank adaptation

Low-rank adaptation (LoRA) Method to update weights by learning an offset that uses fewer parameters. Low-rank adaptation (LoRA)

Consider a weight matrix $\boldsymbol{W} \in \mathbb{R}^{d \times k}$, LoRA decomposes the update into two learnable matrices $\boldsymbol{A} \in \mathbb{R}^{d \times r}$ and $\boldsymbol{B} \in \mathbb{R}^{r \times k}$ (with $r \ll d, k$). Weights update is performed as:

$$\boldsymbol{W}_{\text{fine-tuned}} = \boldsymbol{W}_{\text{pre-trained}} + \boldsymbol{AB}$$



8.2 Model compression

8.2.1 Parameters compression

Parameter sharing Use the same parameters between layers. Parameter sharing

Pruning Remove weights with small impact on the loss. Pruning

> **Remark.** Dropping some weights produce sparse matrices that are unoptimized for parallel hardware. Therefore, this approach does not always improve efficiency.

Quantization Store and perform operations with lower precision floating-points (e.g., FP32 to FP4). Quantization

8.2.2 Training compression

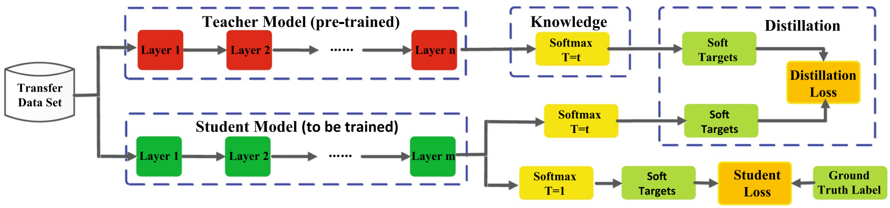
Mixture of experts Specialize smaller models on subset of data and train a router to forward the input to the correct expert. Mixture of experts

> **Remark.** This approach can be easily deployed on distributed systems.

Knowledge distillation Train a student model to emulate the teacher's hidden states. In a general setting, the output distribution of the teacher is used to create the student. Two losses are used: Knowledge distillation

Distillation loss Matches the output distribution of the student to the one of the teacher. A softmax with higher temperature is usually used so that training contribution does not only come from the highest probability.

Student loss Matches the output distribution of the student with the ground-truth (i.e., same loss of the training task).



Vocabulary transfer Use a domain-specific tokenizer to reduce the number of tokens to represent complex/domain-specific words and reduce the size of the embedding matrix.

Vocabulary transfer

Fast vocabulary transfer (FVT) Given:

Fast vocabulary transfer (FVT)

- A starting embedding model with tokenizer $\mathcal{T}_\text{s}$, vocabulary V_s, and embedding matrix $\boldsymbol{E}_\text{s}$,
- A new tokenizer $\mathcal{T}_\text{dom}$ trained on a domain-specific corpus,

The embedding matrix $\boldsymbol{E}_\text{dom}$ for the vocabulary V_dom of $\mathcal{T}_\text{dom}$ is built as follows:

$$\forall t_i \in V_\text{dom} : \boldsymbol{E}_\text{dom}(t_i) = \frac{1}{|\mathcal{T}_\text{s}(t_i)|} \sum_{t_j \in \mathcal{T}_\text{s}(t_i)} \boldsymbol{E}_\text{s}(t_j)$$

In other words, each token in V_dom is encoded as the average of embeddings of the tokens that compose it in the starting embedding model (if the token appear in both vocabularies, the embedding is the same).

8.3 In-context learning

Prompting Pass a prompt to the language model to condition generation.

Prompting

More formally, a prompt is defined by means of a prompting function $f_\text{prompt}(\cdot)$ that formats an input text x. f_prompt typically has a slot for the input and a slot for the answer (e.g., the class in case of classification). The prompt is then fed to the language model that searches the highest scoring word $\hat{z}$ to fill the answer as follows:

$$\hat{z} = \arg\max \mathcal{P}\left(f_\text{fill}(f_\text{prompt}(x), z); \theta\right)$$

Where $f_\text{fill}(f_\text{prompt}(x), z)$ inserts z in the prompt. In other word, we are looking for the word that makes the model least perplexed.

Example. A prompt for sentiment analysis of movie reviews might be:

```
[X] Overall, it was a [Z] movie.
```

Where `[X]` is the placeholder for the review and `[Z]` is for the class.

Remark. The prompt does not necessarily need to be text (i.e., discrete/hard prompts). Continuous/soft prompts (i.e., embeddings) can also be used to condition generation.

Zero-shot learning Solve a task by providing a language model the description of the problem in natural language. Zero-shot learning

One-shot learning Solve a task by providing a language model the description of the problem in natural language and a single demonstration (i.e., an example). One-shot learning

Few-shot learning Solve a task by providing a language model the description of the problem in natural language and a few demonstrations. Few-shot learning

> **Remark.** Empirical results show that not too many examples are required. Also, too many examples might reduce performance.

Remark. Some studies show that an explanation for in-context learning is that causal attention has the same effect of gradient updates (i.e., the left part of the prompt influences the right part).
Another possible explanation is based on the concept of induction heads which are attention heads that specialize in predicting repeated sequences (i.e., in-context learning is seen as the capability of imitating past data). Ablation studies show that by identifying and removing induction heads, in-context learning performance of a model drastically drops.

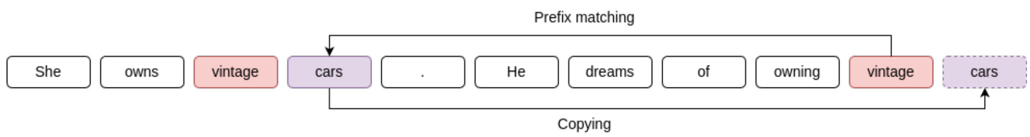
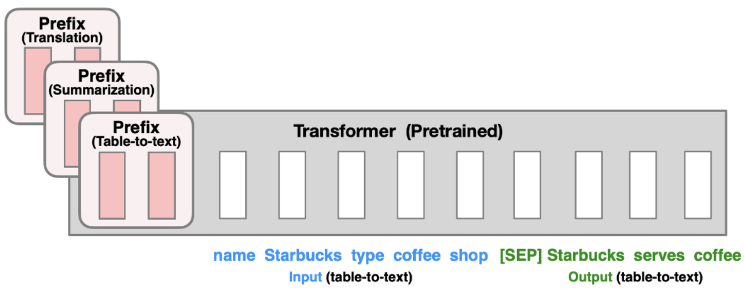


Figure 8.1: Example of induction head

Prefix-tuning Soft prompting technique that learns some prefix embeddings for a specific task to add to the prompt while keeping the rest of the model frozen. Prefix-tuning



Chain-of-thought prompting Provide in the prompt examples of reasoning to make the model provide the output step-by-step. Chain-of-thought prompting

> **Remark.** Empirical results show that the best prompt for chain-of-thought is to add to the prompt `think step by step`.

9 Language model alignment and applications

9.1 Model alignment

Remark. Off-the-shelf pre-trained models tend to only be good at word completion. They are most likely unable to understand instructions and might generate harmful content.

9.1.1 Instruction tuning

Instruction tuning Fine-tune a model on a dataset containing various tasks expressed in natural language in the form (description, examples, solution), all possibly formatted using multiple templates.

Instruction tuning

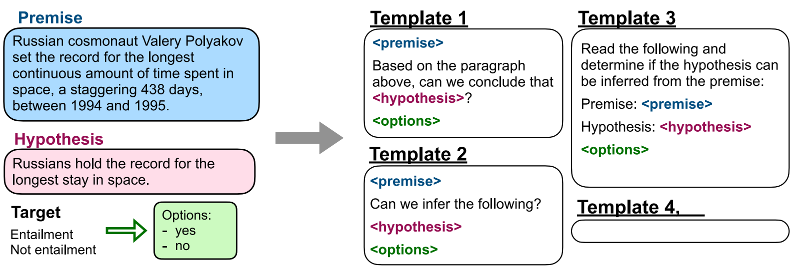


Figure 9.1: Example of templates for entailment detection

Remark. If performed correctly, after performing instruction tuning on a model, it is able to also solve tasks that were not present in the tuning dataset.

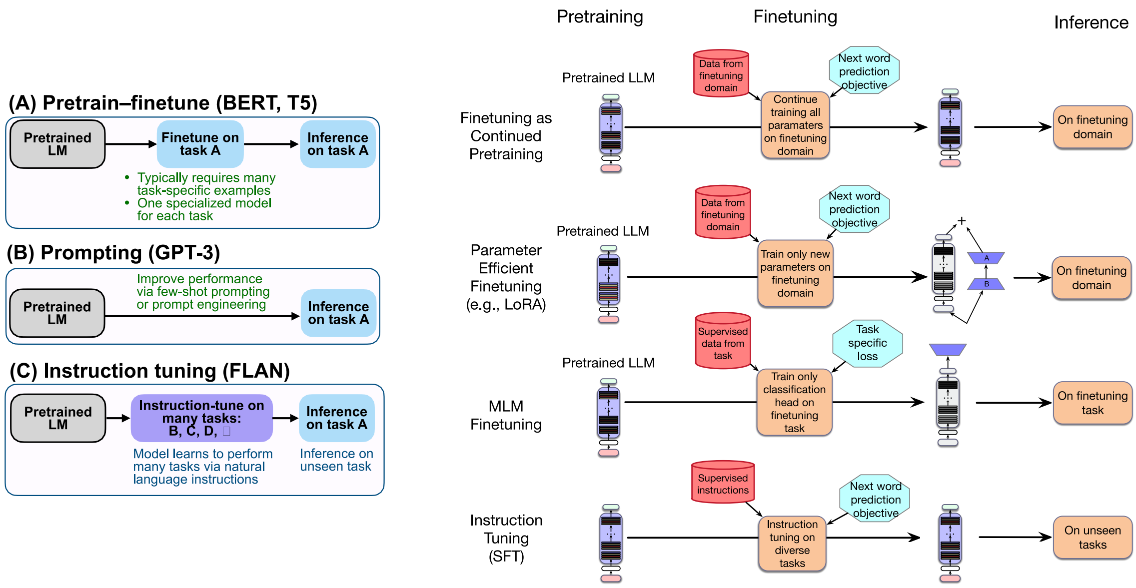


Figure 9.2: Comparison of tuning approaches

9.1.2 Preference alignment

Preference alignment Align the output of a model with human values.

Preference alignment

Reinforcement learning with human feedback (RLHF) Align a language model using a policy-gradient reinforcement learning algorithm. The problem can be formulated as follows:

Reinforcement learning with human feedback (RLHF)

- The policy to learn represents the aligned model (i.e., `prompt` $\mapsto$ `answer` model),
- Prompts are the states,
- Answers are the actions.

RLHF works as follows:

1. Start from a pre-trained language model that already works well.
2. Train a reward model r_θ from a human-annotated dataset that maps text sequences into rewards. The architecture is usually based on transformers.
3. Fine-tune the language model (i.e., train the policy) using an RL algorithm (e.g., PPO) and the learned reward model.

   Given a prompt x and an answer y, the reward r used for the RL update is computed as:

$$r = r_\theta(y \mid x) - \lambda_{\text{KL}} D_{\text{KL}}(\pi_{\text{PPO}}(y \mid x) \| \pi_{\text{base}}(y \mid x))$$

   where:

   - $r_\theta(y \mid x)$ is the reward provided by the reward model.
   - $-\lambda_{\text{KL}} D_{\text{KL}}(\pi_{\text{PPO}}(y \mid x) \| \pi_{\text{base}}(y \mid x))$ is a penalty based on the Kullback-Leibler divergence to prevent the aligned model π_{PPO} from moving too away from the original model π_{base} (i.e., prevent the loss of language capabilities).